Enterprise Systems Architecture/390



# Principles of Operation

SA22-7201-08

LzLabs GmbH. Ex. 1008-1

Enterprise Systems Architecture/390



# Principles of Operation

SA22-7201-08

LzLabs GmbH. Ex. 1008-3

**Note:**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xvii.

**Softcopy Note:**

The reader should be aware of the fact that this publication contains many symbols, such as superscripts, that may not display correctly with any given hardware or software. The definitive version of this publication is the hardcopy version.

**Ninth Edition (June, 2003)**

This edition obsoletes and replaces *Enterprise Systems Architecture/390 Principles of Operation*, SA22-7201-07.

This publication is provided for use in conjunction with other relevant IBM publications, and IBM makes no warranty, express or implied, about its completeness or accuracy. The information in this publication is current as of its publication date but is subject to change without notice.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

International Business Machines Corporation
Department 55JA Mail Station P384
2455 South Road
Poughkeepsie, N.Y., 12601-5400
United States of America

FAX (United States & Canada): 1-845-432-9405
FAX (Other Countries): Your International Access Code + 1-845-432-9405
IBMLink (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1990-2003. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

LzLabs GmbH. Ex. 1008-4

# Contents


**Notices** . . . . . xvii
Trademarks . . . . . xvii

**Preface** . . . . . xix
Size and Number Notation . . . . . xxi
Bytes, Characters, and Codes . . . . . xxi
Other Publications . . . . . xxi
| Summary of Changes in Ninth Edition . . . . xxii
Summary of Changes in Eighth Edition . . . xxiii
Summary of Changes in Seventh Edition . . xxv
Summary of Changes in Sixth Edition . . . . xxvi
Summary of Changes in Fifth Edition . . . . . xxvii
Summary of Changes in Fourth Edition . . xxviii
Summary of Changes in Third Edition . . . xxviii
Summary of Changes in Second Edition . . . xxx

**Chapter 1. Introduction** . . . . . 1-1
Highlights of ESA/390 . . . . . 1-1
The ESA/370 and 370-XA Base . . . . . 1-10
System Program . . . . . 1-12
Compatibility . . . . . 1-12
Compatibility among ESA/390 Systems . . 1-12
Compatibility among ESA/390, ESA/370, 370-XA, and System/370 . . . . . 1-13
Control-Program Compatibility . . . . . 1-13
Problem-State Compatibility . . . . . 1-13
Availability . . . . . 1-14

**Chapter 2. Organization** . . . . . 2-1
Main Storage . . . . . 2-2
Expanded Storage . . . . . 2-2
CPU . . . . . 2-2
PSW . . . . . 2-3
General Registers . . . . . 2-3
Floating-Point Registers . . . . . 2-3
Floating-Point-Control (FPC) Register . . . 2-4
Control Registers . . . . . 2-4
Access Registers . . . . . 2-4
Vector Facility . . . . . 2-6
Cryptographic Facility . . . . . 2-6
External Time Reference . . . . . 2-6
I/O . . . . . 2-6
Channel Subsystem . . . . . 2-6
Channel Paths . . . . . 2-6
I/O Devices and Control Units . . . . . 2-7
Operator Facilities . . . . . 2-7

**Chapter 3. Storage** . . . . . 3-1
Storage Addressing . . . . . 3-2
Information Formats . . . . . 3-2
Integral Boundaries . . . . . 3-3
Address Types and Formats . . . . . 3-3
Address Types . . . . . 3-3
Absolute Address . . . . . 3-4
Real Address . . . . . 3-4
Virtual Address . . . . . 3-4
Primary Virtual Address . . . . . 3-4
Secondary Virtual Address . . . . . 3-4
AR-Specified Virtual Address . . . . . 3-4
Home Virtual Address . . . . . 3-4
Logical Address . . . . . 3-4
Instruction Address . . . . . 3-5
Effective Address . . . . . 3-5
Address Size and Wraparound . . . . . 3-5
Address Wraparound . . . . . 3-5
Storage Key . . . . . 3-8
Protection . . . . . 3-8
Key-Controlled Protection . . . . . 3-9
Storage-Protection-Override Control . . 3-10
Fetch-Protection-Override Control . . . 3-11
Access-List-Controlled Protection . . . . 3-11
Page Protection . . . . . 3-11
Low-Address Protection . . . . . 3-12
Suppression on Protection . . . . . 3-12
Reference Recording . . . . . 3-14
Change Recording . . . . . 3-14
Prefixing . . . . . 3-15
Address Spaces . . . . . 3-16
Changing to Different Address Spaces . 3-17
Address-Space Number . . . . . 3-17
ASN Translation . . . . . 3-18
ASN-Translation Controls . . . . . 3-18
Control Register 14 . . . . . 3-18
Control Register 0 . . . . . 3-19
ASN-Translation Tables . . . . . 3-19
ASN-First-Table Entries . . . . . 3-19
ASN-Second-Table Entries . . . . . 3-19
ASN-Translation Process . . . . . 3-21
ASN-First-Table Lookup . . . . . 3-21
ASN-Second-Table Lookup . . . . . 3-22
Recognition of Exceptions during ASN Translation . . . . . 3-23
ASN Authorization . . . . . 3-23
ASN-Authorization Controls . . . . . 3-23
Control Register 4 . . . . . 3-24
ASN-Second-Table Entry . . . . . 3-24
Authority-Table Entries . . . . . 3-24
ASN-Authorization Process . . . . . 3-24
Authority-Table Lookup . . . . . 3-25


© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-5

Recognition of Exceptions during ASN Authorization . . . 3-26
Dynamic Address Translation . . . 3-26
Translation Control . . . 3-27
Translation Modes . . . 3-28
Control Register 0 . . . 3-28
Control Register 1 . . . 3-28
Control Register 7 . . . 3-29
Control Register 13 . . . 3-29
Translation Tables . . . 3-30
Segment-Table Entries . . . 3-30
Page-Table Entries . . . 3-30
Summary of Segment-Table and Page-Table Sizes . . . 3-31
Translation Process . . . 3-31
Effective Segment-Table Designation . 3-32
Inspection of Control Register 0 . . . 3-34
Segment-Table Lookup . . . 3-34
Page-Table Lookup . . . 3-35
Formation of the Real Address . . . 3-35
Recognition of Exceptions during Translation . . . 3-35
Translation-Lookaside Buffer . . . 3-35
TLB Structure . . . 3-36
Formation of TLB Entries . . . 3-36
Use of TLB Entries . . . 3-37
Modification of Translation Tables . . . 3-38
Address Summary . . . 3-40
Addresses Translated . . . 3-40
Handling of Addresses . . . 3-40
Assigned Storage Locations . . . 3-43

**Chapter 4. Control** . . . 4-1
Stopped, Operating, Load, and Check-Stop States . . . 4-1
Stopped State . . . 4-2
Operating State . . . 4-2
Load State . . . 4-2
Check-Stop State . . . 4-3
Program-Status Word . . . 4-3
Program-Status-Word Format . . . 4-5
Control Registers . . . 4-6
Tracing . . . 4-10
Control-Register Allocation . . . 4-10
Trace Entries . . . 4-11
Operation . . . 4-14
Program-Event Recording . . . 4-14
Control-Register Allocation and Segment-Table Designation . . . 4-15
Operation . . . 4-16
Identification of Cause . . . 4-17
Priority of Indication . . . 4-20
Storage-Area Designation . . . 4-21
PER Events . . . 4-22
Successful Branching . . . 4-22
Instruction Fetching . . . 4-22
Storage Alteration . . . 4-22
General-Register Alteration . . . 4-23
Store Using Real Address . . . 4-24
Indication of PER Events Concurrently with Other Interruption Conditions . . . 4-24
Timing . . . 4-29
Time-of-Day Clock . . . 4-29
Format . . . 4-29
States . . . 4-30
Changes in Clock State . . . 4-31
Setting and Inspecting the Clock . . . 4-31
TOD Programmable Register . . . 4-32
TOD-Clock Synchronization . . . 4-34
Clock Comparator . . . 4-35
CPU Timer . . . 4-35
Externally Initiated Functions . . . 4-37
Resets . . . 4-37
CPU Reset . . . 4-40
Initial CPU Reset . . . 4-41
Subsystem Reset . . . 4-41
Clear Reset . . . 4-42
Power-On Reset . . . 4-42
Initial Program Loading . . . 4-43
Store Status . . . 4-43
Multiprocessing . . . 4-44
Shared Main Storage . . . 4-44
CPU-Address Identification . . . 4-45
CPU Signaling and Response . . . 4-45
Signal-Processor Orders . . . 4-45
Conditions Determining Response . . . 4-50
Conditions Precluding Interpretation of the Order Code . . . 4-50
Status Bits . . . 4-51

**Chapter 5. Program Execution** . . . 5-1
Instructions . . . 5-2
Operands . . . 5-3
Instruction Formats . . . 5-3
Register Operands . . . 5-6
Immediate Operands . . . 5-6
Storage Operands . . . 5-6
Address Generation . . . 5-7
Bimodal Addressing . . . 5-7
Sequential Instruction-Address Generation . 5-7
Operand-Address Generation . . . 5-8
Formation of the Intermediate Value . . 5-8
Formation of the Operand Address . . . 5-8
Branch-Address Generation . . . 5-9
Formation of the Intermediate Value . . 5-9
Formation of the Branch Address . . . 5-9
Instruction Execution and Sequencing . . . 5-9
Decision Making . . . 5-10



LzLabs GmbH. Ex. 1008-6

Loop Control . . . . . 5-10
Subroutine Linkage without the Linkage Stack . . . . . 5-10
Interruptions . . . . . 5-16
Types of Instruction Ending . . . . . 5-16
Completion . . . . . 5-16
Suppression . . . . . 5-16
Nullification . . . . . 5-17
Termination . . . . . 5-17
Interruptible Instructions . . . . . 5-17
Point of Interruption . . . . . 5-17
Unit of Operation . . . . . 5-17
Execution of Interruptible Instructions . 5-17
Condition-Code Alternative to Interruptibility . . . . . 5-19
Exceptions to Nullification and Suppression . . . . . 5-19
Storage Change and Restoration for DAT-Associated Access Exceptions 5-20
Modification of DAT-Table Entries . . . 5-20
Trial Execution for Editing Instructions and Translate Instruction . . . . . 5-21
Authorization Mechanisms . . . . . 5-21
Mode Requirements . . . . . 5-21
Extraction-Authority Control . . . . . 5-22
PSW-Key Mask . . . . . 5-22
Secondary-Space Control . . . . . 5-22
Subsystem-Linkage Control . . . . . 5-22
ASN-Translation Control . . . . . 5-22
Authorization Index . . . . . 5-23
Program-Call-Fast Control . . . . . 5-23
Access-Register and Linkage-Stack Mechanisms . . . . . 5-23
PC-Number Translation . . . . . 5-27
PC-Number Translation Control . . . . . 5-27
Control Register 0 . . . . . 5-27
Control Register 5 . . . . . 5-27
PC-Number Translation Tables . . . . . 5-28
Linkage-Table Entries . . . . . 5-28
Entry-Table Entries . . . . . 5-28
PC-Number-Translation Process . . . . . 5-30
Obtaining the Linkage-Table Designation . . . . . 5-31
Linkage-Table Lookup . . . . . 5-32
Entry-Table Lookup . . . . . 5-32
Recognition of Exceptions during PC-Number Translation . . . . . 5-32
Home Address Space . . . . . 5-33
Access-Register Introduction . . . . . 5-33
Summary . . . . . 5-34
Access-Register Functions . . . . . 5-34
Access-Register-Specified Address Spaces . . . . . 5-34
Access-Register Instructions . . . . . 5-41
Access-Register Translation . . . . . 5-42
Access-Register-Translation Control . . . . 5-42
Address-Space-Function Control . . . . 5-42
Control Register 2 . . . . . 5-43
Control Register 5 . . . . . 5-43
Control Register 8 . . . . . 5-43
Access Registers . . . . . 5-43
Access-Register-Translation Tables . . . . 5-44
Dispatchable-Unit-Control Table and Access-List Designations . . . . . 5-44
Access-List Entries . . . . . 5-46
Extended ASN-Second-Table Entries . 5-47
Access-Register-Translation Process . . . 5-48
Selecting the Access-List-Entry Token . 5-51
Obtaining the Primary or Secondary Segment-Table Designation . . . . . 5-51
Checking the First Byte of the ALET . . 5-51
Obtaining the Effective Access-List Designation . . . . . 5-51
Access-List Lookup . . . . . 5-51
Locating the ASN-Second-Table Entry . 5-52
Authorizing the Use of the Access-List Entry . . . . . 5-52
Checking for Access-List-Controlled Protection . . . . . 5-53
Obtaining the Segment-Table Designation from the ASN-Second-Table Entry . . . . . 5-53
Recognition of Exceptions during Access-Register Translation . . . . 5-53
ART-Lookaside Buffer . . . . . 5-53
ALB Structure . . . . . 5-53
Formation of ALB Entries . . . . . 5-54
Use of ALB Entries . . . . . 5-54
Modification of ART Tables . . . . . 5-55
Subspace Groups . . . . . 5-55
Subspace-Group Tables . . . . . 5-56
Subspace-Group Dispatchable-Unit Control Table . . . . . 5-56
Subspace-Group ASN-Second-Table Entries . . . . . 5-57
Subspace-Replacement Operations . . . . 5-59
Linkage-Stack Introduction . . . . . 5-60
Summary . . . . . 5-60
Linkage-Stack Functions . . . . . 5-61
Transferring Program Control . . . . . 5-61
Branching Using the Linkage Stack . . 5-62
Adding and Retrieving Information . . . 5-63
Testing Authorization . . . . . 5-63
Program-Problem Analysis . . . . . 5-64
Extended Entry-Table Entries . . . . . 5-64
Linkage-Stack Operations . . . . . 5-66
Linkage-Stack-Operations Control . . . . . 5-68
Control Register 0 . . . . . 5-68



LzLabs GmbH. Ex. 1008-7

Control Register 15 . . . . . 5-68
Linkage Stack . . . . . 5-68
Entry Descriptors . . . . . 5-69
Header Entries . . . . . 5-70
Trailer Entries . . . . . 5-70
State Entries . . . . . 5-71
Stacking Process . . . . . 5-73
Locating Space for a New Entry . . . . 5-73
Forming the New Entry . . . . . 5-74
Updating the Current Entry . . . . . 5-75
Updating Control Register 15 . . . . . 5-75
Recognition of Exceptions during the Stacking Process . . . . . 5-75
Unstacking Process . . . . . 5-75
Locating the Current Entry and Processing a Header Entry . . . . . 5-76
Checking for a State Entry . . . . . 5-77
Restoring Information . . . . . 5-77
Updating the Preceding Entry . . . . . 5-77
Updating Control Register 15 . . . . . 5-77
Recognition of Exceptions during the Unstacking Process . . . . . 5-78
Sequence of Storage References . . . . . 5-78
Conceptual Sequence . . . . . 5-78
Overlapped Operation of Instruction Execution . . . . . 5-79
Divisible Instruction Execution . . . . . 5-79
Interlocks for Virtual-Storage References . 5-79
Interlocks between Instructions . . . . . 5-80
Interlocks within a Single Instruction . . 5-80
Instruction Fetching . . . . . 5-82
ART-Table and DAT-Table Fetches . . . . 5-83
Storage-Key Accesses . . . . . 5-84
Storage-Operand References . . . . . 5-85
Storage-Operand Fetch References . . 5-85
Storage-Operand Store References . . 5-85
Storage-Operand Update References . 5-86
Storage-Operand Consistency . . . . . 5-87
Single-Access References . . . . . 5-87
Multiple-Access References . . . . . 5-87
Block-Concurrent References . . . . . 5-88
Consistency Specification . . . . . 5-88
Relation between Operand Accesses . . . 5-90
Other Storage References . . . . . 5-90
Relation between Storage-Key Accesses . 5-90
Serialization . . . . . 5-91
CPU Serialization . . . . . 5-91
Channel-Program Serialization . . . . . 5-92

**Chapter 6. Interruptions** . . . . . 6-1
Interruption Action . . . . . 6-2
Interruption Code . . . . . 6-5
Enabling and Disabling . . . . . 6-6
Handling of Floating Interruption Conditions 6-6
Instruction-Length Code . . . . . 6-7
Zero ILC . . . . . 6-7
ILC on Instruction-Fetching Exceptions . 6-7
Exceptions Associated with the PSW . . . 6-9
Early Exception Recognition . . . . . 6-9
Late Exception Recognition . . . . . 6-9
External Interruption . . . . . 6-10
Clock Comparator . . . . . 6-11
CPU Timer . . . . . 6-11
Emergency Signal . . . . . 6-11
ETR . . . . . 6-12
External Call . . . . . 6-12
Interrupt Key . . . . . 6-12
Malfunction Alert . . . . . 6-12
Service Signal . . . . . 6-12
TOD-Clock Sync Check . . . . . 6-13
I/O Interruption . . . . . 6-13
Machine-Check Interruption . . . . . 6-14
Program Interruption . . . . . 6-14
Exception-Extension Code . . . . . 6-15
Data-Exception Code (DXC) . . . . . 6-15
Priority of Program Interruptions for Data Exceptions . . . . . 6-15
Program-Interruption Conditions . . . . . 6-16
Addressing Exception . . . . . 6-16
AFX-Translation Exception . . . . . 6-19
ALEN-Translation Exception . . . . . 6-19
ALE-Sequence Exception . . . . . 6-19
ALET-Specification Exception . . . . . 6-19
ASN-Translation-Specification Exception . . . . . 6-19
ASTE-Sequence Exception . . . . . 6-20
ASTE-Validity Exception . . . . . 6-20
ASX-Translation Exception . . . . . 6-21
Crypto-Operation Exception . . . . . 6-21
Data Exception . . . . . 6-21
Decimal-Divide Exception . . . . . 6-22
Decimal-Overflow Exception . . . . . 6-22
Execute Exception . . . . . 6-22
EX-Translation Exception . . . . . 6-22
Extended-Authority Exception . . . . . 6-22
Fixed-Point-Divide Exception . . . . . 6-23
Fixed-Point-Overflow Exception . . . . . 6-23
HFP-Divide Exception . . . . . 6-23
HFP-Exponent-Overflow Exception . . . 6-23
HFP-Exponent-Underflow Exception . . 6-24
HFP-Significance Exception . . . . . 6-24
HFP-Square-Root Exception . . . . . 6-24
LX-Translation Exception . . . . . 6-24
Monitor Event . . . . . 6-24
Operand Exception . . . . . 6-25
Operation Exception . . . . . 6-25
Page-Translation Exception . . . . . 6-26
PC-Translation-Specification Exception 6-27



LzLabs GmbH. Ex. 1008-8

PER Event . . . . . 6-27
Primary-Authority Exception . . . . . 6-27
Privileged-Operation Exception . . . . . 6-27
Protection Exception . . . . . 6-28
Secondary-Authority Exception . . . . . 6-29
Segment-Translation Exception . . . . . 6-29
Space-Switch Event . . . . . 6-29
Special-Operation Exception . . . . . 6-30
Specification Exception . . . . . 6-31
Stack-Empty Exception . . . . . 6-33
Stack-Full Exception . . . . . 6-33
Stack-Operation Exception . . . . . 6-33
Stack-Specification Exception . . . . . 6-33
Stack-Type Exception . . . . . 6-34
Trace-Table Exception . . . . . 6-34
Translation-Specification Exception . . . 6-34
Unnormalized-Operand Exception . . . 6-35
Vector-Operation Exception . . . . . 6-35
Collective Program-Interruption Names . . 6-35
Recognition of Access Exceptions . . . . . 6-35
Multiple Program-Interruption Conditions . 6-38
Access Exceptions . . . . . 6-40
ASN-Translation Exceptions . . . . . 6-45
Subspace-Replacement Exceptions . . 6-45
Trace Exceptions . . . . . 6-45
Restart Interruption . . . . . 6-46
Supervisor-Call Interruption . . . . . 6-46
Priority of Interruptions . . . . . 6-46

**Chapter 7. General Instructions** . . . . . 7-1
Data Format . . . . . 7-2
Binary-Integer Representation . . . . . 7-2
Binary Arithmetic . . . . . 7-3
Signed Binary Arithmetic . . . . . 7-3
Addition and Subtraction . . . . . 7-3
Fixed-Point Overflow . . . . . 7-4
Unsigned Binary Arithmetic . . . . . 7-4
Signed and Logical Comparison . . . . . 7-4
Instructions . . . . . 7-5
ADD . . . . . 7-12
ADD HALFWORD . . . . . 7-12
ADD HALFWORD IMMEDIATE . . . . . 7-12
ADD LOGICAL . . . . . 7-13
ADD LOGICAL WITH CARRY . . . . . 7-13
AND . . . . . 7-13
BRANCH AND LINK . . . . . 7-14
BRANCH AND SAVE . . . . . 7-15
BRANCH AND SAVE AND SET MODE . 7-16
BRANCH AND SET MODE . . . . . 7-16
BRANCH ON CONDITION . . . . . 7-17
BRANCH ON COUNT . . . . . 7-18
BRANCH ON INDEX HIGH . . . . . 7-18
BRANCH ON INDEX LOW OR EQUAL . . 7-18
BRANCH RELATIVE AND SAVE . . . . . 7-19
BRANCH RELATIVE AND SAVE LONG . 7-19
BRANCH RELATIVE ON CONDITION . . 7-20
BRANCH RELATIVE ON CONDITION LONG . . . . . 7-20
BRANCH RELATIVE ON COUNT . . . . . 7-21
BRANCH RELATIVE ON INDEX HIGH . . 7-21
BRANCH RELATIVE ON INDEX LOW OR EQUAL . . . . . 7-21
CHECKSUM . . . . . 7-22
CIPHER MESSAGE (KM) . . . . . 7-26
CIPHER MESSAGE WITH CHAINING (KMC) . . . . . 7-26
COMPARE . . . . . 7-35
COMPARE AND FORM CODEWORD . . 7-36
COMPARE AND SWAP . . . . . 7-40
COMPARE DOUBLE AND SWAP . . . . . 7-40
COMPARE HALFWORD . . . . . 7-42
COMPARE HALFWORD IMMEDIATE . . 7-42
COMPARE LOGICAL . . . . . 7-42
COMPARE LOGICAL CHARACTERS UNDER MASK . . . . . 7-43
COMPARE LOGICAL LONG . . . . . 7-43
COMPARE LOGICAL LONG EXTENDED 7-45
COMPARE LOGICAL LONG UNICODE . 7-47
COMPARE LOGICAL STRING . . . . . 7-50
COMPARE UNTIL SUBSTRING EQUAL . 7-51
COMPUTE INTERMEDIATE MESSAGE DIGEST (KIMD) . . . . . 7-55
COMPUTE LAST MESSAGE DIGEST (KLMD) . . . . . 7-55
COMPUTE MESSAGE AUTHENTICATION CODE (KMAC) . . 7-61
CONVERT TO BINARY . . . . . 7-66
CONVERT TO DECIMAL . . . . . 7-67
CONVERT UNICODE TO UTF-8 . . . . . 7-67
CONVERT UTF-8 TO UNICODE . . . . . 7-70
COPY ACCESS . . . . . 7-72
DIVIDE . . . . . 7-73
DIVIDE LOGICAL . . . . . 7-73
EXCLUSIVE OR . . . . . 7-74
EXECUTE . . . . . 7-74
EXTRACT ACCESS . . . . . 7-75
EXTRACT PSW . . . . . 7-76
INSERT CHARACTER . . . . . 7-76
INSERT CHARACTERS UNDER MASK . 7-76
INSERT PROGRAM MASK . . . . . 7-77
LOAD . . . . . 7-77
LOAD ACCESS MULTIPLE . . . . . 7-77
LOAD ADDRESS . . . . . 7-78
LOAD ADDRESS EXTENDED . . . . . 7-78
LOAD ADDRESS RELATIVE LONG . . . 7-79
LOAD AND TEST . . . . . 7-79
LOAD COMPLEMENT . . . . . 7-79
LOAD HALFWORD . . . . . 7-80



LzLabs GmbH. Ex. 1008-9

LOAD HALFWORD IMMEDIATE . . . . 7-80
LOAD MULTIPLE . . . . . 7-80
LOAD NEGATIVE . . . . . 7-80
LOAD POSITIVE . . . . . 7-81
LOAD REVERSED . . . . . 7-81
MONITOR CALL . . . . . 7-82
MOVE . . . . . 7-83
MOVE INVERSE . . . . . 7-83
MOVE LONG . . . . . 7-83
MOVE LONG EXTENDED . . . . . 7-87
MOVE LONG UNICODE . . . . . 7-90
MOVE NUMERICS . . . . . 7-93
MOVE PAGE (Facility 1) . . . . . 7-93
MOVE STRING . . . . . 7-95
MOVE WITH OFFSET . . . . . 7-97
MOVE ZONES . . . . . 7-97
MULTIPLY . . . . . 7-98
MULTIPLY HALFWORD . . . . . 7-98
MULTIPLY HALFWORD IMMEDIATE . . . 7-98
MULTIPLY LOGICAL . . . . . 7-99
MULTIPLY SINGLE . . . . . 7-99
OR . . . . . 7-100
PACK . . . . . 7-100
PACK ASCII . . . . . 7-101
PACK UNICODE . . . . . 7-102
PERFORM LOCKED OPERATION . . . 7-103
ROTATE LEFT SINGLE LOGICAL . . . 7-116
SEARCH STRING . . . . . 7-116
SET ACCESS . . . . . 7-117
SET ADDRESSING MODE . . . . . 7-117
SET PROGRAM MASK . . . . . 7-118
SHIFT LEFT DOUBLE . . . . . 7-118
SHIFT LEFT DOUBLE LOGICAL . . . . 7-119
SHIFT LEFT SINGLE . . . . . 7-119
SHIFT LEFT SINGLE LOGICAL . . . . . 7-120
SHIFT RIGHT DOUBLE . . . . . 7-120
SHIFT RIGHT DOUBLE LOGICAL . . . 7-121
SHIFT RIGHT SINGLE . . . . . 7-121
SHIFT RIGHT SINGLE LOGICAL . . . . 7-121
STORE . . . . . 7-122
STORE ACCESS MULTIPLE . . . . . 7-122
STORE CHARACTER . . . . . 7-122
STORE CHARACTERS UNDER MASK . 7-122
STORE CLOCK . . . . . 7-123
STORE CLOCK EXTENDED . . . . . 7-124
STORE HALFWORD . . . . . 7-126
STORE MULTIPLE . . . . . 7-126
STORE REVERSED . . . . . 7-126
SUBTRACT . . . . . 7-127
SUBTRACT HALFWORD . . . . . 7-127
SUBTRACT LOGICAL . . . . . 7-127
SUBTRACT LOGICAL WITH BORROW . 7-128
SUPERVISOR CALL . . . . . 7-129
TEST ADDRESSING MODE . . . . . 7-129
TEST AND SET . . . . . 7-129
TEST UNDER MASK . . . . . 7-130
TEST UNDER MASK HIGH . . . . . 7-130
TEST UNDER MASK LOW . . . . . 7-130
TRANSLATE . . . . . 7-131
TRANSLATE AND TEST . . . . . 7-132
TRANSLATE EXTENDED . . . . . 7-132
TRANSLATE ONE TO ONE . . . . . 7-134
TRANSLATE ONE TO TWO . . . . . 7-135
TRANSLATE TWO TO ONE . . . . . 7-135
TRANSLATE TWO TO TWO . . . . . 7-135
UNPACK . . . . . 7-139
UNPACK ASCII . . . . . 7-139
UNPACK UNICODE . . . . . 7-140
UPDATE TREE . . . . . 7-141

**Chapter 8. Decimal Instructions** . . . . . 8-1
Decimal-Number Formats . . . . . 8-1
Zoned Format . . . . . 8-1
Packed Format . . . . . 8-1
Decimal Codes . . . . . 8-2
Decimal Operations . . . . . 8-2
Decimal-Arithmetic Instructions . . . . . 8-2
Editing Instructions . . . . . 8-3
Execution of Decimal Instructions . . . . . 8-3
Other Instructions for Decimal Operands . 8-3
Decimal-Operand Data Exception . . . . . 8-4
Instructions . . . . . 8-4
ADD DECIMAL . . . . . 8-6
COMPARE DECIMAL . . . . . 8-6
DIVIDE DECIMAL . . . . . 8-7
EDIT . . . . . 8-7
EDIT AND MARK . . . . . 8-12
MULTIPLY DECIMAL . . . . . 8-12
SHIFT AND ROUND DECIMAL . . . . . 8-13
SUBTRACT DECIMAL . . . . . 8-14
TEST DECIMAL . . . . . 8-14
ZERO AND ADD . . . . . 8-14

**Chapter 9. Floating-Point Overview and Support Instructions** . . . . . 9-1
Registers And Controls . . . . . 9-2
Floating-Point Registers . . . . . 9-2
Additional Floating-Point (AFP) Registers . . . . . 9-2
Valid Floating-Point-Register Designations . . . . . 9-2
Floating-Point-Control (FPC) Register . . . 9-2
AFP-Register-Control Bit . . . . . 9-3
Explicit Rounding Methods . . . . . 9-3
Summary of Rounding Action . . . . . 9-3
Comparison of BFP and HFP Number Representations . . . . . 9-4
BFP and HFP Number Ranges . . . . . 9-4



LzLabs GmbH. Ex. 1008-10

Equivalent BFP and HFP Number Representations . . . 9-4
Instructions . . . 9-6
CONVERT BFP TO HFP . . . 9-8
CONVERT HFP TO BFP . . . 9-9
LOAD . . . 9-10
LOAD ZERO . . . 9-11
STORE . . . 9-11
Summary of All Floating-Point Instructions . . 9-11

**Chapter 10. Control Instructions** . . . 10-1
BRANCH AND SET AUTHORITY . . . 10-7
BRANCH AND STACK . . . 10-10
BRANCH IN SUBSPACE GROUP . . . 10-13
COMPARE AND SWAP AND PURGE . 10-17
DIAGNOSE . . . 10-19
EXTRACT PRIMARY ASN . . . 10-19
EXTRACT SECONDARY ASN . . . 10-20
EXTRACT STACKED REGISTERS . . . 10-20
EXTRACT STACKED STATE . . . 10-22
INSERT ADDRESS SPACE CONTROL .10-23
INSERT PSW KEY . . . 10-24
INSERT STORAGE KEY EXTENDED . 10-25
INSERT VIRTUAL STORAGE KEY . . . 10-25
INVALIDATE PAGE TABLE ENTRY . . . 10-26
LOAD ADDRESS SPACE PARAMETERS . . . 10-28
LOAD CONTROL . . . 10-37
LOAD PSW . . . 10-37
LOAD REAL ADDRESS . . . 10-38
LOAD USING REAL ADDRESS . . . 10-40
MODIFY STACKED STATE . . . 10-40
MOVE PAGE (Facility 2) . . . 10-42
MOVE TO PRIMARY . . . 10-45
MOVE TO SECONDARY . . . 10-45
MOVE WITH DESTINATION KEY . . . 10-47
MOVE WITH KEY . . . 10-47
MOVE WITH SOURCE KEY . . . 10-48
PAGE IN . . . 10-50
PAGE OUT . . . 10-51
PROGRAM CALL . . . 10-52
PROGRAM CALL FAST . . . 10-63
PROGRAM RETURN . . . 10-67
PROGRAM TRANSFER . . . 10-70
PURGE ALB . . . 10-76
PURGE TLB . . . 10-76
RESET REFERENCE BIT EXTENDED . 10-76
RESUME PROGRAM . . . 10-77
SET ADDRESS SPACE CONTROL . . . 10-79
SET ADDRESS SPACE CONTROL FAST . . . 10-79
SET CLOCK . . . 10-81
SET CLOCK COMPARATOR . . . 10-82
SET CLOCK PROGRAMMABLE FIELD .10-82
SET CPU TIMER . . . 10-82
SET PREFIX . . . 10-83
SET PSW KEY FROM ADDRESS . . . 10-83
SET SECONDARY ASN . . . 10-84
SET STORAGE KEY EXTENDED . . . 10-87
SET SYSTEM MASK . . . 10-87
SIGNAL PROCESSOR . . . 10-88
STORE CLOCK COMPARATOR . . . 10-89
STORE CONTROL . . . 10-89
STORE CPU ADDRESS . . . 10-90
STORE CPU ID . . . 10-90
STORE CPU TIMER . . . 10-91
STORE FACILITY LIST . . . 10-91
STORE PREFIX . . . 10-92
STORE SYSTEM INFORMATION . . . 10-92
STORE THEN AND SYSTEM MASK . 10-103
STORE THEN OR SYSTEM MASK . . 10-103
STORE USING REAL ADDRESS . . . 10-104
TEST ACCESS . . . 10-104
TEST BLOCK . . . 10-107
TEST PROTECTION . . . 10-109
TRACE . . . 10-111
TRAP . . . 10-112

**Chapter 11. Machine-Check Handling** . . . 11-1
Machine-Check Detection . . . 11-2
Correction of Machine Malfunctions . . . 11-2
Error Checking and Correction . . . 11-2
CPU Retry . . . 11-2
Effects of CPU Retry . . . 11-3
Checkpoint Synchronization . . . 11-3
Handling of Machine Checks during Checkpoint Synchronization . . . 11-3
Checkpoint-Synchronization Operations 11-3
Checkpoint-Synchronization Action . . . 11-4
Channel-Subsystem Recovery . . . 11-4
Unit Deletion . . . 11-4
Handling of Machine Checks . . . 11-5
Validation . . . 11-5
Invalid CBC in Storage . . . 11-6
Programmed Validation of Storage . . . 11-7
Invalid CBC in Storage Keys . . . 11-7
Invalid CBC in Registers . . . 11-10
Check-Stop State . . . 11-11
System Check Stop . . . 11-11
Machine-Check Interruption . . . 11-11
Exigent Conditions . . . 11-11
Repressible Conditions . . . 11-12
Interruption Action . . . 11-12
Point of Interruption . . . 11-14
Machine-Check-Interruption Code . . . 11-15
Subclass . . . 11-16
System Damage . . . 11-16
Instruction-Processing Damage . . . 11-17



LzLabs GmbH. Ex. 1008-11

System Recovery . . . . . 11-17
Timing-Facility Damage . . . . . 11-17
External Damage . . . . . 11-18
Vector-Facility Failure . . . . . 11-18
Degradation . . . . . 11-18
Warning . . . . . 11-18
Channel Report Pending . . . . . 11-18
Service-Processor Damage . . . . . 11-19
Channel-Subsystem Damage . . . . . 11-19
Subclass Modifiers . . . . . 11-19
Vector-Facility Source . . . . . 11-19
Backed Up . . . . . 11-19
Delayed Access Exception . . . . . 11-19
Ancillary Report . . . . . 11-19
Synchronous Machine-Check-Interruption Conditions 11-20
Processing Backup . . . . . 11-20
Processing Damage . . . . . 11-20
Storage Errors . . . . . 11-20
Storage Error Uncorrected . . . . . 11-21
Storage Error Corrected . . . . . 11-21
Storage-Key Error Uncorrected . . . . 11-21
Storage Degradation . . . . . 11-21
Indirect Storage Error . . . . . 11-21
Machine-Check Interruption-Code Validity Bits . . . . . 11-22
PSW-MWP Validity . . . . . 11-22
PSW Mask and Key Validity . . . . . 11-22
PSW Program-Mask and Condition-Code Validity . . . . . 11-22
PSW-Instruction-Address Validity . . . 11-22
Failing-Storage-Address Validity . . . 11-22
External-Damage-Code Validity . . . . 11-22
Floating-Point-Register Validity . . . . 11-23
General-Register Validity . . . . . 11-23
Control-Register Validity . . . . . 11-23
Storage Logical Validity . . . . . 11-23
Access-Register Validity . . . . . 11-23
Extended-Floating-Point-Register Validity . . . . . 11-23
CPU-Timer Validity . . . . . 11-23
Clock-Comparator Validity . . . . . 11-23
Machine-Check Extended Interruption Information . . . . . 11-24
Register Save Areas . . . . . 11-24
Machine-Check Extended Save Area . . 11-24
External-Damage Code . . . . . 11-24
Failing-Storage Address . . . . . 11-25
Handling of Machine-Check Conditions . . 11-25
Floating Interruption Conditions . . . . . 11-25
Floating Machine-Check-Interruption Conditions . . . . . 11-26
Floating I/O Interruptions . . . . . 11-26
Machine-Check Masking . . . . . 11-26
Channel-Report-Pending Subclass Mask . . . . . 11-26
Recovery Subclass Mask . . . . . 11-26
Degradation Subclass Mask . . . . . 11-26
External-Damage Subclass Mask . . . 11-26
Warning Subclass Mask . . . . . 11-26
Machine-Check Logout . . . . . 11-27
Summary of Machine-Check Masking . . . 11-27

**Chapter 12. Operator Facilities** . . . . . 12-1
Manual Operation . . . . . 12-1
Basic Operator Facilities . . . . . 12-1
Address-Compare Controls . . . . . 12-1
Alter-and-Display Controls . . . . . 12-2
Architectural-Mode Indicator . . . . . 12-2
Architectural-Mode-Selection Controls . . . 12-2
Check-Stop Indicator . . . . . 12-2
IML Controls . . . . . 12-3
Interrupt Key . . . . . 12-3
Load Indicator . . . . . 12-3
Load-Clear Key . . . . . 12-3
Load-Normal Key . . . . . 12-3
Load-Unit-Address Controls . . . . . 12-3
Manual Indicator . . . . . 12-3
Power Controls . . . . . 12-3
Rate Control . . . . . 12-4
Restart Key . . . . . 12-4
Start Key . . . . . 12-4
Stop Key . . . . . 12-4
Store-Status Key . . . . . 12-4
System-Reset-Clear Key . . . . . 12-5
System-Reset-Normal Key . . . . . 12-5
Test Indicator . . . . . 12-5
TOD-Clock Control . . . . . 12-5
Wait Indicator . . . . . 12-5
Multiprocessing Configurations . . . . . 12-6

**Chapter 13. I/O Overview** . . . . . 13-1
Input/Output (I/O) . . . . . 13-1
The Channel Subsystem . . . . . 13-1
Subchannels . . . . . 13-2
Attachment of Input/Output Devices . . . . . 13-2
Channel Paths . . . . . 13-2
Control Units . . . . . 13-4
I/O Devices . . . . . 13-4
I/O Addressing . . . . . 13-5
Channel-Path Identifier . . . . . 13-5
Subchannel Number . . . . . 13-5
Device Number . . . . . 13-5
Device Identifier . . . . . 13-5
Execution of I/O Operations . . . . . 13-6
Start-Function Initiation . . . . . 13-6
Path Management . . . . . 13-6
Channel-Program Execution . . . . . 13-7



LzLabs GmbH. Ex. 1008-12

Conclusion of I/O Operations . . . . . 13-8
I/O Interruptions . . . . . 13-9

**Chapter 14. I/O Instructions** . . . . . 14-1
I/O-Instruction Formats . . . . . 14-1
I/O-Instruction Execution . . . . . 14-1
Serialization . . . . . 14-1
Operand Access . . . . . 14-1
Condition Code . . . . . 14-2
Program Exceptions . . . . . 14-2
Instructions . . . . . 14-2
CANCEL SUBCHANNEL . . . . . 14-4
CLEAR SUBCHANNEL . . . . . 14-5
HALT SUBCHANNEL . . . . . 14-6
MODIFY SUBCHANNEL . . . . . 14-7
RESET CHANNEL PATH . . . . . 14-9
RESUME SUBCHANNEL . . . . . 14-10
SET ADDRESS LIMIT . . . . . 14-11
SET CHANNEL MONITOR . . . . . 14-12
START SUBCHANNEL . . . . . 14-14
STORE CHANNEL PATH STATUS . . . 14-16
STORE CHANNEL REPORT WORD . . 14-17
STORE SUBCHANNEL . . . . . 14-17
TEST PENDING INTERRUPTION . . . . 14-18
TEST SUBCHANNEL . . . . . 14-20

**Chapter 15. Basic I/O Functions** . . . . . 15-1
Control of Basic I/O Functions . . . . . 15-1
Subchannel-Information Block . . . . . 15-1
Path-Management-Control Word . . . . 15-2
Subchannel-Status Word . . . . . 15-8
Model-Dependent Area/Measurement Block Address . . . . . 15-8
Summary of Modifiable Fields . . . . . 15-9
Channel-Path Allegiance . . . . . 15-11
Working Allegiance . . . . . 15-12
Active Allegiance . . . . . 15-12
Dedicated Allegiance . . . . . 15-12
Channel-Path Availability . . . . . 15-13
Control-Unit Type . . . . . 15-13
Clear Function . . . . . 15-14
Clear-Function Path Management . . . . 15-14
Clear-Function Subchannel Modification . 15-14
Clear-Function Signaling and Completion . . . . . 15-15
Halt Function . . . . . 15-15
Halt-Function Path Management . . . . . 15-16
Halt-Function Signaling and Completion . 15-16
Start Function and Resume Function . . . . 15-18
Start-Function and Resume-Function Path Management . . . . . 15-19
Execution of I/O Operations . . . . . 15-21
Blocking of Data . . . . . 15-22
Operation-Request Block . . . . . 15-22
Channel-Command Word . . . . . 15-27
Command Code . . . . . 15-29
Designation of Storage Area . . . . . 15-30
Chaining . . . . . 15-31
Data Chaining . . . . . 15-33
Command Chaining . . . . . 15-34
Skipping . . . . . 15-35
Program-Controlled Interruption . . . . . 15-35
CCW Indirect Data Addressing . . . . . 15-36
Suspension of Channel-Program Execution . . . . . 15-38
Commands and Flags . . . . . 15-40
Branching in Channel Programs . . . . . 15-41
Transfer in Channel . . . . . 15-41
Command Retry . . . . . 15-42
Concluding I/O Operations before Initiation . 15-42
Concluding I/O Operations during Initiation . 15-42
Immediate Conclusion of I/O Operations . . 15-43
Concluding I/O Operations during Data Transfer . . . . . 15-43
Channel-Path-Reset Function . . . . . 15-45
Channel-Path-Reset-Function Signaling . 15-45
Channel-Path-Reset-Function-Completion Signaling . . . . . 15-45

**Chapter 16. I/O Interruptions** . . . . . 16-1
Interruption Conditions . . . . . 16-2
Intermediate Interruption Condition . . . . 16-4
Primary Interruption Condition . . . . . 16-4
Secondary Interruption Condition . . . . . 16-4
Alert Interruption Condition . . . . . 16-4
Priority of Interruptions . . . . . 16-4
Interruption Action . . . . . 16-5
Interruption-Response Block . . . . . 16-6
Subchannel-Status Word . . . . . 16-6
Subchannel Key . . . . . 16-8
Suspend Control (S) . . . . . 16-8
Extended-Status-Word Format (L) . . . 16-8
Deferred Condition Code (CC) . . . . . 16-8
Format (F) . . . . . 16-10
Prefetch (P) . . . . . 16-10
Initial-Status-Interruption Control (I) . . 16-11
Address-Limit-Checking Control (A) . 16-11
Suppress-Suspended Interruption (U) . 16-11
Subchannel-Control Field . . . . . 16-11
Zero Condition Code (Z) . . . . . 16-11
Extended Control (E) . . . . . 16-11
Path Not Operational (N) . . . . . 16-12
Function Control (FC) . . . . . 16-12
Activity Control (AC) . . . . . 16-13
Status Control (SC) . . . . . 16-16
CCW-Address Field . . . . . 16-18
Device-Status Field . . . . . 16-23
Subchannel-Status Field . . . . . 16-23



LzLabs GmbH. Ex. 1008-13

Program-Controlled Interruption . . . . 16-23
Incorrect Length . . . . . 16-23
Program Check . . . . . 16-24
Protection Check . . . . . 16-26
Channel-Data Check . . . . . 16-26
Channel-Control Check . . . . . 16-27
Interface-Control Check . . . . . 16-28
Chaining Check . . . . . 16-29
Count Field . . . . . 16-29
Extended-Status Word . . . . . 16-32
Extended-Status Format 0 . . . . . 16-32
Subchannel Logout . . . . . 16-32
Extended-Report Word . . . . . 16-36
Failing-Storage Address . . . . . 16-37
Secondary-CCW Address . . . . . 16-38
Extended-Status Format 1 . . . . . 16-38
Extended-Status Format 2 . . . . . 16-38
Extended-Status Format 3 . . . . . 16-39
Extended-Control Word . . . . . 16-40
Extended-Measurement Word . . . . . 16-40

**Chapter 17. I/O Support Functions** . . . . 17-1
Channel-Subsystem Monitoring . . . . . 17-1
Channel-Subsystem Timing . . . . . 17-2
Channel-Subsystem Timer . . . . . 17-2
Measurement-Block Update . . . . . 17-3
Measurement Block . . . . . 17-3
Measurement-Block Format . . . . . 17-7
Measurement-Block Origin . . . . . 17-7
Measurement-Block Address . . . . . 17-8
Measurement-Block Key . . . . . 17-8
Measurement-Block Index . . . . . 17-8
Measurement-Block-Update Mode . . . 17-8
Measurement-Block-Format Control . . 17-9
Measurement-Block-Update Enable . . 17-9
Control-Unit-Queuing Measurement . . 17-9
Control-Unit-Defer Time . . . . . 17-9
Device-Active-Only Measurement . . . . 17-9
Initial-Command-Response Measurement . . . . . 17-10
Time-Interval-Measurement Accuracy . 17-10
Device-Connect-Time Measurement . . . 17-10
Device-Connect-Time-Measurement Mode . . . . . 17-10
Device-Connect-Time-Measurement Enable . . . . . 17-11
Extended Measurement Word . . . . . 17-11
Extended-Measurement-Word Enable 17-11
Signals and Resets . . . . . 17-12
Signals . . . . . 17-12
Halt Signal . . . . . 17-12
Clear Signal . . . . . 17-12
Reset Signal . . . . . 17-13
Resets . . . . . 17-13
Channel-Path Reset . . . . . 17-13
I/O-System Reset . . . . . 17-13
Externally Initiated Functions . . . . . 17-17
Initial Program Loading . . . . . 17-17
Reconfiguration of the I/O System . . . . 17-20
Status Verification . . . . . 17-20
Address-Limit Checking . . . . . 17-20
Configuration Alert . . . . . 17-21
Incorrect-Length-Indication Suppression . . 17-21
Concurrent Sense . . . . . 17-21
Channel-Subsystem Recovery . . . . . 17-21
Channel Report . . . . . 17-22
Channel-Report Word . . . . . 17-23
Channel-Subsystem-I/O-Priority Facility . . 17-25
Number of Channel-Subsystem-Priority Levels 17-26

**Chapter 18. Hexadecimal-Floating-Point Instructions** . . . . . 18-1
HFP Arithmetic . . . . . 18-1
HFP Number Representation . . . . . 18-1
Normalization . . . . . 18-3
HFP Data Format . . . . . 18-3
Instructions . . . . . 18-4
ADD NORMALIZED . . . . . 18-8
ADD UNNORMALIZED . . . . . 18-10
COMPARE . . . . . 18-10
CONVERT FROM FIXED . . . . . 18-11
CONVERT TO FIXED . . . . . 18-11
DIVIDE . . . . . 18-12
HALVE . . . . . 18-13
LOAD AND TEST . . . . . 18-14
LOAD COMPLEMENT . . . . . 18-15
LOAD FP INTEGER . . . . . 18-15
LOAD LENGTHENED . . . . . 18-16
LOAD NEGATIVE . . . . . 18-16
LOAD POSITIVE . . . . . 18-17
LOAD ROUNDED . . . . . 18-18
MULTIPLY . . . . . 18-18
MULTIPLY AND ADD . . . . . 18-20
MULTIPLY AND SUBTRACT . . . . . 18-20
SQUARE ROOT . . . . . 18-21
SUBTRACT NORMALIZED . . . . . 18-23
SUBTRACT UNNORMALIZED . . . . . 18-23

**Chapter 19. Binary-Floating-Point Instructions** . . . . . 19-1
Binary-Floating-Point Facility . . . . . 19-1
Floating-Point-Control (FPC) Register . . . 19-2
IEEE Masks and Flags . . . . . 19-3
FPC DXC Byte . . . . . 19-3
Operations on the FPC Register . . . . 19-3
BFP Arithmetic . . . . . 19-4
BFP Data Formats . . . . . 19-4



LzLabs GmbH. Ex. 1008-14

BFP Short Format . . . . . 19-4
BFP Long Format . . . . . 19-4
BFP Extended Format . . . . . 19-4
Biased Exponent . . . . . 19-4
Significand . . . . . 19-4
Values of Nonzero Numbers . . . . . 19-4
Classes of BFP Data . . . . . 19-5
Zeros . . . . . 19-6
Denormalized Numbers . . . . . 19-6
Normalized Numbers . . . . . 19-6
Infinities . . . . . 19-6
Signaling and Quiet NaNs . . . . . 19-6
BFP-Format Conversion . . . . . 19-7
BFP Rounding . . . . . 19-7
Rounding Mode . . . . . 19-7
Normalization and Denormalization . . . . 19-8
BFP Comparison . . . . . 19-8
Condition Codes for BFP Instructions . . . 19-9
Remainder . . . . . 19-9
IEEE Exception Conditions . . . . . 19-10
IEEE Invalid Operation . . . . . 19-10
IEEE Division-By-Zero . . . . . 19-11
IEEE Overflow . . . . . 19-11
IEEE Underflow . . . . . 19-12
IEEE Inexact . . . . . 19-12
Result Figures . . . . . 19-13
Data-Exception Codes (DXC) and Abbreviations . . . . . 19-14
Instructions . . . . . 19-15
ADD . . . . . 19-18
COMPARE . . . . . 19-23
COMPARE AND SIGNAL . . . . . 19-24
CONVERT FROM FIXED . . . . . 19-26
CONVERT TO FIXED . . . . . 19-27
DIVIDE . . . . . 19-29
DIVIDE TO INTEGER . . . . . 19-30
EXTRACT FPC . . . . . 19-34
LOAD AND TEST . . . . . 19-35
LOAD COMPLEMENT . . . . . 19-35
LOAD FP INTEGER . . . . . 19-36
LOAD FPC . . . . . 19-37
LOAD LENGTHENED . . . . . 19-38
LOAD NEGATIVE . . . . . 19-38
LOAD POSITIVE . . . . . 19-39
LOAD ROUNDED . . . . . 19-39
MULTIPLY . . . . . 19-40
MULTIPLY AND ADD . . . . . 19-42
MULTIPLY AND SUBTRACT . . . . . 19-42
SET FPC . . . . . 19-44
SET ROUNDING MODE . . . . . 19-44
SQUARE ROOT . . . . . 19-45
STORE FPC . . . . . 19-45
SUBTRACT . . . . . 19-45
TEST DATA CLASS . . . . . 19-46

**Appendix A. Number Representation and Instruction-Use Examples** . . . . . A-1
Number Representation . . . . . A-2
Binary Integers . . . . . A-2
Signed Binary Integers . . . . . A-2
Unsigned Binary Integers . . . . . A-3
Decimal Integers . . . . . A-4
Hexadecimal-Floating-Point Numbers . . . A-5
Conversion Example . . . . . A-6
Instruction-Use Examples . . . . . A-6
Machine Format . . . . . A-6
Assembler-Language Format . . . . . A-7
Addressing Mode in Examples . . . . A-7
General Instructions . . . . . A-7
ADD HALFWORD (AH) . . . . . A-7
AND (N, NC, NI, NR) . . . . . A-7
NI Example . . . . . A-8
Linkage Instructions (BAL, BALR, BAS, BASR, BASSM, BSM) . . . . . A-8
Other BALR and BASR Examples . . . A-9
BRANCH AND STACK (BAKR) . . . . . A-9
BAKR Example 1 . . . . . A-10
BAKR Example 2 . . . . . A-10
BAKR Example 3 . . . . . A-11
BRANCH ON CONDITION (BC, BCR) . A-11
BRANCH ON COUNT (BCT, BCTR) . . A-12
BRANCH ON INDEX HIGH (BXH) . . . . A-12
BXH Example 1 . . . . . A-12
BXH Example 2 . . . . . A-12
BRANCH ON INDEX LOW OR EQUAL (BXLE) . . . . . A-13
BXLE Example 1 . . . . . A-13
BXLE Example 2 . . . . . A-14
COMPARE AND FORM CODEWORD (CFC) . . . . . A-14
COMPARE HALFWORD (CH) . . . . . A-14
COMPARE LOGICAL (CL, CLC, CLI, CLR) . . . . . A-14
CLC Example . . . . . A-14
CLI Example . . . . . A-15
CLR Example . . . . . A-15
COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) . . . . . A-15
COMPARE LOGICAL LONG (CLCL) . . A-16
COMPARE LOGICAL STRING (CLST) . A-17
CONVERT TO BINARY (CVB) . . . . . A-18
CONVERT TO DECIMAL (CVD) . . . . . A-18
DIVIDE (D, DR) . . . . . A-19
EXCLUSIVE OR (X, XC, XI, XR) . . . . A-19
XC Example . . . . . A-19
XI Example . . . . . A-20
EXECUTE (EX) . . . . . A-21



LzLabs GmbH. Ex. 1008-15

INSERT CHARACTERS UNDER MASK (ICM) . . . A-21
LOAD (L, LR) . . . A-22
LOAD ADDRESS (LA) . . . A-22
LOAD HALFWORD (LH) . . . A-23
MOVE (MVC, MVI) . . . A-23
MVC Example . . . A-23
MVI Example . . . A-24
MOVE INVERSE (MVCIN) . . . A-24
MOVE LONG (MVCL) . . . A-25
MOVE NUMERICS (MVN) . . . A-25
MOVE STRING (MVST) . . . A-26
MOVE WITH OFFSET (MVO) . . . A-26
MOVE ZONES (MVZ) . . . A-27
MULTIPLY (M, MR) . . . A-27
MULTIPLY HALFWORD (MH) . . . A-27
OR (O, OC, OI, OR) . . . A-28
OI Example . . . A-28
PACK (PACK) . . . A-28
SEARCH STRING (SRST) . . . A-29
SRST Example 1 . . . A-29
SRST Example 2 . . . A-29
SHIFT LEFT DOUBLE (SLDA) . . . A-29
SHIFT LEFT SINGLE (SLA) . . . A-30
STORE CHARACTERS UNDER MASK (STCM) . . . A-30
STORE MULTIPLE (STM) . . . A-30
TEST UNDER MASK (TM) . . . A-31
TRANSLATE (TR) . . . A-31
TRANSLATE AND TEST (TRT) . . . A-32
UNPACK (UNPK) . . . A-33
UPDATE TREE (UPT) . . . A-34
Decimal Instructions . . . A-34
ADD DECIMAL (AP) . . . A-34
COMPARE DECIMAL (CP) . . . A-34
DIVIDE DECIMAL (DP) . . . A-34
EDIT (ED) . . . A-35
EDIT AND MARK (EDMK) . . . A-36
MULTIPLY DECIMAL (MP) . . . A-36
SHIFT AND ROUND DECIMAL (SRP) . A-37
Decimal Left Shift . . . A-37
Decimal Right Shift . . . A-37
Decimal Right Shift and Round . . . A-38
Multiplying by a Variable Power of 10 . A-38
ZERO AND ADD (ZAP) . . . A-38
Hexadecimal-Floating-Point Instructions . . A-39
ADD NORMALIZED (AD, ADR, AE, AER, AXR) . . . A-39
ADD UNNORMALIZED (AU, AUR, AW, AWR) . . . A-39
COMPARE (CD, CDR, CE, CER) . . . A-40
DIVIDE (DD, DDR, DE, DER) . . . A-40
HALVE (HDR, HER) . . . A-41
MULTIPLY (MD, MDR, MDE, MDER, MXD, MXDR, MXR) . . . A-41
Hexadecimal-Floating-Point-Number Conversion . . . A-42
Fixed Point to Hexadecimal Floating Point . . . A-42
Hexadecimal Floating Point to Fixed Point . . . A-42
Multiprogramming and Multiprocessing Examples . . . A-43
Example of a Program Failure Using OR Immediate . . . A-43
Conditional Swapping Instructions (CS, CDS) . . . A-44
Setting a Single Bit . . . A-44
Updating Counters . . . A-45
Bypassing Post and Wait . . . A-45
Bypass Post Routine . . . A-45
Bypass Wait Routine . . . A-46
Lock/Unlock . . . A-46
Lock/Unlock with LIFO Queuing for Contentions . . . A-46
Lock/Unlock with FIFO Queuing for Contentions . . . A-47
Free-Pool Manipulation . . . A-48
PERFORM LOCKED OPERATION (PLO) A-50
Sorting Instructions . . . A-51
Tree Format . . . A-51
Example of Use of Sort Instructions . . . A-53

**Appendix B. Lists of Instructions** . . . B-1

**Appendix C. Condition-Code Settings** . . C-1

**Appendix D. Comparison between ESA/370 and ESA/390** . . . D-1
New Facilities in ESA/390 . . . D-1
Access-List-Controlled Protection . . . D-1
Additional Floating-Point . . . D-1
Additional Input/Output . . . D-2
Branch and Set Authority . . . D-2
Called-Space Identification . . . D-2
Checksum . . . D-2
Compare and Move Extended . . . D-2
Concurrent Sense . . . D-2
Extended TOD Clock . . . D-2
Extended Translation 1 . . . D-3
Extended Translation 2 . . . D-3
Immediate and Relative Instruction . . . D-3
Move-Page Facility 2 . . . D-3
PER 2 . . . D-4
Perform Locked Operation . . . D-4
Program Call Fast . . . D-4
Resume Program . . . D-4



LzLabs GmbH. Ex. 1008-16

Set Address Space Control Fast . . . . . D-5
Square Root . . . . . D-5
Storage-Protection Override . . . . . D-5
Store System Information . . . . . D-5
String Instruction . . . . . D-5
Subspace Group . . . . . D-5
Suppression on Protection . . . . . D-6
TOD-Clock-Control Override . . . . . D-6
Trap . . . . . D-6
z/Architecture . . . . . D-6
Comparison of Facilities . . . . . D-7

**Appendix E. Comparison between 370-XA and ESA/370** . . . . . E-1
New Facilities in ESA/370 . . . . . E-1
Access Registers . . . . . E-1
Compare until Substring Equal . . . . . E-1
Home Address Space . . . . . E-1
Linkage Stack . . . . . E-2
Load and Store Using Real Address . . . E-2
Move Page Facility 1 . . . . . E-2
Move with Source or Destination Key . . . E-2
Private Space . . . . . E-2
Comparison of Facilities . . . . . E-2
Summary of Changes . . . . . E-2
New Instructions Provided . . . . . E-2
Comparison of PSW Formats . . . . . E-3
New Control-Register Assignments . . . . E-3
New Assigned Storage Locations . . . . . E-3
New Exceptions . . . . . E-4
Change to Secondary-Space Mode . . . . E-4
Changes to ASN-Second-Table Entry and ASN Translation . . . . . E-4
Changes to Entry-Table Entry and PC-Number Translation . . . . . E-5
Changes to PROGRAM CALL . . . . . E-5
Changes to SET ADDRESS SPACE CONTROL . . . . . E-5
Effects in New Translation Modes . . . . . E-5
Effects on Interlocks for Virtual-Storage References . . . . . E-5
Effect on INSERT ADDRESS SPACE CONTROL . . . . . E-6
Effect on LOAD REAL ADDRESS . . . . . E-6
Effect on TEST PENDING INTERRUPTION . . . . . E-6
Effect on TEST PROTECTION . . . . . E-6

**Appendix F. Comparison between System/370 and 370-XA** . . . . . F-1
New Facilities in 370-XA . . . . . F-1
Bimodal Addressing . . . . . F-1
31-Bit Logical Addressing . . . . . F-1
31-Bit Real and Absolute Addressing . . . F-1
Page Protection . . . . . F-2
Tracing . . . . . F-2
Incorrect-Length-Indication Suppression . F-2
Status Verification . . . . . F-2
Comparison of Facilities . . . . . F-2
Summary of Changes . . . . . F-3
Changes in Instructions Provided . . . . . F-3
Input/Output Comparison . . . . . F-5
Comparison of PSW Formats . . . . . F-5
Changes in Control-Register Assignments . F-6
Changes in Assigned Storage Locations . F-6
Changes to SIGNAL PROCESSOR . . . . F-6
Machine-Check Changes . . . . . F-7
Changes to Addressing Wraparound . . . F-7
Changes to LOAD REAL ADDRESS . . . F-7
Changes to 31-Bit Real Operand Addresses . . . . . F-8

**Appendix G. Table of Powers of 2** . . . . . G-1

**Appendix H. Hexadecimal Tables** . . . . . H-1

**Appendix I. EBCDIC and Other Codes** . . . I-1

**Index** . . . . . X-1



LzLabs GmbH. Ex. 1008-17

LzLabs GmbH. Ex. 1008-18

# Notices

References in this publication to IBM* products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY, 10594 USA.

## Trademarks

The following terms, denoted by an asterisk (*) at the first or most prominent occurrence in this publication, are trademarks of the International Business Machines Corporation in the United States or other countries:

AIX/ESA
BookMaster
CICS
DB2
Enterprise Systems Architecture/370
Enterprise Systems Architecture/390
Enterprise Systems Connection Architecture
ESA/370
ESA/390
ESCON
FICON
IBM
IBMLink
MVS/ESA
OS/390
Processor Resource/Systems Manager
PR/SM
Sysplex Timer
System/370
VM/ESA
z/Architecture
z/OS

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-19

LzLabs GmbH. Ex. 1008-20

# Preface

This publication provides, for reference purposes, a detailed Enterprise Systems Architecture/390* (ESA/390*) description.

The publication applies only to systems operating as defined by ESA/390. For systems operating in accordance with the System/370* or System/370 extended-architecture (370-XA) definitions, the *IBM System/370 Principles of Operation*, GA22-7000, or the *IBM 370-XA Principles of Operation*, SA22-7085, should be consulted. For systems operating in accordance with the Enterprise Systems Architecture/370* (ESA/370*) definition, the *IBM ESA/370 Principles of Operation*, SA22-7200, should be consulted. For systems operating in accordance with the z/Architecture* definition, the *z/Architecture Principles of Operation*, SA22-7832, should be consulted.

The publication describes each function at the level of detail needed to prepare an assembler-language program that relies on that function. It does not, however, describe the notation and conventions that must be employed in preparing such a program, for which the user must instead refer to the appropriate assembler-language publication.

The information in this publication is provided principally for use by assembler-language programmers, although anyone concerned with the functional details of ESA/390 will find it useful.

This publication is written as a reference and should not be considered an introduction or a textbook. It assumes the user has a basic knowledge of data-processing systems.

All facilities discussed in this publication are not necessarily available on every model. Furthermore, in some instances the definitions have been structured to allow for some degree of extendibility, and therefore certain capabilities may be described or implied that are not offered on any model. Examples of such capabilities are the use of a 16-bit field in the subsystem-identification word to identify the subchannel number, the size of the CPU address, and the number of CPUs sharing main storage. The allowance for this type of extendibility should not be construed as implying any intention by IBM to provide such capabilities. For information about the characteristics and availability of facilities on a specific model, see the functional characteristics publication for that model.

Largely because this publication is arranged for reference, certain words and phrases appear, of necessity, earlier in the publication than the principal discussions explaining them. The reader who encounters a problem because of this arrangement should refer to the index, which indicates the location of the key description.

The information presented in this publication is grouped in 19 chapters and several appendixes:

*Chapter 1, Introduction,* highlights the major facilities of the ESA/390 architecture.

*Chapter 2, Organization,* describes the major groupings within the system — main storage, expanded storage, the central processing unit (CPU), the external time reference (ETR), and input/output — with some attention given to the composition and characteristics of those groupings.

*Chapter 3, Storage,* explains the information formats, the addressing of storage, and the facilities for storage protection. It also deals with dynamic address translation (DAT), which, coupled with special programming support, makes the use of a virtual storage possible.

*Chapter 4, Control,* describes the facilities for the switching of system status, for special externally initiated operations, for debugging, and for timing. It deals specifically with CPU states, control modes, the program-status word (PSW), control registers, tracing, program-event recording, timing facilities, resets, store status, and initial program loading.

---

Enterprise Systems Architecture/390, ESA/390, System/370, Enterprise Systems Architecture/370, ESA/370, and z/Architecture are trademarks of the International Business Machines Corporation.

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-21

*Chapter 5, Program Execution,* explains the role of instructions in program execution, looks in detail at instruction formats, and describes briefly the use of the program-status word (PSW), of branching, and of interruptions. It contains the principal description of the advanced address-space facilities that were introduced in ESA/370. It also details the aspects of program execution on one CPU as observed by other CPUs and by channel programs.

*Chapter 6, Interruptions,* details the mechanism that permits the CPU to change its state as a result of conditions external to the system, within the system, or within the CPU itself. Six classes of interruptions are identified and described: machine-check interruptions, program interruptions, supervisor-call interruptions, external interruptions, input/output interruptions, and restart interruptions.

*Chapter 7, General Instructions,* contains detailed descriptions of logical and binary-integer data formats and of all unprivileged instructions except the decimal and floating-point instructions.

*Chapter 8, Decimal Instructions,* describes in detail decimal data formats and the decimal instructions.

*Chapter 9, Floating-Point Overview and Support Instructions,* includes an introduction to the floating-point operations, detailed descriptions of those instructions common to both hexadecimal-floating-point and binary-floating-point operations, and summaries of all floating-point instructions.

*Chapter 10, Control Instructions,* contains detailed descriptions of all of the semiprivileged and privileged instructions except for the I/O instructions.

*Chapter 11, Machine-Check Handling,* describes the mechanism for detecting, correcting, and reporting machine malfunctions.

*Chapter 12, Operator Facilities,* describes the basic manual functions and controls available for operating and controlling the system.

Chapters 13-17 of this publication provide a detailed definition of the functions performed by the channel subsystem and the logical interface between the CPU and the channel subsystem.

*Chapter 13, I/O Overview,* provides a brief description of the basic components and operation of the channel subsystem.

*Chapter 14, I/O Instructions,* contains the description of the I/O instructions.

*Chapter 15, Basic I/O Functions,* describes the basic I/O functions performed by the channel subsystem, including the initiation, control, and conclusion of I/O operations.

*Chapter 16, I/O Interruptions,* covers I/O interruptions and interruption conditions.

*Chapter 17, I/O Support Functions,* describes such functions as channel-subsystem usage monitoring, resets, initial-program loading, reconfiguration, and channel-subsystem recovery.

*Chapter 18, Hexadecimal-Floating-Point Instructions,* contains detailed descriptions of the hexadecimal-floating-point (HFP) data formats and the HFP instructions.

*Chapter 19, Binary-Floating-Point Instructions,* contains detailed descriptions of the binary-floating-point (BFP) data formats and the BFP instructions.

The *Appendixes* include:

- Information about number representation
- Instruction-use examples
- Lists of the instructions arranged in several sequences
- A summary of the condition-code settings
- A summary of the differences between ESA/370 and ESA/390
- A summary of the differences between 370-XA and ESA/370
- A summary of the differences between System/370 and 370-XA
- A table of the powers of 2
- Tabular information helpful in dealing with hexadecimal numbers
- A table of EBCDIC and other codes.

Certain information about commands that is in Chapters 15 and 16 of the ESA/370 Principles of Operation is not in this publication; instead it is in the publication *IBM Enterprise Systems Architecture/390 Common I/O-Device Commands and Self Description*, SA22-7204.



LzLabs GmbH. Ex. 1008-22

## Size and Number Notation

In this publication, the letters K, M, G, and T denote the multipliers $2^{10}$, $2^{20}$, $2^{30}$, and $2^{40}$, respectively. Although the letters are borrowed from the decimal system and stand for kilo ($10^3$), mega ($10^6$), giga ($10^9$), and tera ($10^{12}$), they do not have the decimal meaning but instead represent the power of 2 closest to the corresponding power of 10. Their meaning in this publication is as follows:

| Symbol | Value |
|---|---|
| K (kilo) | 1,024 = $2^{10}$ |
| M (mega) | 1,048,576 = $2^{20}$ |
| G (giga) | 1,073,741,824 = $2^{30}$ |
| T (tera) | 1,099,511,627,776 = $2^{40}$ |

The following are some examples of the use of K, M, G, and T:

2,048 is expressed as 2K.
4,096 is expressed as 4K.
65,536 is expressed as 64K (not 65K).
$2^{24}$ is expressed as 16M.
$2^{31}$ is expressed as 2G.
$2^{42}$ is expressed as 4T.

When the words “thousand” and “million” are used, no special power-of-2 meaning is assigned to them.

All numbers in this publication are in decimal unless they are explicitly noted as being in binary or hexadecimal (hex).

## Bytes, Characters, and Codes

Although the System/360 architecture was originally designed to support the Extended Binary-Coded-Decimal Interchange Code (EBCDIC), the instructions and data formats of the architecture are for the most part independent of the external code which is to be processed by the machine. For most instructions, all 256 possible combinations of bit patterns for a particular byte can be processed, independent of the character which the bit pattern is intended to represent. For instructions which use the zoned format, and for those few instructions which are dependent on a particular external code, the instruction TRANSLATE may be used to convert data from one code to another code. Thus, a machine operating in accordance with ESA/390 can process EBCDIC, ASCII, or any other code which can be represented in eight or fewer bits per character.

In this publication, unless otherwise specified, the value given for a byte is the value obtained by considering the bits of the byte to represent a binary code. Thus, when a byte is said to contain a zero, the value 00000000 binary, or 00 hex, is meant, and not the value for an EBCDIC character “0,” which would be F0 hex.

## Other Publications

The parallel-I/O interface is described in the publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information*, GA22-6974.

The parallel-I/O channel-to-channel adapter is described in the publication *IBM Enterprise Systems Architecture/390 Channel-to-Channel Adapter for the System/360 and System/370 I/O Interface*, SA22-7091.

The Enterprise Systems Connection Architecture* (ESCON*) I/O interface, referred to in this publication along with the FICON I/O interface as the serial I/O interface, is described in the publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

The FICON I/O interface is described in the ANSI® standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

The channel-to-channel adapter for the serial-I/O interface is described in the publication *IBM Enterprise Systems Architecture/390 ESCON Channel-to-Channel-Adapter*, SA22-7203.

The commands, status, and sense data that are common to all I/O devices that comply with

---

Enterprise Systems Connection Architecture and ESCON are trademarks of the International Business Machines Corporation.

ANSI is a registered trademark of the American National Standards Institute.



LzLabs GmbH. Ex. 1008-23

ESA/390 are described in the publication *IBM Enterprise Systems Architecture/390 Common I/O-Device Commands and Self Description*, SA22-7204.

Vector operations are described in the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207.

The compression facility is described in the publication *IBM Enterprise Systems Architecture/390 Data Compression*, SA22-7208.

The interpretive-execution facility is described in the publication *IBM 370-XA Interpretive Execution*, SA22-7095.

## Summary of Changes in Ninth Edition

The current, ninth edition of this publication differs from the previous edition principally by containing the definitions of the HFP-multiply-add/subtract facility and the message-security assist. The ninth edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 3, “Storage”:
  - The STORE FACILITY LIST facility list at real locations 200-203 is added. This is a correction to the eighth edition.
  - The primary segment-table designation (STD) in control register 1 attaches a segment-table entry even when the CPU is in the home-space mode, and the home STD in control register 13 attaches a segment-table entry even when the CPU is in the secondary-space mode.
- In Chapter 4, “Control”:
  - The relationships between ETR time (TOD-clock time), UTC, and International Atomic Time are described in a programming note on page 4-32.
  - The description of the SIGNAL PROCESSOR store-extended-status-at-address order is corrected to state that the address of the extended save area must be zero.
  - Code 0 of the SIGNAL PROCESSOR set-architecture order, and also a CPU reset due to activation of the load normal key when the CPU is in the z/Architecture architectural mode, are changed to save the current z/Architecture PSW when switching to the ESA/390 architectural mode. Also, code 2 of the order is added, and this restores, for CPUs other than the one executing SIGNAL PROCESSOR, the saved PSW when switching to the z/Architecture architectural mode, provided that the saved PSW has not been set to all zeros by certain resets.
- In Chapter 5, “Program Execution”:
  - The results when a PER instruction-fetching event occurs along with certain exceptions or exception conditions are clarified. See “Indication of PER Events Concurrently with Other Interruption Conditions” on page 4-24.
  - The change bit is not necessarily set to one currently with the related storage reference, as observed by other CPUs; it may be set to one before or after the reference, within certain limits. See “Storage-Key Accesses” on page 5-84 for a detailed description of when the change bit is set.
  - The five instructions of the message-security assist are added to the list of instructions having multiple-access references.
- In Chapter 6, “Interruptions,” the list of conditions causing a specification exception to be recognized is extended to include those caused by the message-security assist instructions.
- In Chapter 7, “General Instructions”:
  - The definition of EXTRACT PSW is corrected by stating that bit 32 of the current PSW is placed in bit position 0 of the second operand, and bits 1-31 of the second operand are set to zeros.
  - Five instructions provided by the message-security assist are added.
- In Chapter 10, “Control Instructions”:
  - The definition of LOAD ADDRESS SPACE PARAMETERS is clarified.



LzLabs GmbH. Ex. 1008-24

- The description of the bits set by STORE FACILITY LIST is clarified, and new bits are assigned.

- In Chapter 14, "I/O Instructions":
  - The definition of MODIFY SUBCHANNEL is modified.
  - The definition of SET CHANNEL MONITOR is modified.
- In Chapter 15, "Basic I/O Functions," the following changes are made to the subchannel-information-block (SCHIB):
  - Bit 29 of word 6 of the path-management-control word (PMCW) is defined as the measurement-block-format control.
  - Bit 30 of word 6 of the PMCW is defined as the extended-measurement-word-mode enablement bit.
  - The definition of words 10-11 (words 0-1 of the model-dependent area) are changed to contain a measurement-block address, when the extended-I/O-measurement-block facility is installed.
- In Chapter 16, "I/O Interruptions," the interruption-response block (IRB) is extended to include the extended-measurement word.
- In Chapter 17, "I/O Support Functions":
  - The requirement that the measurement block be updated when secondary status is accepted is clarified.
  - The extended-measurement-block facility is added.
  - The extended-measurement-word facility is added.
- In Chapter 18, "Hexadecimal-Floating-Point Instructions," the MULTIPLY AND ADD (four instructions) and MULTIPLY AND SUBTRACT (four instructions) instructions provided by the HFP-multiply-add/subtract facility are added.

The above changes may affect other chapters besides the ones listed. All technical changes to the text or to an illustration are indicated by a vertical line to the left of the change.

## Summary of Changes in Eighth Edition

The eighth edition of this publication differs from the previous edition principally by containing:

- Definitions of a number of new instructions introduced in z/Architecture and also added to ESA/390. These instructions include, but are not limited to, those of the extended-translation facility 2.
- Additional indirect-data-addressing functions introduced in z/Architecture and also added to ESA/390. These are a doubleword format-2 IDAW and the ability of all format-2 IDAWs of a channel program to specify either 2K-byte or 4K-byte data blocks.
- Definitions of new input/out facilities placed in z/Architecture and also added to ESA/390. These are the FICON-channel facility, the ORB-extension facility, and the channel-subsystem-I/O-priority facility.

The eighth edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 1, "Introduction":
  - The new z/Architecture instructions are highlighted.
  - The ability to simulate the instructions of the extended-translation facility 2 by means of the MVS CSRUNIC macro instruction is referenced.
  - The new I/O facilities and functions are highlighted.
- In Chapter 3, "Storage":
  - The description of the translation-lookaside buffer (TLB) is improved.
  - If z/Architecture is installed, LOAD REAL ADDRESS may use the TLB whether DAT is on or off, but TLB entries still are formed only if DAT is on.
  - Bit 29 of the translation-exception identification, real locations 144-147, and the operand access identification, real location 162, are described. These are related to a page-translation exception recognized by the MOVE PAGE instruction.


LzLabs GmbH. Ex. 1008-25

  - The store-status and machine-check architectural-mode identification at real and absolute locations 163 is added.
  - The I/O-interruption-identification word at real locations 192-195 is described.
- In Chapter 4, “Control”:
  - On a model on which z/Architecture is installed, recognition of a storage-alteration PER event causes no more than 4K bytes to be stored beginning with the byte that caused the event, and this may result in partial completion of an interruptible instruction.
  - BRANCH RELATIVE AND SAVE LONG and BRANCH RELATIVE ON CONDITION LONG are added to those instructions that cause a successful-branching PER event.
  - Storing of the architectural-mode identification during the store-status operation is described.
  - The set-architecture order of the SIGNAL PROCESSOR instruction is added. This can be used to set the architectural mode of the configuration to z/Architecture.
- In Chapter 5, “Program Execution”:
  - The RSE, RSL, and RIL instruction formats are added, and an $M_3$ field is described as an alternative in the RS format.
  - The description of the ART-lookaside buffer (ALB) is improved.
- In Chapter 6, “Interruptions,” the crypto-operation exception is added.
- In Chapter 7, “General Instructions”:
  - The following new instructions that have been placed in both z/Architecture and ESA/390 are added:
    - ADD LOGICAL WITH CARRY
    - BRANCH RELATIVE AND SAVE LONG
    - BRANCH RELATIVE ON CONDITION LONG
    - DIVIDE LOGICAL
    - EXTRACT PSW
    - LOAD ADDRESS RELATIVE LONG
    - LOAD REVERSED
    - MULTIPLY LOGICAL
    - ROTATE LEFT SINGLE LOGICAL
    - SET ADDRESSING MODE
    - STORE REVERSED
    - SUBTRACT LOGICAL WITH BORROW
    - TEST ADDRESSING MODE
  - The following instructions of the extended-translation facility 2 are added:
    - COMPARE LOGICAL LONG UNICODE
    - MOVE LONG UNICODE
    - PACK ASCII
    - PACK UNICODE
    - TRANSLATE ONE TO ONE
    - TRANSLATE ONE TO TWO
    - TRANSLATE TWO TO ONE
    - TRANSLATE TWO TO TWO
    - UNPACK ASCII
    - UNPACK UNICODE
  - The definitions of PACK ASCII, PACK UNICODE, UNPACK ASCII, and UNPACK UNICODE are clarified as compared to their definitions in *z/Architecture Principles of Operation*, SA22-7832-00.
  - It is clarified that the following instructions perform multiple-access references to their storage operands:
    - CHECKSUM
    - COMPARE AND FORM CODEWORD
    - CONVERT UNICODE TO UTF-8
    - CONVERT UTF-8 TO UNICODE
  - It is clarified that the following instructions do not necessarily process their storage operands left to right as observed by other CPUs: MOVE LONG, MOVE LONG EXTENDED, and MOVE LONG UNICODE (which is new in the current edition of this publication but appears in SA22-7832-00 without this clarification). Special padding characters of MOVE LONG and MOVE LONG EXTENDED specify whether left-to-right processing should be performed, as observed by other CPUs, and whether the data being moved should or should not be placed in the cache for availability for subsequent processing.
  - The MOVE INVERSE instruction is described as being basic in ESA/390, as opposed to being provided by a move-inverse facility.



LzLabs GmbH. Ex. 1008-26

- In Chapter 8, “Decimal Instructions,” the TEST DECIMAL instruction of the extended-translation facility 2 is added.
- In Chapter 10, “Control Instructions”:
  - The COMPARE AND SWAP AND PURGE, PAGE IN, and PAGE OUT instructions are described.
  - The STORE FACILITY LIST instruction of z/Architecture, which has been placed also in ESA/390, is added.
  - It is clarified that the following instructions perform multiple-access references to their storage operands:
    - LOAD ADDRESS SPACE PARAMETERS
    - RESUME PROGRAM
    - STORE SYSTEM INFORMATION
- In Chapter 11, “Machine-Check Handling,” storing of the architectural-mode identification during a machine-check interruption is described.
- In Chapter 12, “Operator Facilities,” the contents of the floating-point-control register can be altered and displayed.
- In Chapter 13, “I/O Overview,” FICON and FICON-converted I/O interfaces and the frame-multiplex mode are introduced.
- In Chapter 14, “I/O Instructions”:
  - The CANCEL SUBCHANNEL instruction is described.
  - TEST PENDING INTERRUPTION, when the second-operand address is zero, stores a three-word I/O-interruption code at real locations 184-195. The new third word contains an interruption-identification word that further identifies the source of the I/O interruption.
- In Chapter 15, “Basic I/O Functions”:
  - The ORB is extended to eight words and newly contains a streaming-mode control, modification control, synchronization control, format-2-IDAW control, 2K-IDAW control, ORB-extension control, channel-subsystem priority, and control-unit priority.
  - A doubleword format-2 IDAW and 4K-byte data blocks optionally designated by format-2 IDAWs are added.
- In Chapter 16, “I/O Interruptions”:
  - A secondary-CCW-address-validity bit and failing-storage-address-format bit are added to the extended-report word.
  - A two-word failing-storage address and a secondary-CCW address are added to the format-0 extended-status word.
- In Chapter 17, “I/O Support Functions”:
  - Control-unit-defer time is added. This has an effect on the device-connect time and device-disconnect time in the measurement block.
  - On a model with z/Architecture installed, references to the measurement block by the measurement-block-update facility are single-access references and appear to be word concurrent as observed by CPUs.
  - Device-active-only time is added to the measurement block.
  - The channel-subsystem-I/O-priority facility, providing channel-subsystem priority and control-unit priority, is added.

## Summary of Changes in Seventh Edition

The seventh edition of this publication differs from the previous edition principally by containing the definitions of the extended-TOD-clock, TOD-clock-control-override, extended-translation, and store-system-information facilities. The seventh edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 4, “Control”:
  - The ETR subclass mask, bit 27 of control register 0, and the TOD-clock-control-override control, bit 10 of control register 14, are added.
  - An extension to the TOD clock, and the TOD programmable register, are added.
  - Leap second 22 is added.
- In Chapter 6, “Interruptions”:
  - The ETR external interruption is added.


LzLabs GmbH. Ex. 1008-27

  - The TOD-clock-sync-check external interruption is affected by the extended-TOD-clock facility.

- In Chapter 7, "General Instructions," the CONVERT UNICODE TO UTF-8, CONVERT UTF-8 TO UNICODE, STORE CLOCK EXTENDED, and TRANSLATE EXTENDED instructions are added.
- In Chapter 10, "Control Instructions," the SET CLOCK PROGRAMMABLE FIELD and STORE SYSTEM INFORMATION instructions are added.

The above changes may affect other chapters besides the ones listed.

## Summary of Changes in Sixth Edition

The sixth edition of this publication differs from the previous edition principally by containing the definitions of the basic floating-point, floating-point-support, and hexadecimal-floating-point (HFP) extension facilities and the binary-floating-point (BFP), program-call-fast, resume-program, and trap facilities. The sixth edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 2, "Introduction," 12 floating-point registers and the floating-point-control register are added.
- In Chapter 3, "Storage":
  - In the section "Prefixing," the term "prefix area" is changed to mean the 4K-byte area designated by the prefix instead of real locations 0-4095. This change is consistent with how the term has been used in the definition of the SET PREFIX instruction.
  - Assigned storage locations for the PCF-entry-table origin, data-exception code, and machine-check and store-status extended-save-area address are added.
- In Chapter 4, "Control":
  - Bits 22 and 23 of the PSW are renamed the HFP-exponent-underflow mask and the HFP-significance mask, respectively.
  - The AFP-register control and extended-save-area control are added in the control registers.
  - RESUME PROGRAM and TRAP cause branch trace entries to be made.
  - RESUME PROGRAM and TRAP cause successful-branching PER events to occur and a valid ATMID (addressing-and-translation-mode identification) to be stored.
  - The use of an extended save area for saving floating-point registers 0-15 and the floating-point-control register by the store-status operation is added.
  - The store-extended-status-at-address SIGNAL PROCESSOR order is added.
- In Chapter 5, "Program Execution":
  - The RRF, RXE, and RXF instruction formats are added.
  - A trap-control-block address and TRAP-enabled bit are added to the dispatchable-unit control table.
- In Chapter 6, "Interruptions":
  - The exception names "exponent overflow," "exponent underflow," "significance," "floating-point divide," and "square root" are changed to "HFP exponent overflow," "HFP exponent underflow," "HFP significance," "HFP divide," and "HFP square root," respectively.
  - The reasons for recognizing a data exception are expanded to include reasons related to the new floating-point facilities. When a data exception is recognized for an old reason related to decimal operands, it is called a decimal-operand data exception. The detailed description of the reasons for recognizing a decimal-operand data exception is moved from Chapter 6 to Chapter 8, "Decimal Instructions."
  - When a program interruption for a data exception occurs, a data-exception code (DXC) may be stored at real location 147 and placed in the floating-point-control (FPC) register to indicate the reason for the exception.
  - The instruction ending for a data exception may be suppression when previously



LzLabs GmbH. Ex. 1008-28

it was termination, depending on the model. The ending may be completion in new cases related to floating point.

  - The figure "Priority of Access Exceptions" has superscripts added indicating when an exception is not applicable when not in the access-register mode.

- In Chapter 7, "General Instructions," additional details are added to the definition of the PERFORM LOCKED OPERATION instruction.
- In Chapter 8, "Decimal Instructions," the reasons for recognizing a decimal-operand data exception are described (the definition is moved to here from Chapter 6).
- Chapter 9, "Floating-Point Overview and Support Instructions," replaces the previous Chapter 9, "Floating-Point Instructions." The new Chapter 9 introduces the BFP and HFP operations, defines instructions that are common to BFP and HFP or that convert between BFP and HFP data formats, and summarizes all floating-point instructions. Other contents of the old Chapter 9 are moved to Chapter 18, "Hexadecimal-Floating-Point Instructions."
- In Chapter 10, "Control Instructions":
  - The PROGRAM CALL FAST, RESUME PROGRAM, and TRAP instructions are added.
  - The method of description used in the figure "Priority of Execution: PROGRAM RETURN" is changed. The technical content of the figure is not changed.
- In Chapter 11, "Machine-Check Handling," the storing of 16 floating-point registers and the floating-point-control register in an extended save area during a machine-check interruption is added.
- Chapter 18, "Hexadecimal-Floating-Point Instructions," is new. It contains definitions of old and new instructions having operands in the HFP data format. Also, alternate mnemonics are assigned to some forms of the LOAD ROUNDED and MULTIPLY instructions.
- Chapter 19, "Binary-Floating-Point (BFP) Instructions," is new. It contains definitions of new instructions having operands in the BFP data format.
- In Appendix A, "Number Representation and Instruction-Use Examples":
  - The term "floating-point number" is changed to "hexadecimal-floating-point number."
  - An example of the use of the PERFORM LOCKED OPERATION instruction is added.

The above changes may affect other chapters besides the ones listed.

## Summary of Changes in Fifth Edition

The fifth edition of this publication differs from the previous edition principally by containing the definitions of the branch-and-set-authority facility and the perform-locked-operation facility. The fifth edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 3, "Storage," in the section "Handling of Addresses," the dispatchable-unit and primary-space access-list origins and the authority-table origin used by access-register translation are unpredictably real or absolute addresses instead of real addresses.
- In Chapter 4, "Control":
  - Leap second 21 is added.
  - The effects of clear reset and power-on reset on the locks used by PERFORM LOCKED OPERATION are described.
- In Chapter 5, "Program Execution":
  - The section "Subroutine Linkage without the Linkage Stack" is enhanced to describe a calling linkage made by the PROGRAM TRANSFER instruction when the purpose is to reduce authority. The BRANCH AND SET AUTHORITY instruction then is introduced.
  - The change described for Chapter 3 appears throughout the section "Access-Register Translation."
- In Chapter 6, "Interruptions," the definitions of the privileged-operation, protection, special-operation, and specification exceptions are added to or corrected.


LzLabs GmbH. Ex. 1008-29

- In Chapter 7, “General Instructions,” the PERFORM LOCKED OPERATION instruction is added.
- In Chapter 10, “Control Instructions,” the BRANCH AND SET AUTHORITY instruction is added.

The above changes may affect other chapters besides the ones listed.

## Summary of Changes in Fourth Edition

The fourth edition of this publication differs from the previous edition principally by containing the definitions of the following facilities: called-space identification, checksum, compare and move extended, and immediate and relative instruction. The fourth edition also contains additional information about the PER-2 facility, and it describes the ancillary-report bit in certain fields. The fourth edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 4, “Control”:
  - Descriptions of an additional bit in the PER code, the addressing-and-translation-mode identification, and the PER STD identification are added.
  - Leap second 20 is added.
- In Chapter 5, “Program Execution”:
  - Instruction formats RI and RSI are added.
  - Relative branching is added.
  - The section “Condition-Code Alternative to Interruptibility” is added.
  - The called-space identification in the linkage-stack state entry formed by the stacking PROGRAM CALL instruction is added.
  - It is clarified that a storage-operand fetch reference for an instruction can precede the execution of the instruction by an unlimited amount of time.
  - A programming note showing effects when CPU serialization is or is not performed is added.
- In Chapter 7, “General Instructions”:
  - The CHECKSUM instruction is added.
  - The COMPARE LOGICAL LONG EXTENDED and MOVE LONG EXTENDED instructions of the compare-and-move-extended facility are added.
  - The instructions of the immediate-and-relative-instruction facility are added. These are:
    - ADD HALFWORD IMMEDIATE
    - BRANCH RELATIVE AND SAVE
    - BRANCH RELATIVE ON CONDITION
    - BRANCH RELATIVE ON COUNT
    - BRANCH RELATIVE ON INDEX HIGH
    - BRANCH RELATIVE ON INDEX LOW OR EQUAL
    - COMPARE HALFWORD IMMEDIATE
    - LOAD HALFWORD IMMEDIATE
    - MULTIPLY SINGLE (two instructions)
    - MULTIPLY HALFWORD IMMEDIATE
    - TEST UNDER MASK HIGH
    - TEST UNDER MASK LOW
- In Chapter 10, “Control Instructions,” in the STORE CPU ID definition, the term “model number” is changed to “machine-type number,” and programming notes about the version code and CPU identification number are added.
- In Chapter 11, “Machine-Check Handling,” the ancillary-report bit in the machine-check-interruption code is described.
- In Chapter 16, “I/O Interruptions,” the ancillary-report bit in the subchannel logout is described.
- In Chapter 17, “I/O Support Functions,” the ancillary-report bit in the channel-report word is described.

The above changes may affect other chapters besides the ones listed.

## Summary of Changes in Third Edition

The third edition of this publication differs from the previous edition principally by containing the definition of the subspace-group facility. The third edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 3, “Storage”:



LzLabs GmbH. Ex. 1008-30

  - The virtual-address enhancement of suppression on protection is added.
  - Fields of the subspace-group facility are added to the ASN-second-table entry and the segment-table designation.
  - For CPU table entries that are addressed by real or absolute addresses, it is unpredictable whether the address wraps or an addressing exception is recognized.
  - All zeros may be stored at real location 160 during a subspace-replacement operation.
- In Chapter 4, “Control”:
  - A trace entry for BRANCH IN SUBSPACE GROUP is added.
  - An instruction-fetching PER event for an interruptible instruction may be discarded under certain conditions when a unit of operation of the instruction remains to be executed.
  - Leap seconds 18 and 19 are added.
- In Chapter 5, “Program Execution”:
  - The BRANCH IN SUBSPACE GROUP instruction is introduced in “Subroutine Linkage without the Linkage Stack” on page 5-10.
  - Fields for the subspace-group facility are added to the dispatchable-unit control table and the ASN-second-table entry.
  - Effects of the subspace-group facility on the instructions PROGRAM CALL, PROGRAM TRANSFER, PROGRAM RETURN, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS are introduced in “Subspace-Replacement Operations” on page 5-59.
- In Chapter 6, “Interruptions”:
  - It is clarified that an instruction is considered to be executed even if it has an odd instruction address or cannot be fetched because of an access exception.
  - Subspace-replacement exceptions (a collective name) are added.
- In Chapter 8, “Decimal Instructions,” for ZERO AND ADD when the operands overlap and the rightmost byte of the first operand is to the left of the rightmost byte of the second operand, a data exception may or may not be recognized.
- In Chapter 10, “Control Instructions”:
  - The BRANCH IN SUBSPACE GROUP instruction is added, and subspace-replacement operations are added to the definitions of PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS.
  - The address placed in general register $R_1$ when LOAD REAL ADDRESS sets a nonzero condition code is real or absolute in accordance with the type of address used during the attempted translation.
  - For MOVE PAGE (facility 2), when the first operand is valid in main storage and the second operand is valid in an unavailable expanded-storage block, a storage-alteration PER event may be recognized, and the change bit may be set, for the first operand even though the first-operand location remains unchanged.
- In Chapter 12, “Operator Facilities,” a model may have, as an alternative to a wait indicator, a means of indicating a time-averaged value of the PSW wait-state bit.
- In Chapter 16, “I/O Interruptions”:
  - The initial status that causes a sequence code of 010 binary to be placed in the subchannel logout is described in “Sequence Code (SC)” on page 16-35.
  - An authorization-check bit is added to the extended-report word in a format-0 extended-status word. The bit indicates, when one, that the start or resume function was terminated because the channel subsystem is in the isolated state.
- In Chapter 17, “I/O Support Functions”:
  - Additional conditions under which the device-connect-time field in the measurement block is not updated are described in “Device-Connect Time” on page 17-5.
  - It is clarified that an IPL program should not be placed in the low 512 bytes of storage because that area is reserved.


LzLabs GmbH. Ex. 1008-31

- In Appendix I, “EBCDIC and Other Codes,” a chart showing control codes and a 94-character EBCDIC character set is replaced by a table showing control codes, various EBCDIC character sets and code pages, ASCII, ISO-8, and IBM-PC code pages, and BookMaster* symbols.

The above changes may affect other chapters besides the ones listed.

## Summary of Changes in Second Edition

The second edition of this publication contains minor clarifications and corrections and also the following significant changes relative to the previous edition with TNL SN22-5400:

- In Chapter 3, “Storage”:
  - The suppression-on-protection facility is defined.
  - The checking of bits 28-31 or 26-31 in the ASN-first-table entry and bits 30, 31, and 60-63 in the ASN-second-table entry is made optional.
- In Chapter 4, “Control”:
  - It is made unpredictable whether an instruction-fetching PER event is indicated for the first halfword of an instruction when an access exception is recognized for the first halfword.
  - The standard epoch for the time-of-day (TOD) clock is described in terms of Coordinated Universal Time instead of Greenwich Mean Time.
- In Chapter 7, “General Instructions”:
  - The instructions of the string-instruction facility, COMPARE LOGICAL STRING, MOVE STRING, and SEARCH STRING, are added.
  - The COMPARE UNTIL SUBSTRING EQUAL instruction is added. This instruction was introduced in ESA/370 but has not previously been described.
- In Chapter 10, “Control Instructions,” the SET ADDRESS SPACE CONTROL FAST instruction is added.
- In Chapter 12, “Operator Facilities,” a definition of the effect of initial machine loading (IML) on expanded storage is added.
- In Chapter 17, “I/O Support Functions,” the control-unit-queuing-measurement facility is added.
- In Appendix A, “Number Representation and Instruction-Use Examples”:
  - Examples of the use of the instructions of the string-instruction facility are added.
  - A description of the tree used by the sorting instructions, COMPARE AND FORM CODEWORD and UPDATE TREE, and an example of the use of the sorting instructions are added.

The above changes may affect other chapters besides the ones listed.

BookMaster is a trademark of the International Business Machines Corporation.



LzLabs GmbH. Ex. 1008-32

# Chapter 1. Introduction


Highlights of ESA/390 . . . . . 1-1
The ESA/370 and 370-XA Base . . . . . 1-10
System Program . . . . . 1-12
Compatibility . . . . . 1-12
Compatibility among ESA/390 Systems . . 1-12
Compatibility among ESA/390, ESA/370, 370-XA, and System/370 . . . . . 1-13
Control-Program Compatibility . . . . . 1-13
Problem-State Compatibility . . . . . 1-13
Availability . . . . . 1-14


This publication provides, for reference purposes, a detailed Enterprise Systems Architecture/390 (ESA/390) description.

The architecture of a system defines its attributes as seen by the programmer, that is, the conceptual structure and functional behavior of the machine, as distinct from the organization of the data flow, the logical design, the physical design, and the performance of any particular implementation. Several dissimilar machine implementations may conform to a single architecture. When the execution of a set of programs on different machine implementations produces the results that are defined by a single architecture, the implementations are considered to be compatible for those programs.

## Highlights of ESA/390

ESA/390 is the next step in the evolution from the System/360 to the System/370, System/370 extended architecture (370-XA), and Enterprise Systems Architecture/370 (ESA/370). ESA/390 includes all of the facilities of ESA/370 and provides a broad range of extensions. Some of these extensions either apply directly to application-program development or are basic machine interfaces, and they are described in detail in either this publication or another generally available publication. The remaining extensions are suitable for use only by means of specialized control or support programs, and detailed descriptions of these extensions are not provided.

ESA/390 is followed by the z/Architecture architectural mode. See *z/Architecture Principles of Operation*, SA22-7832, for a description of z/Architecture.

All extensions to ESA/370 that form ESA/390 are summarized below. For those extensions described in detail in this publication, a comparison of the differences among ESA/390, ESA/370, 370-XA, and System/370 appears in Appendixes D, E, and F.

ESA/390 was announced in September, 1990. Any extension added subsequently has the date of its announcement in parentheses at the end of its summary.

The following extensions are described in detail in this publication:

- *Access-list-controlled protection* allows store-type storage references to an address space to be prohibited by means of a bit in the access-list entry used to access the space. Thus, different users having different access lists can have different capabilities to store in the same address space.
- The *program-event-recording facility 2 (PER 2)* is an alternative to the original PER facility, which is now named PER 1. PER 2 provides the option of having a successful-branching event occur only when the branch target is within the designated storage area, and it provides the option of having a storage-alteration event occur only when the storage area is within designated address spaces. The use of these options improves performance by allowing only PER events of interest to occur. PER 2 deletes the ability to monitor for general-register-alteration events.

  PER 2 includes extensions that provide additional information about PER events. The extensions were described in detail beginning in the fourth edition of this publication.
- *Concurrent sense* improves performance by allowing sense information to be presented at the time of an interruption due to a unit-check condition, thus avoiding the need for a separate I/O operation to obtain the sense information.

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-33

- *Broadcasted purging* provides the COMPARE AND SWAP AND PURGE instruction for conditionally updating tables associated with dynamic address translation and access-register translation and clearing associated buffers in multiple CPUs. This is described in detail beginning in the eighth edition of this publication.
- *Storage-protection override* provides a new form of subsystem storage protection that improves the reliability of a subsystem executed in an address space along with possibly erroneous application programs. When storage-protection override is made active by a control-register bit, fetches and stores by the CPU are permitted to storage locations having a storage key of 9 regardless of the access key used by the CPU. If the subsystem is in key-8 storage and is executed with a PSW key of 8, for example, and the application programs are in key-9 storage and are executed with a PSW key of 9, accesses by the subsystem to the application-program areas are permitted while accesses by the application programs to the subsystem area are denied. (September, 1991)
- *Move-page facility 2* extends the MOVE PAGE instruction introduced in ESA/370 by allowing use of a specified access key for either the source or the destination operand, by allowing improved performance when the destination operand will soon be referenced, and by allowing improved performance when an operand is invalid in both main and expanded storage. The ESA/370 version of MOVE PAGE is now called move-page facility 1 and is in Chapter 7, "General Instructions." MOVE PAGE of move-page facility 2 is in Chapter 10, "Control Instructions." (September, 1991)
- The *square-root facility* consists of the SQUARE ROOT instruction and the square-root exception. The instruction extracts the square root of a floating-point operand in either the long or short format. The instruction is the same as that provided on some models of the IBM 4341, 4361, and 4381 Processors. (September, 1991)
- The *string-instruction facility* (or *logical string assist*) provides instructions for (1) moving a string of bytes until a specified ending byte is found, (2) logically comparing two strings until an inequality or a specified ending byte is found, and (3) searching a string of a specified length for a specified byte. The first two instructions are particularly useful in a C program in which strings are normally delimited by an ending byte of all zeros. (June, 1992)
- The *suppression-on-protection facility* causes a protection exception due to page protection to result in suppression of instruction execution instead of termination of instruction execution, and it causes the address and an address-space identifier of the protected page to be stored in low storage. This is useful in performing the AIX/ESA* copy-on-write function, in which AIX/ESA causes the same page of different address spaces to map to a single page frame of real storage so long as a store in the page is not attempted and then, when a store is attempted in a particular address space, assigns a unique page frame to the page in that address space and copies the contents of the page to the new page frame. (February, 1993)
- The *set-address-space-control-fast facility* consists of the SET ADDRESS SPACE CONTROL FAST (SACF) instruction, which possibly can be used instead of the previously existing SET ADDRESS SPACE CONTROL (SAC) instruction, depending on whether all of the SAC functions are required. SACF, unlike SAC, does not perform the serialization and checkpoint-synchronization functions, nor does it cause copies of prefetched instructions to be discarded. SACF provides improved performance on some models. (February, 1993)
- The *subspace-group facility* includes the BRANCH IN SUBSPACE GROUP instruction, which can be used to give or return control from one address space to another in a group of address spaces called a subspace group, with this giving and returning of control being done with better performance than can be obtained by means of the PROGRAM CALL and PROGRAM RETURN or PROGRAM

AIX/ESA and CICS are trademarks of the International Business Machines Corporation.



LzLabs GmbH. Ex. 1008-34

TRANSFER instructions. One address space in the subspace group is called the base space, and the other address spaces in the group are called subspaces. It is intended that each subspace contain a different subset of the storage in the base space, that the base space and each subspace contain a subsystem control program, such as CICS*, and application programs, and that each subspace contain the data for a single transaction being processed under the subsystem control program. The placement of the data for each transaction in a different subspace prevents the processing of a transaction from erroneously damaging the data of other transactions. The data of the control program can be protected from the transaction processing by means of the storage-protection-override facility. (April, 1994)

- The *virtual-address enhancement of suppression on protection* provides that if dynamic address translation (DAT) was on when a protection exception was recognized, the suppression-on-protection result is indicated, and the address of the protected location is stored, only if the address is one that was to be translated by DAT; the suppression-on-protection result is not indicated if the address that would be stored is a real address. This enhancement allows the stored address to be translated reliably by the control program to determine if the exception was due to page protection as opposed to key-controlled protection. The enhancement extends the usefulness of suppression on protection to operating systems like MVS/ESA* that use key-controlled protection. (September, 1994)
- The *immediate-and-relative-instruction facility* includes 13 new instructions, most of which use a halfword-immediate value for either signed-binary arithmetic operations or relative branching. The facility reduces the need for general registers, and, in particular, it eliminates the need to use general registers to address branch targets. As a result, the general registers and access registers can be allocated more efficiently in programs that require many registers. (September, 1996)
- The *compare-and-move-extended facility* provides new versions of the COMPARE LOGICAL LONG and MOVE LONG instructions. The new versions increase the size of the operand-length specifications from 24 bits to 32 bits, which can be useful when objects larger than 16M bytes are processed through the use of 31-bit addressing. The new versions also periodically complete to allow software polling in a multiprocessing system. (September, 1996)
- The *checksum facility* consists of the CHECKSUM instruction, which can be used to compute a 16-bit or 32-bit checksum in order to improve TCP/IP (transmission-control protocol/internet protocol) performance. (September, 1996)
- The *called-space-identification facility* improves serviceability by further identifying the called address space in a linkage-stack state entry formed by the PROGRAM CALL instruction. (September, 1996)
- The *branch-and-set-authority facility* consists of the BRANCH AND SET AUTHORITY instruction, which can be used to improve the performance of linkages within an address space by replacing PROGRAM CALL, PROGRAM TRANSFER, and SET PSW KEY FROM ADDRESS instructions. (June, 1997)
- The *perform-locked-operation facility* consists of the unprivileged PERFORM LOCKED OPERATION instruction, which appears to provide concurrent interlocked-update references to multiple storage operands. A function code of the instruction can specify any of six operations: compare and load, compare and swap, double compare and swap, compare and swap and store, compare and swap and double store, and compare and swap and triple store. The function code further specifies either word or doubleword operands. The instruction can be used to avoid the use of programmed locks in a multiprocessing system. (June, 1997)
- Four additional floating-point facilities improve the hexadecimal-floating-point (HFP) capability of the machine and add a binary-floating-point (BFP) capability. The facilities are:
  - *Basic floating-point extensions,* which provides 12 additional floating-point registers to make a total of 16 floating-point registers. This facility also includes a floating-point-control register and means for saving the contents of the new registers


LzLabs GmbH. Ex. 1008-35

during a store-status operation or a machine-check interruption.

  - *Floating-point-support (FPS) extensions,* which provides eight new instructions, including four to convert data between the HFP and BFP formats.
  - *Hexadecimal-floating-point (HFP) extensions,* which provides 26 new instructions to operate on data in the HFP format. All of these are counterparts to new instructions provided by the BFP facility, including conversion between floating-point and fixed-point formats, and a more complete set of operations on the extended format.
  - *Binary floating-point (BFP),* which defines short, long, and extended binary-floating-point (BFP) data formats and provides 87 new instructions to operate on data in these formats. The BFP formats and operations provide everything necessary to conform to the IEEE standard (ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, dated August 12, 1985) except for conversion between binary-floating-point numbers and decimal strings, which must be provided in software.

  (May, 1998)

- The *program-call-fast facility* provides the PROGRAM CALL FAST instruction, which is a variation of the stacking PROGRAM CALL instruction. PROGRAM CALL FAST omits certain authorization checking and state changes and also tracing, with the result that its performance is improved relative to that of stacking PROGRAM CALL. (May, 1998)

  The program-call-fast facility is not provided on machines having the z/Architecture architectural mode installed.

- The *resume-program facility* consists of the RESUME PROGRAM instruction, which restores, from a specified save area, the instruction address and certain other fields in the current PSW and also the contents of an access-and-general-register pair. RESUME PROGRAM allows a problem-state interruption-handling program to restore the state of an interrupted program and return to that program despite that a register is required for addressing the save area from which the state is restored. (May, 1998)

- The *trap facility* provides the TRAP instructions (a two-byte TRAP2 instruction and a four-byte TRAP4 instruction) that can overlay instructions in an application program to give control to a program that can perform fix-up operations on data being processed, such as dates that may be a "Year-2000" problem. RESUME PROGRAM can be used to return from the fix-up program. TRAP and RESUME PROGRAM can improve performance by avoiding program interruptions that would otherwise be needed to give control to and from the fix-up program. (May, 1998)

- The *extended-TOD-clock facility* includes (1) an extension of the TOD clock from 64 bits to 104 bits, allowing greater resolution; (2) a TOD programmable register, which contains a TOD programmable field that can be used to identify the configuration providing a TOD-clock value in a sysplex; (3) the SET CLOCK PROGRAMMABLE FIELD instruction, for setting the TOD programmable field in the TOD programmable register; and (4) the STORE CLOCK EXTENDED instruction, which stores both the longer TOD-clock value and the TOD programmable field. STORE CLOCK EXTENDED can be used in the future when the TOD clock is further extended to contain time values that exceed the current year-2042 limit (when there is a carry out of the current bit 0 of the TOD clock). (August, 1998)

- The *TOD-clock-control-override facility* provides a control-register bit that allows setting the TOD clock under program control, without use of the manual TOD-clock control of any CPU. (August, 1998)

- The *store-system-information facility* provides the privileged STORE SYSTEM INFORMATION instruction, which can be used to obtain information about a component or components of a virtual machine, a logical partition, or the basic machine. (January, 1999)

- The *extended-translation facility*, now called the extended-translation facility 1, includes the CONVERT UNICODE TO UTF-8, CONVERT UTF-8 TO UNICODE, and TRANSLATE EXTENDED instructions, all of which can improve performance. TRANSLATE



LzLabs GmbH. Ex. 1008-36

EXTENDED can be used in place of a TRANSLATE AND TEST instruction that locates an escape character, followed by a TRANSLATE instruction that translates the bytes preceding the escape character. (April, 1999)

- Certain *new z/Architecture instructions* are available on a model in the ESA/390 architectural mode when z/Architecture is installed. These new instructions are highlighted as follows:
  - ADD LOGICAL WITH CARRY and SUBTRACT LOGICAL WITH BORROW operate on 32-bit unsigned binary integers and include a carry or borrow, as represented by the leftmost bit of the two-bit condition code in the PSW, in the computation. This can improve the performance of operating on extended-precision integers (integers longer than 32 bits).
  - BRANCH RELATIVE AND SAVE LONG and BRANCH RELATIVE ON CONDITION LONG are like the BRANCH RELATIVE AND SAVE and BRANCH RELATIVE ON CONDITION instructions except that the new instructions use a 32-bit immediate field. This increases the target range available through relative branching.
  - DIVIDE LOGICAL uses a 64-bit unsigned binary dividend and a 32-bit unsigned binary divisor to produce a 32-bit quotient and remainder. MULTIPLY LOGICAL uses a 32-bit unsigned binary multiplicand and multiplier to produce a 64-bit product.
  - EXTRACT PSW extracts the left half of the current PSW and the addressing-mode bit to allow determination of the current machine state, for example, determination of whether the CPU is in the problem state or the supervisor state.
  - LOAD ADDRESS RELATIVE LONG forms an address relative to the current (unupdated) instruction address by means of a signed 32-bit immediate field.
  - LOAD REVERSED and STORE REVERSED load or store a two-byte or four-byte unit in storage with the left-to-right sequence of the bytes reversed. LOAD REVERSED also can move a four-byte unit between two general registers. These operations allow conversion between “little-endian” and “big-endian” formats.
  - ROTATE LEFT SINGLE LOGICAL obtains 32 bits from a general register, rotates them (the leftmost bit replaces the rightmost bit), and places the result in another general register (a nondestructive rotate).
  - SET ADDRESSING MODE can set either of the 24-bit and 31-bit addressing modes.
  - STORE FACILITY LIST is a privileged instruction that stores at real location 200 an indication of whether z/Architecture is installed and of whether it is active. This instruction also stores an indication of whether the new z/Architecture instructions that have been added to ESA/390 are available. Real location 200 has previously contained all zeros in most systems and normally can be examined by a problem-state program whether or not STORE FACILITY LIST is installed. The information stored at real location 200 also indicates whether the extended-translation facility 2 is installed.
  - TEST ADDRESSING MODE sets the condition code to indicate whether bit 32 of the current PSW specifies the 24-bit addressing mode or the 31-bit addressing mode.

  Also, SIGNAL PROCESSOR has a new order that can be used to switch all CPUs in the configuration either from the ESA/390 architectural mode to the z/Architecture architectural mode or from z/Architecture to ESA/390. (A system that is to operate using z/Architecture must first be IPLed in the ESA/390 mode.) When going from ESA/390 to z/Architecture, either the ESA/390 PSW can be transformed to a z/Architecture PSW or, in certain cases, the previously existing z/Architecture PSW can be restored. (October, 2000)
- The *extended-translation facility 2* performs operations on double-byte, ASCII, and decimal data. The double-byte data may be


LzLabs GmbH. Ex. 1008-37

Unicode™ data — data that uses the binary codes of the Unicode Worldwide Character Standard and enables the use of characters of most of the world's written languages. The facility provides the following instructions:

COMPARE LOGICAL LONG UNICODE
MOVE LONG UNICODE
PACK ASCII
PACK UNICODE
TEST DECIMAL
TRANSLATE ONE TO ONE
TRANSLATE ONE TO TWO
TRANSLATE TWO TO ONE
TRANSLATE TWO TO TWO
UNPACK ASCII
UNPACK UNICODE

The extended-translation facility 2 is called facility 2 since an extended-translation facility, now called facility 1, was introduced previously. Facility 1 provides the instructions:

CONVERT UNICODE TO UTF-8
CONVERT UTF-8 TO UNICODE
TRANSLATE EXTENDED

For when either or both of facility 1 and facility 2 are not installed on the machine, both facilities are simulated by the MVS CSRUNIC macro instruction, which is provided in OS/390* Release 10 and z/OS*.

*OS/390 MVS Assembler Services Reference*, GC28-1910-10, contains programming requirements, register information, syntax, return codes, and examples for the CSRUNIC macro instruction.

When CSRUNIC is used, the program exceptions listed in this publication do not cause program interruptions; instead, the exception conditions are indicated by CSRUNIC by means of return codes, as described in GC28-1910-10.

(October, 2000)

- Additional I/O functions and facilities are available on a model in the ESA/390 architectural mode when z/Architecture is installed, as follows:

  – Indirect data addressing is enhanced by the provision of a doubleword format-2 IDAW that is intended to allow operations on data at or above the 2G-byte absolute-address boundary in z/Architecture. The previously existing IDAW, a word containing a 31-bit address, is now called a format-1 IDAW. The format-2 IDAW contains a 64-bit address. A bit in the operation-request block (ORB) associated with a channel program specifies whether the program uses format-1 or format-2 IDAWs. A further enhancement, with direct applicability to ESA/390, is the ability of all format-2 IDAWs of a channel program to specify either 2K-byte or 4K-byte data blocks, as determined by another bit in the ORB. The use of 4K-byte blocks improves the efficiency of data transfers.

  – The *FICON-channel facility* provides the capabilities of attaching FICON-I/O-interface and FICON-converted-I/O-interface channel paths and of fully utilizing these channel-path types. FICON channel paths can significantly enhance overall data throughput by providing increased data-transfer rates in comparison to ESCON channel paths and by allowing multiple commands and associated data to be “streamed” to control units, thus further improving performance. The facility supports the following additional control mechanisms:

    - The modification-control bit in the ORB allows the program to optimize the performance of FICON channel paths when dynamically modifying channel programs.
    - The synchronization-control bit in the ORB ensures data integrity along with maximum channel-path performance by delaying the execution of a write command until the completion of an immediately preceding read command when performing unlimited prefetching

Unicode is a trademark of Unicode, Inc.

OS/390 and z/OS are trademarks of the International Business Machines Corporation.



LzLabs GmbH. Ex. 1008-38

of CCWs and when the data to be written may be the data read.

- The streaming-mode-control bit in the ORB allows the program to prevent command streaming in cases that require such prevention.
- The secondary-CCW-address field in the extended-status word assists in the recovery of channel programs that terminate abnormally when command streaming to a control unit is being perfomed. The field identifies a CCW that failed at the control unit.

– The *ORB-extension facility* expands the size of the ORB from three words to eight words. This makes fields available for use by the channel-subsystem-I/O-priority facility.

– The *channel-subsystem-I/O-priority facility* allows the program to establish a priority relationship among subchannels that have pending I/O operations. The priority relationship specifies the order in which I/O operations are initiated by the channel subsystem. Additionally, for fibre-channel-attached control units, the facility allows the program to specify the priority in which I/O operations pending at the control unit are performed.

(October, 2000)

- The *HFP-multiply-add/subtract facility* provides instructions for improved processing of hexadecimal floating-point numbers. The MULTIPLY AND ADD (or SUBTRACT) instruction is intended to be used in place of MULTIPLY followed by ADD (or SUBTRACT) NORMALIZED. (October, 2001)
- The *message-security assist* (MSA) may be available on a model. The MSA basic facility includes the following instructions:
  – CIPHER MESSAGE
  – CIPHER MESSAGE WITH CHAINING
  – COMPUTE INTERMEDIATE MESSAGE DIGEST
  – COMPUTE LAST MESSAGE DIGEST
  – COMPUTE MESSAGE AUTHENTICATION CODE

  Also included are five query functions and two functions for generating a message digest based on the secure-hash algorithm (SHA-1). The five query functions, one for each instruction, are used to determine the additional installed MSA facilities, which may include the following.
  – The MSA data-encryption-algorithm (DEA) facility consists of nine functions for ciphering messages, with or without chaining, and for generating a message-authentication code (MAC) using a 56-bit, 112-bit, or 168-bit cryptographic key.[1] All of these functions are based on the DEA algorithm.

  (June, 2003)
- The *extended-I/O-measurement-block facility* may be available on models implementing z/Architecture. The facility includes the following features:
  – A new format of the channel-measurement block. The new measurement block, termed a format-1 channel-measurement block, is expanded to 64 bytes and is addressed using a separate measurement-block address for each subchannel. The new measurement-block format provides additional measurement information and the flexibility to store the measurement blocks in non-contiguous, real storage.
  – The previously existing channel-measurement block is termed a format-0 channel-measurement block. A device-busy-time field is added to the format-0 channel-measurement block.

  (June, 2003)
- The *extended-I/O-measurement-word facility* may be available on models implementing z/Architecture. The extended-measurement-word (EMW) is an extension to the interruption-response block (IRB) and allows channel-measurement data to be provided on an I/O operation basis. This reduces

[1] These key lengths reflect the cryptographic strength. In subsequent chapters, they are referred to as 64-bit, 128-bit, or 192-bit, respectively, to include the DEA-key-parity bits.


LzLabs GmbH. Ex. 1008-39

program overhead by alleviating the previous requirement that the program fetch the measurement block before and after an operation and calculate the difference between the respective measurement data values. (June, 2003)

The following extensions are described in detail in other publications:

- The *Enterprise Systems Connection Architecture (ESCON)* introduces a new type of channel that uses an optical-fiber communication link between channels and control units. Information is transferred serially by bit, at 200 million bits per second, up to a maximum distance of 60 kilometers. The optical-fiber technology and serial transmission simplify cabling and improve reliability. See the publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.
- The *ESCON channel-to-channel adapter (ESCON CTCA)* provides the same type of function for serial channel paths as is available for the parallel-I/O-interface channel paths. See the publication *IBM Enterprise Systems Architecture/390 ESCON Channel-to-Channel Adapter*, SA22-7203.
- *I/O-device self-description* allows a device to describe itself and its position in the I/O configuration. See the publication *IBM Enterprise Systems Architecture/390 Common I/O-Device Commands and Self Description*, SA22-7204.
- *Vector extensions* include an instruction for loading the vector interruption index independent of the rest of the vector-status register, four instructions for multiplying and then adding or subtracting, and four instructions for extracting the square root. See the publication *IBM Enterprise Systems Architecture/390 Vector Operations,* SA22-7207. (September, 1991)

  The vector facility, to which the vector extensions were added, is not provided on current models.
- The *compression facility* performs a Ziv-Lempel type of compression and expansion by means of static (nonadaptive) dictionaries that are to be prepared by a program before the compression and expansion operations. Because the dictionaries are static, the compression facility can provide good compression not only for long sequential data streams (for example, archival or network data) but also for randomly accessed short records (for example, 80 bytes). See the publication *IBM Enterprise Systems Architecture/390 Data Compression*, SA22-7208. (February, 1993)

The remaining extensions of ESA/390, for which detailed descriptions are not provided, are as follows:

- The integrated *cryptographic facility* provides a number of instructions to protect data privacy, to support message authentication and personal identification, and to facilitate key management. The high-performance cipher capability of the facility is designed for financial-transaction and bulk-encryption environments, and it complies with the Data Encryption Standard (DES).
  - Usability of the cryptographic facility is extended to virtual-machine environments, which allows the facility to be used by MVS/ESA being executed under VM/ESA*, which in turn may be executed either under another VM/ESA or in a logical partition. (September, 1991)
- The *external-time-reference facility* provides a means to initiate and maintain the synchronization of TOD clocks to an external time reference (ETR). Synchronization tolerance of a few microseconds can be achieved, and the effect of leap seconds is taken into account. The facility consists of an ETR sending unit (Sysplex Timer*), which may be duplexed, two or more ETR receiving units, and optical-fiber cables. The cables are used to connect the ETR sending unit, which is an external device, to ETR receiving units of the configuration. CPU instructions are provided for setting the TOD clock to the value supplied by the ETR sending unit.
  - The ETR *automatic-propagation-delay-adjustment* function adjusts the time signals from the ETR to

MVS/ESA, VM/ESA, Sysplex Timer, and DB2 are trademarks of the International Business Machines Corporation.



LzLabs GmbH. Ex. 1008-40

the attached processors to compensate for the propagation delay on the cables to the processors, thus allowing the cables to be of different lengths. (September, 1991)

  - The ETR *external-time-source* function synchronizes the ETR to a time signal received from a remote location by means of a telephone or radio. (September, 1991)

- *Extended sorting* provides instructions that improve the performance of the DB2* sorting function.
- Other *PER extensions,* besides those described beginning in the fourth edition of this publication, are an augmentation of PER 2 that provide additional PER function in the interpretive-execution mode.
- *Channel-subsystem call* provides various functions for use in the management of the I/O configuration. Some of the functions acquire information about the configuration from the accessible elements of the configuration, while others dynamically change the configuration.
- The *cancel-I/O facility* allows the program to withdraw a pending start function from a designated subchannel without signaling the device, which is useful in certain error-recovery situations. (September, 1991)

  The cancel-I/O facility provides the CANCEL SUBCHANNEL instruction and is described in detail beginning in the eighth edition of this publication.
- The *operational extensions* are a number of other improvements that result in increased availability and ease of use of the system, as follows:

  - *Automatic-reconfiguration* permits an operating system in an LPAR partition to declare itself willing to be terminated suddenly, usually to permit its storage and CPU resources to be acquired by an adjacent partition that is dynamically absorbing the work load of another system that has failed. Other functions deactivate and reset designated participating partitions.
  - A new *storage-reconfiguration* command decreases the time needed to reconfigure storage by allowing multiple requests for reconfiguration to be made by means of a single communication with the service processor.
  - *SCP-initiated reset* allows a system control program (SCP) to reset its I/O configuration prior to entering the disabled wait state following certain check conditions.
  - *Console integration* simplifies configuration requirements by reducing by one the number of consoles required by MVS.
  - The *processor-availability facility* enables a CPU experiencing an unrecoverable error that will cause a check stop to save its state and alert the other CPUs in the configuration. This allows, in many cases, another CPU to continue execution of the program that was in execution on the failing CPU. The facility is applicable in both the ESA/390 mode and the LPAR mode. (April, 1991)

- *Extensions for virtual machines* are a number of improvements to the interpretive-execution facility, as follows:

  - The *VM-data-space facility* provides for making the ESA/390 access-register architecture more useful in virtual-machine applications. The facility improves the ability to address a larger amount of data and to share data. For information on how VM/ESA uses the VM-data-space facility, see the publication *VM/ESA CP Programming Services*, SC24-5520.
  - A new *storage-key function* improves performance by removing the need for the previously used RCP area.
  - Interpreted SIE (available with region relocation) is improved to permit preferred guests under VM when VM itself is operating as a high-performance guest.
  - Other improvements include an optional special-purpose lookaside for some of the guest-state information and greater freedom in certain implementation choices.

- The *ESCON-multiple-image facility (EMIF)* allows multiple logical partitions to share ESCON channels (and FICON channels) and optionally to share any of the control units and associated I/O devices configured to these


LzLabs GmbH. Ex. 1008-41

shared ESCON channels. This can reduce ESCON-channel requirements, improve channel utilization, and improve I/O connectivity. (June, 1992)

- *PR/SM LPAR* mode is enhanced to allow up to 10 logical partitions in a single-image configuration and 20 in a physically-partitioned configuration. The previous limits were seven and 14, respectively. (June, 1992)

  On a machine with z/Architecture installed, PR/SM LPAR mode allows 15 logical partitions, and physical partitioning is not supported.

- The *asynchronous-pageout facility* consists of instructions for initiating and testing for completion of the asynchronous, to the CPU, transfer of a 4K-byte block of data from main storage to expanded storage. These instructions can be used to improve performance when a large amount of paging activity to expanded storage is required. (June, 1992)

  The asynchronous-pageout facility is not provided on machines having the z/Architecture architectural mode installed.

- The *asynchronous data mover* provides for transferring one or more groups of multiple contiguous pages from main storage to expanded storage or from expanded storage to main storage in a single operation. Similar to I/O operations, these transfers are performed largely asynchronous to instruction execution. This facility can improve processor performance when large groups of pages are moved between main storage and expanded storage. (February, 1993)

  The asynchronous data mover is not provided on machines having the z/Architecture architectural mode installed.

- The *coupling facility* enables high-performance data sharing among MVS/ESA systems that are connected by means of the facility. The coupling facility provides storage that can be dynamically partitioned for caching data in shared buffers, maintaining work queues and status information in shared lists, and locking data by means of shared lock controls. MVS/ESA services provide access to and manipulation of the coupling-facility contents. (April, 1994)

## The ESA/370 and 370-XA Base

ESA/390 includes the complete set of facilities of ESA/370 as its base. This section briefly outlines most of the facilities that were additions in 370-XA as compared to System/370 and that were additions in ESA/370 as compared to 370-XA.

The CPU-related facilities that were new in 370-XA are as follows:

- *Bimodal addressing* provides two modes of operation: a 24-bit addressing mode for the execution of old programs and a 31-bit addressing mode.
- *31-bit logical addressing* extends the virtual address space from the 16M bytes addressable with 24-bit addresses to 2G bytes (2,147,483,648 bytes).
- *31-bit real and absolute addressing* provides addressability for up to 2G bytes of main storage.
- The 370-XA *protection* facilities include key-controlled protection on only 4K-byte blocks, page protection, and, as in System/370, low-address protection for addresses below 512. Fetch-protection override eliminates fetch protection for locations 0-2047.
- The *tracing* facility assists in the determination of system problems by providing an ongoing record in storage of significant events.
- The COMPARE AND FORM CODEWORD and UPDATE TREE instructions facilitate sorting applications. An example of use is in Appendix A, “Number Representation and Instruction-Use Examples.”
- The *vector facility* is a high-performance means of performing numerically intensive computations. This facility is described in the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207. The vector facility is not provided on current models.
- The *interpretive-execution facility* allows creation of virtual machines that may operate according to several architectures and whose performance is enhanced because many virtual-machine functions are directly interpreted by the machine rather than simulated by the program. This facility is described in



LzLabs GmbH. Ex. 1008-42

the publication *IBM 370-XA Interpretive Execution*, SA22-7095.

- The *service-call-logical-processor (SCLP) facility* provides a means of communicating between the control program and the service processor for the purpose of describing and changing the configuration. This facility is not described.

The I/O-related differences between 370-XA and System/370 result from the 370-XA *channel subsystem*, which includes:

- *Path-independent addressing* of I/O devices, which permits the initiation of I/O operations without regard to which CPU is executing the I/O instruction or how the I/O device is attached to the channel subsystem. Any I/O interruption can be handled by any CPU enabled for it.
- *Path management*, whereby the channel subsystem determines which paths are available for selection, chooses a path, and manages any busy conditions encountered while attempting to initiate I/O processing with the associated devices.
- *Dynamic reconnection*, which permits any I/O device using this capability to reconnect to any available channel path to which it has access in order to continue execution of a chain of commands.
- *Programmable interruption subclasses*, which permit the programmed assignment of I/O-interruption requests from individual I/O devices to any one of eight maskable interruption queues.
- An *additional CCW format* for the direct use of 31-bit addresses in channel programs. The new CCW format, called format 1, is provided in addition to the System/370 CCW format, now called format 0.
- *Address-limit checking*, which provides an additional storage-protection facility to prevent data access to storage locations above or below a specified absolute address.
- *Monitoring facilities*, which can be invoked by the program to cause the channel subsystem to measure and accumulate key I/O-resource usage parameters.
- *Status-verification facility*, which reports inappropriate combinations of device-status bits presented by a device.
- A set of 13 *I/O instructions*, with associated control blocks, which are provided for the control of the channel subsystem.

The facilities appearing in System/370 but not provided in 370-XA are described in Appendix F.

The facilities that were new in ESA/370 are as follows:

- Sixteen *access registers* permit the program to have immediate access to storage operands in up to 16 2G-byte address spaces, including the address space in which the program resides. In a dynamic-address-translation mode named *access-register mode*, the instruction B field, or for certain instructions the R field, designates both a general register and an access register, and the contents of the access register, along with the contents of protected tables, specify the operand address space to be accessed. By changing the contents of the access registers, the program, under the control of an authorization mechanism, can have fast access to hundreds of different operand address spaces.
- A *linkage stack* is used in a functionally expanded mechanism for passing control between programs in either the same or different address spaces. This mechanism makes use also of the previously existing PROGRAM CALL instruction, an extended entry-table entry, and a new PROGRAM RETURN instruction. The mechanism saves various elements of status, including access-register and general-register contents, during a calling linkage, provides for changing the current status during the calling linkage, and restores the original status during the returning linkage. The linkage stack can also be used to save and restore access-register and general-register contents during a branch-type linkage performed by the new instruction BRANCH AND STACK.
- A translation mode named *home-space mode* provides an efficient means for the control program to obtain control in the address space, called the home address space, in


LzLabs GmbH. Ex. 1008-43

which the principal control blocks for a dispatchable unit (a task or process) are kept.

- The semiprivileged MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY instructions allow bidirectional movement of data between storage areas having different storage keys, without the need to change the PSW key.
- The privileged LOAD USING REAL ADDRESS and STORE USING REAL ADDRESS instructions allow the control program to access data in real storage more efficiently.
- The *private-space facility* allows an address space not to contain any common segments and causes low-address protection and fetch-protection override not to apply to the address space.
- The unprivileged MOVE PAGE instruction allows the program to move a page of data between main and expanded storage, provided that the source and destination pages are both valid. The ESA/370 version of MOVE PAGE is now called move-page facility 1.
- The *Processor Resource/Systems Manager* (PR/SM*)* feature provides support for multiple preferred guests under VM/XA and provides the logically partitioned (LPAR) mode, with the latter providing flexible partitioning of processor resources among multiple logical partitions. Certain aspects of the LPAR use of PR/SM are described in the publication *IBM ES/3090 Processor Complex Processor Resource/Systems Manager Planning Guide*, GA22-7123.
- The COMPARE UNTIL SUBSTRING EQUAL instruction provides improved performance of the compression of IMS log data sets and can be useful in other programs also. (The instruction is in Chapter 7, "General Instructions." It previously was not described.)

## System Program

ESA/390 is designed to be used with a control program that coordinates the use of system resources and executes all I/O instructions, handles exceptional conditions, and supervises scheduling and execution of multiple programs.

## Compatibility

## Compatibility among ESA/390 Systems

Although systems operating as defined by ESA/390 may differ in implementation and physical capabilities, logically they are upward and downward compatible. Compatibility provides for simplicity in education, availability of system backup, and ease in system growth. Specifically, any program written for ESA/390 gives identical results on any ESA/390 implementation, provided that the program:

1. Is not time-dependent.
2. Does not depend on system facilities (such as storage capacity, I/O equipment, or optional facilities) being present when the facilities are not included in the configuration.
3. Does not depend on system facilities being absent when the facilities are included in the configuration. For example, the program must not depend on interruptions caused by the use of operation codes or command codes that are not installed in some models. Also, it must not use or depend on fields associated with uninstalled facilities. For example, data should not be placed in an area used by another model for fixed-logout information. Similarly, the program must not use or depend on unassigned fields in machine formats (control registers, instruction formats, etc.) that are not explicitly made available for program use.
4. Does not depend on results or functions that are defined to be unpredictable or model-dependent or are identified as undefined. This includes the requirement that the

Processor Resource/Systems Manager and PR/SM are trademarks of the International Business Machines Corporation.



LzLabs GmbH. Ex. 1008-44

program should not depend on the assignment of device numbers and CPU addresses.

5. Does not depend on results or functions that are defined in the functional-characteristics publication for a particular model to be deviations from the architecture.
6. Takes into account any changes made to the architecture that are identified as affecting compatibility.

# Compatibility among ESA/390, ESA/370, 370-XA, and System/370

## Control-Program Compatibility

Control programs written for 370-XA or ESA/370 can be directly transferred to systems operating as defined by ESA/390. Almost all of the new functions that were introduced in ESA/370 are enabled only when a control-register bit assigned in ESA/370 and ESA/390 is set to one. When this bit is zero, the machine operates essentially as specified for 370-XA; the most significant exceptions are (1) instructions that load and store the contents of the access registers can be executed successfully, and (2) certain previously unassigned real and absolute storage locations below address 512 are stored in during the store-status operation, certain program interruptions, and the machine-check interruption. When the new control-register bit is zero, no unprivileged or semiprivileged instruction can place the CPU in the access-register mode, and so the access registers cannot be used to specify address spaces.

Control programs written for System/370 cannot be directly transferred to systems operating as defined by ESA/390. This is because in the 370-XA base of ESA/390 the basic-control mode is not present and the facilities for I/O and dynamic address translation are changed. (See Appendixes D, E, and F for a detailed comparison among ESA/390, ESA/370, 370-XA, and System/370.)

## Problem-State Compatibility

A high degree of compatibility exists at the problem-state level in going forward from ESA/370, 370-XA, or System/370 to ESA/390. Because the majority of a user's applications are written for the problem state, this problem-state compatibility is useful in many installations.

A problem-state program written for ESA/370, 370-XA, or System/370 operates with ESA/390, provided that the program:

1. Complies with the limitations described in “Compatibility among ESA/390 Systems” on page 1-12.
2. Is not dependent on privileged facilities which are unavailable on the system.
3. Takes into account other changes made to the System/370 architectural definition that affect compatibility between System/370 and the 370-XA base of ESA/390. These changes are described in Appendix F, “Comparison between System/370 and 370-XA.”

**Programming Notes:**

1. This publication assigns meanings to various operation codes, to bit positions in instructions, channel-command words, registers, and table entries, and to fixed locations in the low 512 bytes of storage. Unless specifically noted, the remaining operation codes, bit positions, and low-storage locations are reserved for future assignment to new facilities and other extensions of the architecture.

   To ensure that existing programs operate if and when such new facilities are installed, programs should not depend on an indication of an exception as a result of invalid values that are currently defined as being checked. If a value must be placed in unassigned positions that are not checked, the program should enter zeros. When the machine provides a code or field, the program should take into account that new codes and bits may be assigned in the future. The program should not use unassigned low-storage locations for keeping information since these locations may be assigned in the future in such a way that the machine causes the contents of the locations to be changed.
2. If a control program is used that does not support the use of access registers, a problem-state program under this control program still is able to load and store the contents of the access registers, and it might do so simply to use the access registers for data storage instead of for addressing. However, the use of access registers in such circumstances may be unsuccessful because the unsupporting control program does not save


LzLabs GmbH. Ex. 1008-45

and restore the contents of the access registers when switching between dispatchable units. Furthermore, the use of access registers in such circumstances may constitute a loss of security because the contents of access registers loaded by one dispatchable unit will be visible to other dispatchable units. To avoid the problems referred to here, a program using access registers must be executed only in a system with a control program that properly supports the use of access registers.

## Availability

Availability is the capability of a system to accept and successfully process an individual job. Systems operating in accordance with ESA/390 permit substantial availability by (1) allowing a large number and broad range of jobs to be processed concurrently, thus making the system readily accessible to any particular job, and (2) limiting the effect of an error and identifying more precisely its cause, with the result that the number of jobs affected by errors is minimized and the correction of the errors facilitated.

Several design aspects make this possible.

- A program is checked for the correctness of instructions and data as the program is executed, and program errors are indicated separate from equipment errors. Such checking and reporting assists in locating failures and isolating effects.
- The protection facilities, in conjunction with dynamic address translation and the separation of programs and data in different address spaces, permit the protection of the contents of storage from destruction or misuse caused by erroneous or unauthorized storing or fetching by a program. This provides increased security for the user, thus permitting applications with different security requirements to be processed concurrently with other applications.
- Dynamic address translation allows isolation of one application from another, still permitting them to share common resources. Also, it permits the implementation of virtual machines, which may be used in the design and testing of new versions of operating systems along with the concurrent processing of application programs. Additionally, it provides for the concurrent operation of incompatible operating systems.
- The use of access registers allows programs, data, and different collections of data to reside in different address spaces, and this further reduces the likelihood that a store using an incorrect address will produce either erroneous results or a system-wide failure.
- Multiprocessing and the channel subsystem permit better use of storage and processing capabilities, more direct communication between CPUs, and duplication of resources, thus aiding in the continuation of system operation in the event of machine failures.
- MONITOR CALL, program-event recording, and the timing facilities permit the testing and debugging of programs without manual intervention and with little effect on the concurrent processing of other programs.
- On most models, error checking and correction (ECC) in main storage, CPU retry, and command retry provide for circumventing intermittent equipment malfunctions, thus reducing the number of equipment failures.
- An enhanced machine-check-handling mechanism provides model-independent fault isolation, which reduces the number of programs impacted by uncorrected errors. Additionally, it provides model-independent recording of machine-status information. This leads to greater machine-check-handling compatibility between models and improves the capability for loading and operating a program on a different model when a system failure occurs.
- A small number of manual controls are required for basic system operation, permitting most operator-system interaction to take place *via* a unit operating as an I/O device and thus reducing the possibility of operator errors.
- The logical partitions made available by the PR/SM feature allow continued reliable production operations in one or more partitions while new programming systems are tested in other partitions. This is an advancement in particular for non-VM installations.
- The operational extensions and channel-subsystem-call facility of ESA/390 improve the ability to continue execution of application programs in the presence of system incidents



LzLabs GmbH. Ex. 1008-46

and the ability to make configuration changes with less disruption to operations.


LzLabs GmbH. Ex. 1008-47

LzLabs GmbH. Ex. 1008-48

# Chapter 2. Organization


Main Storage . . . . . 2-2
Expanded Storage . . . . . 2-2
CPU . . . . . 2-2
PSW . . . . . 2-3
General Registers . . . . . 2-3
Floating-Point Registers . . . . . 2-3
Floating-Point-Control (FPC) Register . . . 2-4
Control Registers . . . . . 2-4
Access Registers . . . . . 2-4
Vector Facility . . . . . 2-6
Cryptographic Facility . . . . . 2-6
External Time Reference . . . . . 2-6
I/O . . . . . 2-6
Channel Subsystem . . . . . 2-6
Channel Paths . . . . . 2-6
I/O Devices and Control Units . . . . . 2-7
Operator Facilities . . . . . 2-7


Logically, a system consists of main storage, one or more central processing units (CPUs), operator facilities, a channel subsystem, and I/O devices. I/O devices are attached to the channel subsystem through control units. The connection between the channel subsystem and a control unit is called a channel path.

A channel path employs either a parallel-transmission protocol or a serial-transmission protocol and, accordingly, is called either a parallel or a serial channel path. A serial channel path may connect to a control unit through a dynamic switch that is capable of providing different internal connections between the ports of the switch.

Expanded storage may also be available in the system, a vector or cryptographic unit may be included in a CPU, and an external time reference (ETR) may be connected to the system.

The physical identity of the above functions may vary among implementations, called “models.” Figure 2-1 depicts the logical structure of a two-CPU multiprocessing system that includes expanded storage, a vector unit, and a cryptographic unit and that is connected to an ETR.

Specific processors may differ in their internal characteristics, the installed facilities, the number of subchannels, channel paths, and control units which can be attached to the channel subsystem, the size of main and expanded storage, and the representation of the operator facilities.

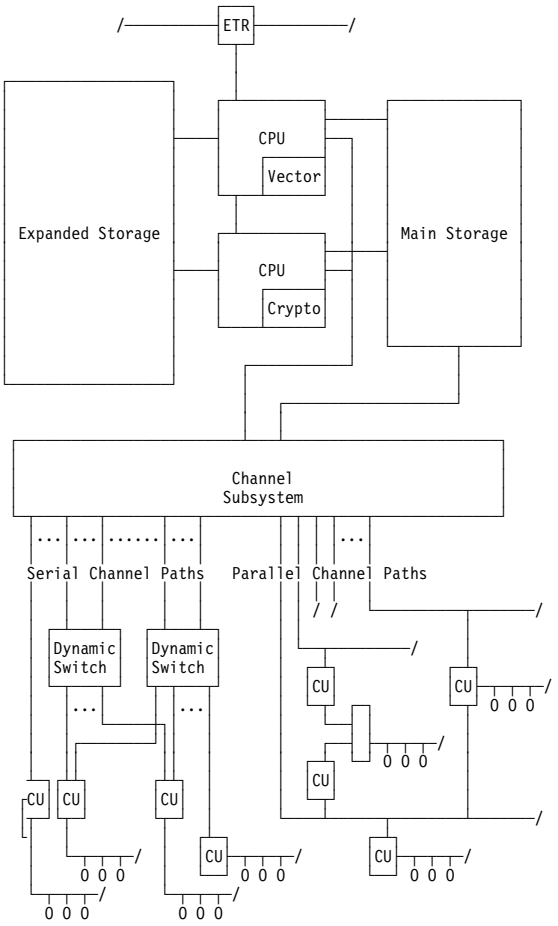



*Figure 2-1. Logical Structure of an ESA/390 System with Two CPUs*

A system viewed without regard to its I/O devices is referred to as a configuration. All of the physical equipment, whether in the configuration or not, is referred to as the installation.

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-49

Model-dependent reconfiguration controls may be provided to change the amount of main and expanded storage and the number of CPUs and channel paths in the configuration. In some instances, the reconfiguration controls may be used to partition a single configuration into multiple configurations. Each of the configurations so reconfigured has the same structure, that is, main and expanded storage, one or more CPUs, and one or more subchannels and channel paths in the channel subsystem.

Each configuration is isolated in that the main and expanded storage in one configuration is not directly addressable by the CPUs and the channel subsystem of another configuration. It is, however, possible for one configuration to communicate with another by means of shared I/O devices or a channel-to-channel adapter. At any one time, the storage, CPUs, subchannels, and channel paths connected together in a system are referred to as being in the configuration. Each CPU, subchannel, channel path, main-storage location, and expanded-storage location can be in only one configuration at a time.

## Main Storage

Main storage, which is directly addressable, provides for high-speed processing of data by the CPUs and the channel subsystem. Both data and programs must be loaded into main storage from input devices before they can be processed. The amount of main storage available in the system depends on the model, and, depending on the model, the amount in the configuration may be under control of model-dependent configuration controls. The storage is available in multiples of 4K-byte blocks. At any instant, the channel subsystem and all CPUs in the configuration have access to the same blocks of storage and refer to a particular block of main-storage locations by using the same absolute address.

Main storage may include a faster-access buffer storage, sometimes called a cache. Each CPU may have an associated cache. The effects, except on performance, of the physical construction and the use of distinct storage media are not observable by the program.

## Expanded Storage

Expanded storage may be available on some models. Expanded storage, when available, can be accessed by all CPUs in the configuration by means of instructions that transfer 4K-byte blocks of data from expanded storage to main storage or from main storage to expanded storage. These instructions are the PAGE IN and PAGE OUT instructions, described in Chapter 10, “Control Instructions.” Another capability for accessing expanded storage is described in the definition of the MOVE PAGE instruction in Chapter 7, “General Instructions,” and Chapter 10, “Control Instructions.”

Each 4K-byte block of expanded storage is addressed by means of a 32-bit unsigned binary integer called an expanded-storage block number.

Expanded storage is not further described.

## CPU

The central processing unit (CPU) is the controlling center of the system. It contains the sequencing and processing facilities for instruction execution, interruption action, timing functions, initial program loading, and other machine-related functions.

The physical implementation of the CPU may differ among models, but the logical function remains the same. The result of executing an instruction is the same for each model, providing that the program complies with the compatibility rules.

The CPU, in executing instructions, can process binary integers and floating-point numbers (binary and hexadecimal) of fixed length, decimal integers of variable length, and logical information of either fixed or variable length. Processing may be in parallel or in series; the width of the processing elements, the multiplicity of the shifting paths, and the degree of simultaneity in performing the different types of arithmetic differ from one model of CPU to another without affecting the logical results.

Instructions which the CPU executes fall into seven classes: general, decimal, floating-point-support (FPS), binary-floating-point (BFP), hexadecimal-floating-point (HFP), control, and I/O



LzLabs GmbH. Ex. 1008-50

instructions. The general instructions are used in performing binary-integer-arithmetic operations and logical, branching, and other nonarithmetic operations. The decimal instructions operate on data in the decimal format. The BFP and HFP instructions operate on data in the BFP and HFP formats, respectively, while the FPS instructions operate on floating-point data independent of the format or convert it from one format to the other. The privileged control instructions and the I/O instructions can be executed only when the CPU is in the supervisor state; the semiprivileged control instructions can be executed in the problem state, subject to the appropriate authorization mechanisms.

The CPU provides registers which are available to programs but do not have addressable representations in main storage. They include the current program-status word (PSW), the general registers, the floating-point registers and floating-point-control register, the control registers, the access registers, the prefix register, and the registers for the clock comparator and the CPU timer. Each CPU in an installation provides access to a time-of-day (TOD) clock, which may be local to that CPU or shared with other CPUs in the installation. The instruction operation code determines which type of register is to be used in an operation. See Figure 2-2 on page 2-5 for the format of the control, access, general, and floating-point registers.

## PSW

The program-status word (PSW) includes the instruction address, condition code, and other information used to control instruction sequencing and to determine the state of the CPU. The active or controlling PSW is called the current PSW. It governs the program currently being executed.

The CPU has an interruption capability, which permits the CPU to switch rapidly to another program in response to exceptional conditions and external stimuli. When an interruption occurs, the CPU places the current PSW in an assigned storage location, called the old-PSW location, for the particular class of interruption. The CPU fetches a new PSW from a second assigned storage location. This new PSW determines the next program to be executed. When it has finished processing the interruption, the program handling the interruption may reload the old PSW, making it again the current PSW, so that the interrupted program can continue.

There are six classes of interruption: external, I/O, machine check, program, restart, and supervisor call. Each class has a distinct pair of old-PSW and new-PSW locations permanently assigned in real storage.

## General Registers

Instructions may designate information in one or more of 16 general registers. The general registers may be used as base-address registers and index registers in address arithmetic and as accumulators in general arithmetic and logical operations. Each register contains 32 bit positions. The general registers are identified by the numbers 0-15 and are designated by a four-bit R field in an instruction. Some instructions provide for addressing multiple general registers by having several R fields. For some instructions, the use of a specific general register is implied rather than explicitly designated by an R field of the instruction.

For some operations, two adjacent general registers are coupled, providing a 64-bit format. In these operations, the program must designate an even-numbered register, which contains the leftmost (high-order) 32 bits. The next higher-numbered register contains the rightmost (low-order) 32 bits.

In addition to their use as accumulators in general arithmetic and logical operations, 15 of the 16 general registers are also used as base-address and index registers in address generation. In these cases, the registers are designated by a four-bit B field or X field in an instruction. A value of zero in the B or X field specifies that no base or index is to be applied, and, thus, general register 0 cannot be designated as containing a base address or index.

## Floating-Point Registers

All floating-point instructions (FPS, BFP, and HFP) use the same floating-point registers. When the basic-floating-point-extensions facility is installed, the CPU has 16 floating-point registers. The floating-point registers are identified by the numbers 0-15 and are designated by a four-bit R


LzLabs GmbH. Ex. 1008-51

field in floating-point instructions. Each floating-point register is 64 bits long and can contain either a short (32-bit) or a long (64-bit) floating-point operand. As shown in Figure 2-2 on page 2-5, pairs of floating-point registers can be used for extended (128-bit) operands. Each of the eight pairs is referred to by the number of the lower-numbered register of the pair. When the basic-floating-point-extensions facility is not installed, the CPU has four floating-point registers numbered 0, 2, 4, and 6.

## Floating-Point-Control (FPC) Register

The floating-point-control (FPC) register is a 32-bit register that contains mask bits, flag bits, a data-exception code, and rounding-mode bits. The FPC register is installed when the binary-floating-point facility is installed and is described in the section "Floating-Point-Control (FPC) Register" on page 19-2.

## Control Registers

The CPU has 16 control registers, each having 32 bit positions. The bit positions in the registers are assigned to particular facilities in the system, such as program-event recording, and are used either to specify that an operation can take place or to furnish special information required by the facility.

The control registers are identified by the numbers 0-15 and are designated by four-bit R fields in the instructions LOAD CONTROL and STORE CONTROL. Multiple control registers can be addressed by these instructions.

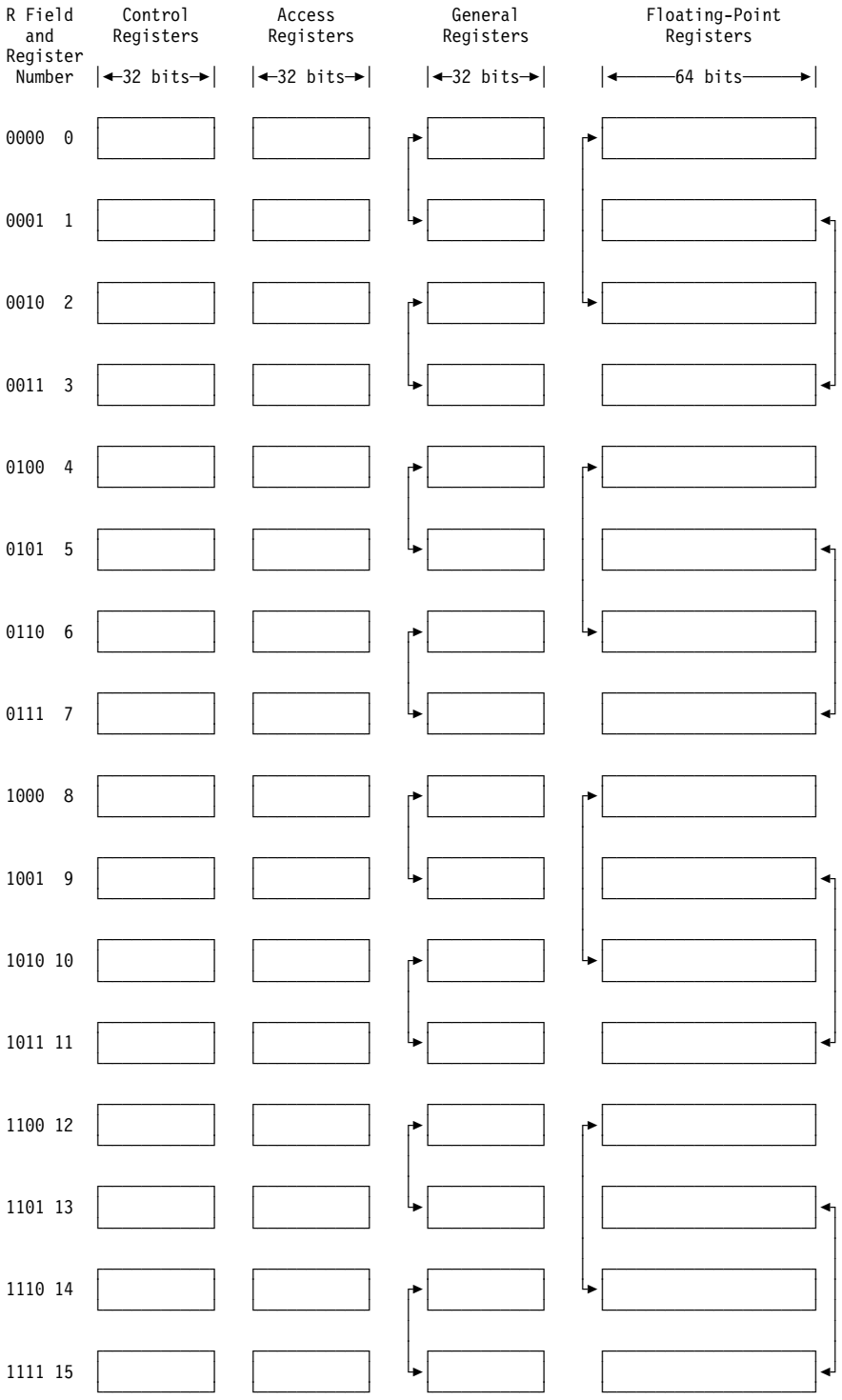
## Access Registers

The CPU has 16 access registers numbered 0-15. An access register consists of 32 bit positions containing an indirect specification (not described here in detail) of a segment-table designation. A segment-table designation is a parameter used by the dynamic-address-translation (DAT) mechanism to translate references to a corresponding address space. When the CPU is in a mode called the access-register mode (controlled by bits in the PSW), an instruction B field, used to specify a logical address for a storage-operand reference, designates an access register, and the segment-table designation specified by the access register is used by DAT for the reference being made. For some instructions, an R field is used instead of a B field. Instructions are provided for loading and storing the contents of the access registers and for moving the contents of one access register to another.

Each of access registers 1-15 can designate any address space, including the current instruction space (the primary address space). Access register 0 always designates the current instruction space. When one of access registers 1-15 is used to designate an address space, the CPU determines which address space is designated by translating the contents of the access register. When access register 0 is used to designate an address space, the CPU treats the access register as designating the current instruction space, and it does not examine the actual contents of the access register. Therefore, the 16 access registers can designate, at any one time, the current instruction space and a maximum of 15 other spaces.



LzLabs GmbH. Ex. 1008-52




*Note*: The arrows indicate that the two registers may be coupled as a double-register pair, designated by specifying the lower-numbered register in the R field. For example, the floating-point register pair 13 and 15 is designated by 1101 binary in the R field.

*Figure 2-2. Control, Access, General, and Floating-Point Registers*



LzLabs GmbH. Ex. 1008-53

## Vector Facility

Depending on the model, a vector facility may be provided as an extension of the CPU. When the vector facility is provided on a CPU, it functions as an integral part of that CPU. The functions of the vector facility and its registers are described in the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207.

## Cryptographic Facility

Depending on the model, an integrated cryptographic facility may be provided as an extension of the CPU. When the cryptographic facility is provided on a CPU, it functions as an integral part of that CPU. A summary of the benefits of the cryptographic facility is given in "Highlights of ESA/390" on page 1-1; the facility is otherwise not described.

## External Time Reference

Depending on the model, an external time reference (ETR) may be connected to the configuration. A summary of the benefits of the ETR is given in "Highlights of ESA/390" on page 1-1; the facility is otherwise not described.

## I/O

Input/output (I/O) operations involve the transfer of information between main storage and an I/O device. I/O devices and their control units attach to the channel subsystem, which controls this data transfer.

## Channel Subsystem

The channel subsystem directs the flow of information between I/O devices and main storage. It relieves CPUs of the task of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O processing. The channel subsystem uses one or more channel paths as the communication link in managing the flow of information to or from I/O devices. As part of I/O processing, the channel subsystem also performs the path-management function of testing for channel-path availability, selecting an available channel path, and initiating execution of the operation with the I/O device. Within the channel subsystem are subchannels.

One subchannel is provided for and dedicated to each I/O device accessible to the channel subsystem. Each subchannel contains storage for information concerning the associated I/O device and its attachment to the channel subsystem. The subchannel also provides storage for information concerning I/O operations and other functions involving the associated I/O device. Information contained in the subchannel can be accessed by CPUs using I/O instructions as well as by the channel subsystem and serves as the means of communication between any CPU and the channel subsystem concerning the associated I/O device. The actual number of subchannels provided depends on the model and the configuration; the maximum number of subchannels is 65,536.

## Channel Paths

I/O devices are attached through control units to the channel subsystem via channel paths. Control units may be attached to the channel subsystem via more than one channel path, and an I/O device may be attached to more than one control unit. In all, an individual I/O device may be accessible to a channel subsystem by as many as eight different channel paths, depending on the model and the configuration. The total number of channel paths provided by a channel subsystem depends on the model and the configuration; the maximum number of channel paths is 256.

A channel path can use one of three types of communication links:

- System/360 and System/370 I/O interface, called the parallel-I/O interface; the channel path is called a parallel channel path
- ESCON* I/O interface, called the serial-I/O interface; the channel path is called a serial channel path
- FICON* I/O interface; the channel path is again called a serial channel path

Each parallel-I/O interface consists of a number of electrical signal lines between the channel sub-

ESCON and FICON are trademarks of the International Business Machines Corporation.



LzLabs GmbH. Ex. 1008-54

system and one or more control units. Eight control units can share a single parallel-I/O interface. Up to 256 I/O devices can be addressed on a single parallel-I/O interface. The parallel-I/O interface is described in the publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information*, GA22-6974.

Each serial-I/O interface consists of two optical-fiber conductors between any two of a channel subsystem, a dynamic switch, and a control unit. A dynamic switch can be connected by means of multiple serial-I/O interfaces to either the same or different channel subsystems and to multiple control units. The number of control units which can be connected on one channel path depends on the channel-subsystem and dynamic-switch capabilities. Up to 256 devices can be attached to each control unit that uses the serial-I/O interface, depending on the control unit. The ESCON I/O interface is described in the publication *ESA/390 ESCON I/O Interface*, SA22-7202. The FICON I/O interface is described in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

## I/O Devices and Control Units

I/O devices include such equipment as printers, magnetic-tape units, direct-access-storage devices, displays, keyboards, communications controllers, teleprocessing devices, and sensor-based equipment. Many I/O devices function with an external medium, such as paper or magnetic tape. Other I/O devices handle only electrical signals, such as those found in displays and communications networks. In all cases, I/O-device operation is regulated by a control unit that provides the logical and buffering capabilities necessary to operate the associated I/O device. From the programming point of view, most control-unit functions merge with I/O-device functions. The control-unit function may be housed with the I/O device or in the CPU, or a separate control unit may be used.

## Operator Facilities

The operator facilities provide the functions necessary for operator control of the machine. Associated with the operator facilities may be an operator-console device, which may also be used as an I/O device for communicating with the program.

The main functions provided by the operator facilities include resetting, clearing, initial program loading, start, stop, alter, and display.


LzLabs GmbH. Ex. 1008-55

LzLabs GmbH. Ex. 1008-56

# Chapter 3. Storage


Storage Addressing . . . . . 3-2
Information Formats . . . . . 3-2
Integral Boundaries . . . . . 3-3
Address Types and Formats . . . . . 3-3
Address Types . . . . . 3-3
Absolute Address . . . . . 3-4
Real Address . . . . . 3-4
Virtual Address . . . . . 3-4
Primary Virtual Address . . . . . 3-4
Secondary Virtual Address . . . . . 3-4
AR-Specified Virtual Address . . . . . 3-4
Home Virtual Address . . . . . 3-4
Logical Address . . . . . 3-4
Instruction Address . . . . . 3-5
Effective Address . . . . . 3-5
Address Size and Wraparound . . . . . 3-5
Address Wraparound . . . . . 3-5
Storage Key . . . . . 3-8
Protection . . . . . 3-8
Key-Controlled Protection . . . . . 3-9
Storage-Protection-Override Control . . 3-10
Fetch-Protection-Override Control . . . 3-11
Access-List-Controlled Protection . . . . . 3-11
Page Protection . . . . . 3-11
Low-Address Protection . . . . . 3-12
Suppression on Protection . . . . . 3-12
Reference Recording . . . . . 3-14
Change Recording . . . . . 3-14
Prefixing . . . . . 3-15
Address Spaces . . . . . 3-16
Changing to Different Address Spaces . 3-17
Address-Space Number . . . . . 3-17
ASN Translation . . . . . 3-18
ASN-Translation Controls . . . . . 3-18
Control Register 14 . . . . . 3-18
Control Register 0 . . . . . 3-19
ASN-Translation Tables . . . . . 3-19
ASN-First-Table Entries . . . . . 3-19
ASN-Second-Table Entries . . . . . 3-19
ASN-Translation Process . . . . . 3-21
ASN-First-Table Lookup . . . . . 3-21
ASN-Second-Table Lookup . . . . . 3-22
Recognition of Exceptions during ASN Translation . . . . . 3-23
ASN Authorization . . . . . 3-23
ASN-Authorization Controls . . . . . 3-23
Control Register 4 . . . . . 3-24
ASN-Second-Table Entry . . . . . 3-24
Authority-Table Entries . . . . . 3-24
ASN-Authorization Process . . . . . 3-24
Authority-Table Lookup . . . . . 3-25
Recognition of Exceptions during ASN Authorization . . . . . 3-26
Dynamic Address Translation . . . . . 3-26
Translation Control . . . . . 3-27
Translation Modes . . . . . 3-28
Control Register 0 . . . . . 3-28
Control Register 1 . . . . . 3-28
Control Register 7 . . . . . 3-29
Control Register 13 . . . . . 3-29
Translation Tables . . . . . 3-30
Segment-Table Entries . . . . . 3-30
Page-Table Entries . . . . . 3-30
Summary of Segment-Table and Page-Table Sizes . . . . . 3-31
Translation Process . . . . . 3-31
Effective Segment-Table Designation . 3-32
Inspection of Control Register 0 . . . . 3-34
Segment-Table Lookup . . . . . 3-34
Page-Table Lookup . . . . . 3-35
Formation of the Real Address . . . . . 3-35
Recognition of Exceptions during Translation . . . . . 3-35
Translation-Lookaside Buffer . . . . . 3-35
TLB Structure . . . . . 3-36
Formation of TLB Entries . . . . . 3-36
Use of TLB Entries . . . . . 3-37
Modification of Translation Tables . . . 3-38
Address Summary . . . . . 3-40
Addresses Translated . . . . . 3-40
Handling of Addresses . . . . . 3-40
Assigned Storage Locations . . . . . 3-43


This chapter discusses the representation of information in main storage, as well as addressing, protection, and reference and change recording. The aspects of addressing which are covered include the format of addresses, the concept of address spaces, the various types of addresses, and the manner in which one type of address is translated to another type of address. A list of permanently assigned storage locations appears at the end of the chapter.

Main storage provides the system with directly addressable fast-access storage of data. Both data and programs must be loaded into main

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-57

storage (from input devices) before they can be processed.

Main storage may include one or more smaller faster-access buffer storages, sometimes called caches. A cache is usually physically associated with a CPU or an I/O processor. The effects, except on performance, of the physical construction and use of distinct storage media are not observable by the program.

Fetching and storing of data by a CPU are not affected by any concurrent channel-subsystem activity or by a concurrent reference to the same storage location by another CPU. When concurrent requests to a main-storage location occur, access normally is granted in a sequence determined by the system. If a reference changes the contents of the location, any subsequent storage fetches obtain the new contents.

Main storage may be volatile or nonvolatile. If it is volatile, the contents of main storage are not preserved when power is turned off. If it is nonvolatile, turning power off and then back on does not affect the contents of main storage, provided all CPUs are in the stopped state and no references are made to main storage when power is being turned off. In both types of main storage, the contents of storage keys are not necessarily preserved when the power for main storage is turned off.

**Note:** Because most references in this publication apply to virtual storage, the abbreviated term "storage" is often used in place of "virtual storage." The term "storage" may also be used in place of "main storage," "absolute storage," or "real storage" when the meaning is clear. The terms "main storage" and "absolute storage" are used to describe storage which is addressable by means of an absolute address. The terms describe fast-access storage, as opposed to auxiliary storage, such as that provided by direct-access storage devices. "Real storage" is synonymous with "absolute storage" except for the effects of prefixing.

## Storage Addressing

Storage is viewed as a long horizontal string of bits. For most operations, accesses to storage proceed in a left-to-right sequence. The string of bits is subdivided into units of eight bits. An eight-bit unit is called a byte, which is the basic building block of all information formats.

Each byte location in storage is identified by a unique nonnegative integer, which is the address of that byte location or, simply, the byte address. Adjacent byte locations have consecutive addresses, starting with 0 on the left and proceeding in a left-to-right sequence. Addresses are either 24-bit or 31-bit unsigned binary integers and are described in "Address Size and Wraparound" on page 3-5.

## Information Formats

Information is transmitted between storage and a CPU or the channel subsystem one byte, or a group of bytes, at a time. Unless otherwise specified, a group of bytes in storage is addressed by the leftmost byte of the group. The number of bytes in the group is either implied or explicitly specified by the operation to be performed. When used in a CPU operation, a group of bytes is called a field.

Within each group of bytes, bits are numbered in a left-to-right sequence. The leftmost bits are sometimes referred to as the "high-order" bits and the rightmost bits as the "low-order" bits. Bit numbers are not storage addresses, however. Only bytes can be addressed. To operate on individual bits of a byte in storage, it is necessary to access the entire byte.

The bits in a byte are numbered 0 through 7, from left to right.

The bits in an address are numbered 8 through 31 for 24-bit addresses and 1 through 31 for 31-bit addresses. Within any other fixed-length format of multiple bytes, the bits making up the format are consecutively numbered starting from 0.

For purposes of error detection, and in some models for correction, one or more check bits may be transmitted with each byte or with a group of bytes. Such check bits are generated automatically by the machine and cannot be directly con-



LzLabs GmbH. Ex. 1008-58

trolled by the program. References in this publication to the length of data fields and registers exclude mention of the associated check bits. All storage capacities are expressed in number of bytes.

When the length of a storage-operand field is implied by the operation code of an instruction, the field is said to have a fixed length, which can be one, two, four, or eight bytes. Larger fields may be implied for some instructions.

When the length of a storage-operand field is not implied but is stated explicitly, the field is said to have a variable length. Variable-length operands can vary in length by increments of one byte.

When information is placed in storage, the contents of only those byte locations are replaced that are included in the designated field, even though the width of the physical path to storage may be greater than the length of the field being stored.

## Integral Boundaries

Certain units of information must be on an integral boundary in storage. A boundary is called integral for a unit of information when its storage address is a multiple of the length of the unit in bytes. Special names are given to fields of two, four, and eight bytes on an integral boundary. A halfword is a group of two consecutive bytes on a two-byte boundary and is the basic building block of instructions. A word is a group of four consecutive bytes on a four-byte boundary. A doubleword is a group of eight consecutive bytes on an eight-byte boundary. (See Figure 3-1.)

When storage addresses designate halfwords, words, and doublewords, the binary representation of the address contains one, two, or three rightmost zero bits, respectively.

Instructions must be on two-byte integral boundaries, and CCWs, IDAWs, and the storage operands of certain instructions must be on other integral boundaries. The storage operands of most instructions do not have boundary-alignment requirements.

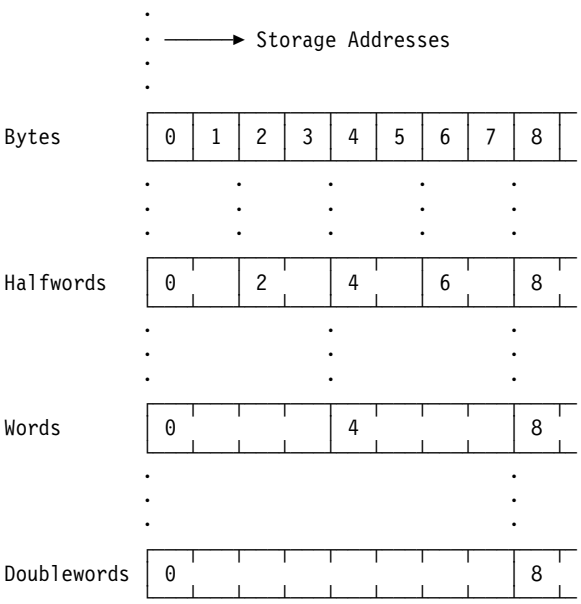



*Figure 3-1. Integral Boundaries with Storage Addresses*

**Programming Note:** For fixed-field-length operations with field lengths that are a power of 2, significant performance degradation is possible when storage operands are not positioned at addresses that are integral multiples of the operand length. To improve performance, frequently used storage operands should be aligned on integral boundaries.

# Address Types and Formats

## Address Types

For purposes of addressing main storage, three basic types of addresses are recognized: absolute, real, and virtual. The addresses are distinguished on the basis of the transformations that are applied to the address during a storage access. Address translation converts virtual to real, and prefixing converts real to absolute. In addition to the three basic address types, additional types are defined which are treated as one or another of the three basic types, depending on the instruction and the current mode.


LzLabs GmbH. Ex. 1008-59

## Absolute Address

An absolute address is the address assigned to a main-storage location. An absolute address is used for a storage access without any transformations performed on it.

The channel subsystem and all CPUs in the configuration refer to a shared main-storage location by using the same absolute address. Available main storage is usually assigned contiguous absolute addresses starting at 0, and the addresses are always assigned in complete 4K-byte blocks on integral boundaries. An exception is recognized when an attempt is made to use an absolute address in a block which has not been assigned to physical locations. On some models, storage-reconfiguration controls may be provided which permit the operator to change the correspondence between absolute addresses and physical locations. However, at any one time, a physical location is not associated with more than one absolute address.

Storage consisting of byte locations sequenced according to their absolute addresses is referred to as absolute storage.

## Real Address

A real address identifies a location in real storage. When a real address is used for an access to main storage, it is converted, by means of prefixing, to an absolute address.

At any instant there is one real-address to absolute-address mapping for each CPU in the configuration. When a real address is used by a CPU to access main storage, it is converted to an absolute address by prefixing. The particular transformation is defined by the value in the prefix register for the CPU.

Storage consisting of byte locations sequenced according to their real addresses is referred to as real storage.

## Virtual Address

A virtual address identifies a location in virtual storage. When a virtual address is used for an access to main storage, it is translated by means of dynamic address translation to a real address, which is then further converted by prefixing to an absolute address.

## Primary Virtual Address

A primary virtual address is a virtual address which is to be translated by means of the primary segment-table designation. Logical addresses are treated as primary virtual addresses when in the primary-space mode. Instruction addresses are treated as primary virtual addresses when in the primary-space mode, secondary-space mode, or access-register mode. The first-operand address of MOVE TO PRIMARY and the second-operand address of MOVE TO SECONDARY are always treated as primary virtual addresses.

## Secondary Virtual Address

A secondary virtual address is a virtual address which is to be translated by means of the secondary segment-table designation. Logical addresses are treated as secondary virtual addresses when in the secondary-space mode. The second-operand address of MOVE TO PRIMARY and the first-operand address of MOVE TO SECONDARY are always treated as secondary virtual addresses.

## AR-Specified Virtual Address

An AR-specified virtual address is a virtual address which is to be translated by means of an access-register-specified segment-table designation. Logical addresses are treated as AR-specified addresses when in the access-register mode.

## Home Virtual Address

A home virtual address is a virtual address which is to be translated by means of the home segment-table designation. Logical addresses and instruction addresses are treated as home virtual addresses when in the home-space mode.

## Logical Address

Except where otherwise specified, the storage-operand addresses for most instructions are logical addresses. Logical addresses are treated as real addresses in the real mode, as primary virtual addresses in the primary-space mode, as secondary virtual addresses in the secondary-space mode, as AR-specified virtual addresses in the access-register mode, and as home virtual addresses in the home-space mode. Some instructions have storage-operand addresses or storage accesses associated with the instruction which do not follow the rules for logical addresses.



LzLabs GmbH. Ex. 1008-60

In all such cases, the instruction definition contains a definition of the type of address.

### Instruction Address

Addresses used to fetch instructions from storage are called instruction addresses. Instruction addresses are treated as real addresses in the real mode, as primary virtual addresses in the primary-space mode, secondary-space mode, or access-register mode, and as home virtual addresses in the home-space mode. The instruction address in the current PSW and the target address of EXECUTE are instruction addresses.

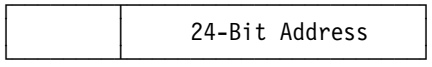
### Effective Address

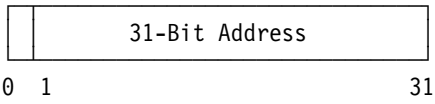
In some situations, it is convenient to use the term "effective address." An effective address is the address which exists before any transformation by dynamic address translation or prefixing is performed. An effective address may be specified directly in a register or may result from address arithmetic. Address arithmetic is the addition of the base and displacement or of the base, index, and displacement.

## Address Size and Wraparound

Two sizes of addresses are provided: 24-bit and 31-bit. A 24-bit address can accommodate a maximum of 16,777,216 (16M) bytes; with a 31-bit address, 2,147,483,648 (2G) bytes of storage can be addressed.

The bits of the address are numbered 8-31 and 1-31, respectively, corresponding to the numbering of base-address and index bits in a general register:




A 24-bit virtual address is expanded to 31 bits by appending seven zeros on the left before it is translated by means of the DAT process, and a 24-bit real address is similarly expanded to 31 bits before it is transformed by prefixing. A 24-bit absolute address is expanded to 31 bits before main storage is accessed. Thus, the 24-bit address always designates the first 16M-byte block of the 2G-byte storage addressable by a 31-bit address.

Unless specifically stated to the contrary, the following definition applies in this publication: whenever the machine generates and provides to the program an address, a 31-bit value imbedded in a 32-bit field is made available (placed in storage or loaded into a register). For 24-bit addresses, bits 0-7 are set to zeros, and the address appears in bit positions 8-31; for 31-bit addresses, bit 0 is set to zero, and the address appears in bit positions 1-31.

The size of effective addresses is controlled by bit 32 of the PSW, the addressing-mode bit. When the bit is zero, the CPU is in the 24-bit addressing mode, and 24-bit operand and instruction effective addresses are specified. When the bit is one, the CPU is in the 31-bit addressing mode, and 31-bit operand and instruction effective addresses are specified (see "Address Generation" on page 5-7).

The size of the real addresses yielded by the ASN-translation, PC-number-translation, ASN-authorization, access-register translation, and tracing processes, and the real (or absolute) addresses yielded by the DAT process, is always 31 bits.

The size of the data address in a CCW is under control of the format-control bit in the operation-request block designated by a START SUBCHANNEL instruction. The CCWs with 24-bit and 31-bit addresses are called format-0 and format-1 CCWs, respectively. Format-0 and format-1 CCWs are described in Chapter 15, "Basic I/O Functions."

### Address Wraparound

The CPU performs address generation when it forms an operand or instruction address or when it generates the address of a table entry from the appropriate table origin and index. It also performs address generation when it increments an address to access successive bytes of a field. Similarly, the channel subsystem performs address generation when it increments an address (1) to fetch a CCW, (2) to fetch an IDAW, (3) to transfer data, or (4) to compute the address of an I/O measurement block.

When, during the generation of the address, an address is obtained that exceeds the value


LzLabs GmbH. Ex. 1008-61

allowed for the address size ($2^{24}$ - 1 or $2^{31}$ - 1), one of the following two actions is taken:

1. The carry out of the high-order bit position of the address is ignored. This handling of an address of excessive size is called *wraparound*.
2. An interruption condition is recognized.

The effect of wraparound is to make an address space appear circular; that is, address 0 appears to follow the maximum allowable address. Address arithmetic and wraparound occur before transformation, if any, of the address by DAT or prefixing.

Addresses generated by the CPU that may be virtual addresses always wrap. Wraparound also occurs when the linkage-stack-entry address in control register 15 is decremented below 0 by PROGRAM RETURN. For CPU table entries that are addressed by real or absolute addresses, it is unpredictable whether the address wraps or an addressing exception is recognized.

For channel-program execution, when the generated address exceeds the value for the address size (or, for the read-backward command is decremented below 0), an I/O program-check condition is recognized.

Figure 3-2 on page 3-7 identifies what limit values apply to the generation of different addresses and how addresses are handled when they exceed the allowed value.



LzLabs GmbH. Ex. 1008-62

| Address Generation for | Address Type | Handling when Address Would Wrap |
|---|---|---|
| Instructions and operands when AM is zero | L,I,R,V | W24 |
| Successive bytes of instructions and operands when AM is zero | I,L,V[1] | W24 |
| Instructions and operands when AM is one | L,I,R,V | W31 |
| Successive bytes of instructions and operands when AM is one | I,L,V[1] | W31 |
| DAT-table entries when used for implicit translation or LRA | A or R[2] | X31 |
| ASN-second-table, authority-table (during ASN authorization), linkage-table, entry-table, and PCF-entry-table entries | R | X31 |
| Authority-table (during access-register translation) and access-list entries | A or R[2] | X31 |
| Linkage-stack entry | V | W31 |
| I/O measurement block | A | P31 |
| For a channel program with format-0 CCWs: | | |
| Successive CCWs | A | P24 |
| Successive IDAWs | A | P24 |
| Successive bytes of I/O data (without IDAWs) | A | P24 |
| Successive bytes of I/O data (with IDAWs) | A | P31 |
| For a channel program with format-1 CCWs: | | |
| Successive CCWs | A | P31 |
| Successive IDAWs | A | P31 |
| Successive bytes of I/O data (without IDAWs) | A | P31 |
| Successive bytes of I/O data (with IDAWs) | A | P31 |

*Figure 3-2 (Part 1 of 2). Address Wraparound*


LzLabs GmbH. Ex. 1008-63

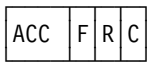
**Explanation:**

1 Real addresses do not apply in this case since the instructions which designate operands by means of real addresses cannot designate operands that cross boundaries $2^{24}$ and $2^{31}$.
2 It is unpredictable whether the address is absolute or real.
A Absolute address.
AM Addressing-mode bit in the PSW.
I Instruction address.
L Logical address.
P24 An I/O program-check condition is recognized when the address exceeds $2^{24} - 1$ or is decremented below zero.
P31 An I/O program-check condition is recognized when the address exceeds $2^{31} - 1$ or is decremented below zero.
R Real address.
V Virtual address.
W24 Wrap to location 0 after location $2^{24} - 1$ and vice versa.
W31 Wrap to location 0 after location $2^{31} - 1$ and vice versa.
X31 When the address exceeds $2^{31} - 1$, it is unpredictable whether the address wraps to location 0 after location $2^{31} - 1$ or whether an addressing exception is recognized.

*Figure 3-2 (Part 2 of 2). Address Wraparound*

## Storage Key

A storage key is associated with each 4K-byte block of storage that is available in the configuration. The storage key has the following format:




The bit positions in the storage key are allocated as follows:

***Access-Control Bits (ACC):*** If a reference is subject to key-controlled protection, the four access-control bits, bits 0-3, are matched with the four-bit access key when information is stored and when information is fetched from a location that is protected against fetching.

***Fetch-Protection Bit (F):*** If a reference is subject to key-controlled protection, the fetch-protection bit, bit 4, controls whether key-controlled protection applies to fetch-type references: a zero indicates that only store-type references are monitored and that fetching with any access key is permitted; a one indicates that key-controlled protection applies to both fetching and storing. No distinction is made between the fetching of instructions and of operands.

***Reference Bit (R):*** The reference bit, bit 5, normally is set to one each time a location in the corresponding storage block is referred to either for storing or for fetching of information.

***Change Bit (C):*** The change bit, bit 6, is set to one each time information is stored at a location in the corresponding storage block.

Storage keys are not part of addressable storage. The entire storage key is set by SET STORAGE KEY EXTENDED and inspected by INSERT STORAGE KEY EXTENDED. Additionally, the instruction RESET REFERENCE BIT EXTENDED provides a means of inspecting the reference and change bits and of setting the reference bit to zero. Bits 0-4 of the storage key are inspected by the INSERT VIRTUAL STORAGE KEY instruction. The contents of the storage key are unpredictable during and after the execution of the usability test of the TEST BLOCK instruction.

## Protection

Four protection facilities are provided to protect the contents of main storage from destruction or misuse by programs that contain errors or are unauthorized: key-controlled protection, access-list-controlled protection, page protection, and low-address protection. The protection facilities are applied independently; access to main storage is



LzLabs GmbH. Ex. 1008-64

only permitted when none of the facilities prohibits the access.

Key-controlled protection affords protection against improper storing or against both improper storing and fetching, but not against improper fetching alone.

## Key-Controlled Protection

When key-controlled protection applies to a storage access, a store is permitted only when the storage key matches the access key associated with the request for storage access; a fetch is permitted when the keys match or when the fetch-protection bit of the storage key is zero.

The keys are said to match when the four access-control bits of the storage key are equal to the access key, or when the access key is zero.

The protection action is summarized in Figure 3-3.

| Conditions | | Is Access to Storage Permitted? | |
|---|---|---|---|
| Fetch-Protection Bit of Storage Key | Key Relation | Fetch | Store |
| 0 | Match | Yes | Yes |
| 0 | Mismatch | Yes | No |
| 1 | Match | Yes | Yes |
| 1 | Mismatch | No | No |

**Explanation:**

Match — The four access-control bits of the storage key are equal to the access key, or the access key is zero.

Yes — Access is permitted.

No — Access is not permitted. On fetching, the information is not made available to the program; on storing, the contents of the storage location are not changed.

*Figure 3-3. Summary of Protection Action*

When the access to storage is initiated by the CPU and key-controlled protection applies, the PSW key is the access key, except that the access key is specified in a general register for the first operand of MOVE TO SECONDARY and MOVE WITH DESTINATION KEY, for the second operand of MOVE TO PRIMARY, MOVE WITH KEY, and MOVE WITH SOURCE KEY, and for either the first or the second operand of MOVE PAGE. The PSW key occupies bit positions 8-11 of the current PSW.

When the access to storage is for the purpose of channel-program execution, the subchannel key associated with that channel program is the access key. The subchannel key for a channel program is specified in the operation-request block (ORB). When, for purposes of channel-subsystem monitoring, an access to the measurement block is made, the measurement-block key is the access key. The measurement-block key is specified by the SET CHANNEL MONITOR instruction.

When a CPU access is prohibited because of key-controlled protection, the execution of the instruction is terminated, and a program interruption for a protection exception takes place. However, if the suppression-on-protection facility is installed, the execution of the instruction may be suppressed. When a channel-program access is prohibited, the start function is ended, and the protection-check condition is indicated in the associated interruption-response block (IRB). When a measurement-block access is prohibited, the I/O measurement-block protection-check condition is indicated.

When a store access is prohibited because of key-controlled protection, the contents of the protected location remain unchanged. When a fetch access is prohibited, the protected information is not loaded into a register, moved to another storage location, or provided to an I/O device. For a prohibited instruction fetch, the instruction is suppressed, and an arbitrary instruction-length code is indicated.

Key-controlled protection is independent of whether the CPU is in the problem or the supervisor state and, except as described below, does not depend on the type of CPU instruction or channel-command word being executed.

Except where otherwise specified, all accesses to storage locations that are explicitly designated by the program and that are used by the CPU to store or fetch information are subject to key-controlled protection.

Key-controlled protection does not apply when the storage-protection-override control is one and the value of the four access-control bits of the storage key is 9. Key-controlled protection for fetches


LzLabs GmbH. Ex. 1008-65

may or may not apply when the fetch-protection-override control is one, depending on the effective address and the private-space control.

The storage-protection-override control and fetch-protection-override control do not affect storage references made by the channel subsystem.

Accesses to the second operand of TEST BLOCK are not subject to key-controlled protection.

Key-controlled protection is not applied to accesses by the instructions PAGE IN and PAGE OUT.

All storage accesses by the channel subsystem to access the I/O measurement block, or by a channel program to fetch a CCW or IDAW or to access a data area designated during the execution of a CCW, are subject to key-controlled protection. However, if a CCW, an IDAW, or output data is prefetched, a protection check is not indicated until the CCW or IDAW is due to take control or until the data is due to be written.

Key-controlled protection is not applied to accesses that are implicitly made for any of such sequences as:

- An interruption
- CPU logout
- Fetching of table entries for access-register translation, dynamic-address translation, PC-number translation, ASN translation, or ASN authorization
- Tracing
- A store-status function
- Storing in real locations 184-191 when TEST PENDING INTERRUPTION has an operand address of zero
- Initial program loading

Similarly, protection does not apply to accesses initiated via the operator facilities for altering or displaying information. However, when the program explicitly designates these locations, they are subject to protection.

## Storage-Protection-Override Control

Bit 7 of control register 0 is the storage-protection-override control. When the storage-protection-override facility is installed and this bit is one, storage-protection override is active. When the storage-protection-override facility is not installed or this bit is zero, storage-protection override is inactive. When storage-protection override is active, key-controlled storage protection is ignored for storage locations having an associated storage-key value of 9. When storage-protection override is inactive, no special action is taken for a storage-key value of 9.

Storage-protection override applies to instruction fetch and to the fetch and store accesses of instructions whose operand addresses are logical, virtual, or real. It does not apply to accesses made for the purpose of channel-program execution or for the purpose of channel-subsystem monitoring.

Storage-protection override applies to the operands of MOVE PAGE even when the operand is in expanded storage.

Storage-protection override has no effect on accesses which are not subject to key-controlled protection.

**Programming Notes:**

1. The storage-protection-override facility can be used to improve reliability in the case when a possibly erroneous application program is executed in conjunction with a reliable subsystem, provided that the application program needs to access only a portion of the storage accessed by the subsystem. The technique for doing this is as follows. The storage accessed by the application program is given storage key 9. The storage accessed by only the subsystem is given some other nonzero storage key, for example, key 8. The application is executed with PSW key 9. The subsystem is executed with PSW key 8 (in this example). As a result, the subsystem can access both the key-8 and the key-9 storage, while the application program can access only the key-9 storage.
2. Storage-protection override affects the accesses to storage made by the CPU and also affects the result set by TEST PROTECTION. However, those instructions which,



LzLabs GmbH. Ex. 1008-66

in the problem state, test the PSW-key mask to determine if a particular key value may be used are not affected by whether storage-protection override is active. These instructions include, among others, MOVE WITH KEY and SET PSW KEY FROM ADDRESS. To permit these instructions to use an access key of 9 in the problem state, bit 9 of the PSW-key mask must be one.

### Fetch-Protection-Override Control

Bit 6 of control register 0 is the fetch-protection-override control. When the bit is one, fetch protection is ignored for locations at effective addresses 0-2047. An effective address is the address which exists before any transformation by dynamic address translation or prefixing. However, fetch protection is not ignored if the effective address is subject to dynamic address translation and the private-space control, bit 23, is one in the segment-table designation used in the translation.

Fetch-protection override applies to instruction fetch and to the fetch accesses of instructions whose operand addresses are logical, virtual, or real. It does not apply to fetch accesses made for the purpose of channel-program execution or for the purpose of channel-subsystem monitoring. When this bit is set to zero, fetch protection of locations at effective addresses 0-2047 is determined by the state of the fetch-protection bit of the storage key associated with those locations.

Fetch-protection override has no effect on accesses which are not subject to key-controlled protection.

**Programming Note:** The fetch-protection-override control allows fetch protection of locations at addresses 2048-4095 along with no fetch protection of locations at addresses 0-2047.

## Access-List-Controlled Protection

In the access-register mode, bit 6 of the access-list entry, the fetch-only bit, controls which types of operand references are permitted to the address space specified by the access-list entry. When the entry is used in the access-register-translation part of a reference and bit 6 is zero, both fetch-type and store-type references are permitted; when bit 6 is one, only fetch-type references are permitted, and an attempt to store causes a protection exception to be recognized and the execution of the instruction to be suppressed.

The fetch-only bit is included in the ALB access-list entry. A change to the fetch-only bit in an access-list entry in main storage does not necessarily have an immediate, if any, effect on whether a protection exception is recognized. However, this change to the bit will have an effect immediately after PURGE ALB or a COMPARE AND SWAP AND PURGE instruction that purges the ALB is executed.

TEST PROTECTION takes into consideration access-list-controlled protection when the CPU is in the access-register mode. A violation of access-list-controlled protection causes condition code 1 to be set, except that it does not prevent condition code 2 or 3 from being set when the conditions for those codes are satisfied.

Access-list-controlled protection does not affect LOAD REAL ADDRESS.

**Programming Note:** A violation of access-list-controlled protection always causes suppression. A violation of any of the other protection types may cause termination.

## Page Protection

The page-protection facility controls access to virtual storage by using the page-protection bit in each page-table entry. It provides protection against improper storing.

The page-protection bit, bit 22 of the page-table entry, controls whether storing is allowed into the corresponding 4K-byte page. When the bit is zero, both fetching and storing are permitted; when the bit is one, only fetching is permitted. When an attempt is made to store into a protected page, the contents of the page remain unchanged, the execution of the instruction is terminated, and a program interruption for protection takes place. However, if the suppression-on-protection facility is installed, the execution of the instruction is suppressed.

Page protection applies to all store-type references that use a virtual address.


LzLabs GmbH. Ex. 1008-67

## Low-Address Protection

The low-address-protection facility provides protection against the destruction of main-storage information used by the CPU during interruption processing. This is accomplished by prohibiting instructions from storing with effective addresses in the range 0 through 511. The range criterion is applied before address transformation, if any, of the address by dynamic address translation or prefixing. However, the range criterion is not applied, with the result that low-address protection does not apply, if the effective address is subject to dynamic address translation and the private-space control, bit 23, is one in the segment-table designation used in the translation. Low-address protection does not apply if the segment-table designation to be used is not available due to another type of exception.

Low-address protection is under control of bit 3 of control register 0, the low-address-protection-control bit. When the bit is zero, low-address protection is off; when the bit is one, low-address protection is on.

If an access is prohibited because of low-address protection, the contents of the protected location remain unchanged, the execution of the instruction is terminated, and a program interruption for a protection exception takes place. However, if the suppression-on-protection facility is installed, the execution of the instruction may be suppressed.

Any attempt by the program to store by using effective addresses in the range 0 through 511 is subject to low-address protection. Low-address protection is applied to the store accesses of instructions whose operand addresses are logical, virtual, or real. Low-address protection is also applied to the trace table.

Low-address protection is not applied to accesses made by the CPU or the channel subsystem for such sequences as interruptions, CPU logout, the storing of the I/O-interruption code in real locations 184-191 by TEST PENDING INTERRUPTION, and the initial-program-loading and store-status functions, nor is it applied to data stores during I/O data transfer. However, explicit stores by a program at any of these locations are subject to low-address protection.

**Programming Notes:**

1. Low-address protection and key-controlled protection apply to the same store accesses, except that:
   a. Low-address protection does not apply to storing performed by the channel subsystem, whereas key-controlled protection does.
   b. Key-controlled protection does not apply to tracing, the second operand of TEST BLOCK, or instructions that operate specifically on the linkage stack, whereas low-address protection does.
2. Because fetch-protection override and low-address protection do not apply to an address space for which the private-space control is one in the segment-table designation, locations 0-2047 in the address space are usable the same as the other locations in the space.

## Suppression on Protection

If the suppression-on-protection facility is installed, then, during a program interruption due to a protection exception, either a one or a zero is stored in bit position 29 of real locations 144-147. The storing of a one in bit position 29 indicates that:

- The unit of operation or instruction execution during which the exception was recognized was suppressed, except that, if the instruction execution would set the condition code if completed normally, the condition code is unpredictable. However, for ADD LOGICAL WITH CARRY and SUBTRACT LOGICAL WITH BORROW, suppression leaves the condition code unchanged.
- Bit positions 1-19 of real locations 144-147 contain bits 1-19 of the effective address that caused the exception. The effective address is the address which exists before any transformation by dynamic address translation (DAT) or prefixing. If the effective address was to be translated by DAT, bit positions 30 and 31 of real locations 144-147, and real location 160, contain the same information as is stored during a program interruption due to a page-translation exception — this information identifies the address space containing the protected address. If the effective address was not to be translated by DAT, the contents



LzLabs GmbH. Ex. 1008-68

of bit positions 30 and 31 of real locations 144-147, and the contents of real location 160, are unpredictable. The contents of bit positions 0 and 20-28 of real locations 144-147 are always unpredictable.

- If an additional facility named the virtual-address enhancement of suppression on protection is installed[1], and if DAT was on, as indicated by the DAT-mode bit in the program old PSW, the effective address in real locations 144-147 is always one that was to be translated by DAT. (Bit 29 is set to zero if DAT was on but the effective address was not to be translated by DAT because it is a real address.)

Bit 29 being zero indicates that the operation was either suppressed or terminated and that the contents of the remainder of real locations 144-147, and of real location 160, are unpredictable.

Bit 29 is set to one if the protection exception was due to access-list-controlled protection or page protection. Bit 29 may be set to one if the protection exception was due to low-address protection or key-controlled protection.

If a protection-exception condition exists due to either access-list-controlled protection or page protection but also exists due to either low-address protection or key-controlled protection, it is unpredictable whether bit 29 is set to zero or one.

**Programming Notes:**

1. The suppression-on-protection facility is useful in performing the AIX/ESA copy-on-write function, in which AIX/ESA causes the same page of different address spaces to map to a single page frame of real storage so long as a store in the page is not attempted and then, when a store is attempted in a particular address space, assigns a unique page frame to the page in that address space and copies the contents of the page to the new page frame.
2. In the problem state, the effective address that caused a protection exception is known to have required translation by DAT if DAT was on, as indicated by the DAT-mode bit in the program old PSW. In the supervisor state when the virtual-address enhancement of suppression on protection is not installed, the DAT-mode bit is not a reliable indicator of whether DAT was required since the effective address may be a real address used by, for example, STORE USING REAL ADDRESS. When the virtual-address enhancement is installed, the effective address stored at real locations 144-147 is known to be a virtual address if DAT was on. The knowledge that the address is virtual allows programmed forms of access-register translation and dynamic address translation to be performed to determine whether the exception was due to either access-list-controlled or page protection as opposed to low-address or key-controlled protection.
3. AIX/ESA does not use key-controlled protection. The virtual-address enhancement extends the usefulness of suppression on protection to other operating systems that do use key-controlled protection.
4. The results of suppression on protection are summarized in Figure 3-4 on page 3-14.

[1] The virtual-address enhancement is always installed along with the suppression-on-protection facility except that, on 9121 models, the virtual-address enhancement is installed only if SEC C35954 is installed.


LzLabs GmbH. Ex. 1008-69

| LA or Key-Cont. Prot. | DAT | ALC or Page Prot. | Eff. Addr. | Virt. Addr. Enhmt. Instl. | Bit 29 | Bits 30,31 and Loc. 160 if Bit 29 One |
|---|---|---|---|---|---|---|
| No | On | Yes | Log. | - | 1 | P |
| Yes | On | Yes | Log. | - | U1 | P |
| Yes | Off | No | Log. | - | U2 | U3 |
| Yes | Off | No | Real | - | U2 | U3 |
| Yes | On | No | Log. | - | U2 | P |
| Yes | On | No | Real | No | U2 | U3 |
| Yes | On | No | Real | Yes | 0A | - |

**Explanation:**

- `-` Immaterial or not applicable.
- `ALC` Access-list-controlled.
- `LA` Low-address.
- `Log.` Logical.
- `P` Predictable.
- `U1` Unpredictable because low-address or key-controlled protection may be recognized instead of access-list-controlled or page protection.
- `U2` Unpredictable because bit 29 is only required to be set to one for access-list-controlled or page protection.
- `U3` Unpredictable because effective address is not to be translated by DAT.
- `0A` Zero because DAT is on and a virtual effective address would not be stored.

*Figure 3-4. Suppression-on-Protection Results*

## Reference Recording

Reference recording provides information for use in selecting pages for replacement. Reference recording uses the reference bit, bit 5 of the storage key. The reference bit is set to one each time a location in the corresponding storage block is referred to either for fetching or for storing information, regardless of whether DAT is on or off.

Reference recording is always active and takes place for all storage accesses, including those made by any CPU, any operator facility, or the channel subsystem. It takes place for implicit accesses made by the machine, such as those which are part of interruptions and I/O-instruction execution.

Reference recording does not occur for operand accesses of the following instructions since they directly refer to a storage key without accessing a storage location:

- INSERT STORAGE KEY EXTENDED
- RESET REFERENCE BIT EXTENDED (reference bit is set to zero)
- SET STORAGE KEY EXTENDED (reference bit is set to a specified value)

The record provided by the reference bit is substantially accurate. The reference bit may be set to one by fetching data or instructions that are neither designated nor used by the program, and, under certain conditions, a reference may be made without the reference bit being set to one. Under certain unusual circumstances, a reference bit may be set to zero by other than explicit program action.

## Change Recording

Change recording provides information as to which pages have to be saved in auxiliary storage when they are replaced in main storage. Change recording uses the change bit, bit 6 of the storage key.

The change bit is set to one each time a store access causes the contents of the corresponding storage block to be changed. A store access that does not change the contents of storage may or may not set the change bit to one.

The change bit is not set to one for an attempt to store if the access is prohibited. In particular:

1. For the CPU, a store access is prohibited whenever an access exception exists for that access, or whenever an exception exists which is of higher priority than the priority of an access exception for that access.
2. For the channel subsystem, a store access is prohibited whenever a key-controlled-protection violation exists for that access.

Change recording is always active and takes place for all store accesses to storage, including those made by any CPU, any operator facility, or the channel subsystem. It takes place for implicit references made by the machine, such as those which are part of interruptions.

It is unpredictable whether change recording takes place for the instruction PAGE IN.

Change recording does not take place for the operands of the following instructions since they



LzLabs GmbH. Ex. 1008-70

directly modify a storage key without modifying a storage location:

- RESET REFERENCE BIT EXTENDED
- SET STORAGE KEY EXTENDED (change bit is set to a specified value)

Change bits which have been changed from zeros to ones are not necessarily restored to zeros on CPU retry (see “CPU Retry” on page 11-2). See “Exceptions to Nullification and Suppression” on page 5-19 for a description of the handling of the change bit in certain unusual situations.

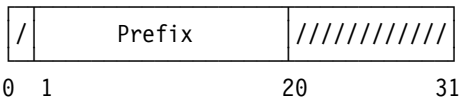
## Prefixing

Prefixing provides the ability to assign the range of real addresses 0-4095 to a different block in absolute storage for each CPU, thus permitting more than one CPU sharing main storage to operate concurrently with a minimum of interference, especially in the processing of interruptions.

Prefixing causes real addresses in the range 0-4095 to correspond to the block of 4K-byte absolute addresses (the prefix area) identified by the value in the prefix register for the CPU, and the block of real addresses identified by the value in the prefix register to correspond to absolute addresses 0-4095. The remaining real addresses are the same as the corresponding absolute addresses. This transformation allows each CPU to access all of main storage, including the first 4K bytes and the locations designated by the prefix registers of other CPUs.

The relationship between real and absolute addresses is graphically depicted in Figure 3-5 on page 3-16.

The prefix is a 19-bit quantity contained in bit positions 1-19 of the prefix register. The register has the following format:




The contents of the register can be set and inspected by the privileged instructions SET PREFIX and STORE PREFIX, respectively. On setting, bits corresponding to bit positions 0 and 20-31 of the prefix register are ignored. On storing, zeros are provided for these bit positions. When the contents of the prefix register are changed, the change is effective for the next sequential instruction.

When prefixing is applied, the real address is transformed into an absolute address by using one of the following rules, depending on bits 1-19 of the real address:

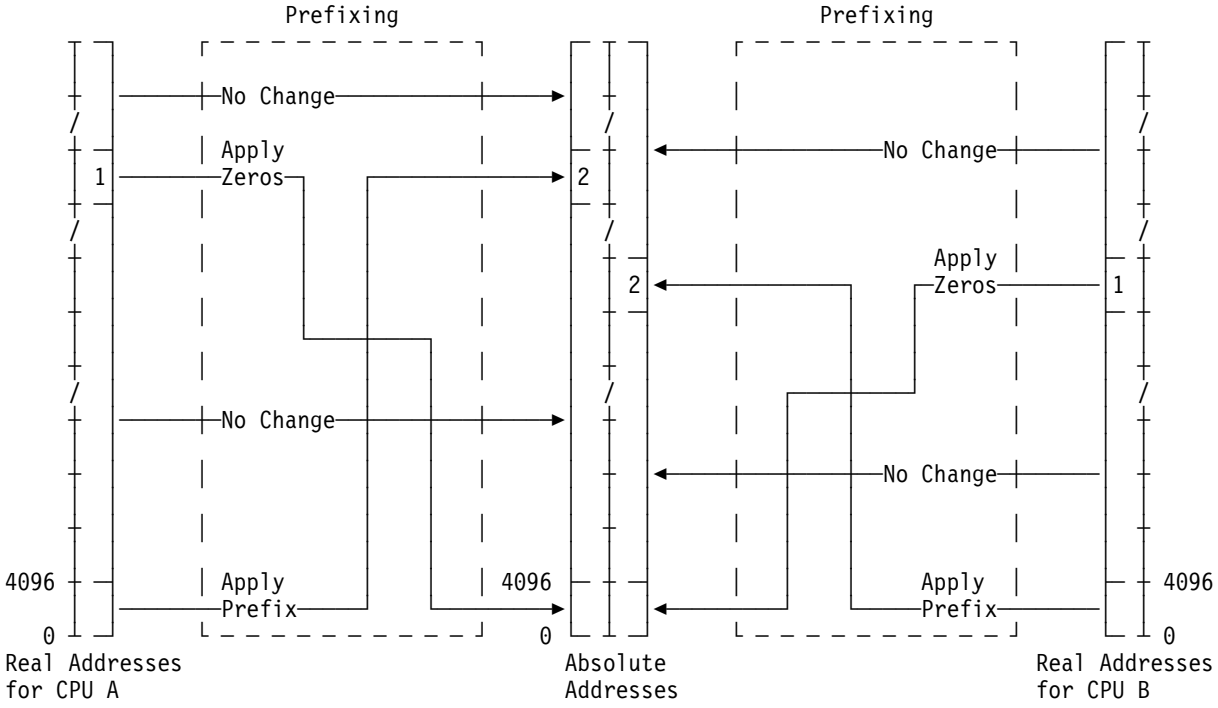
1. Bits 1-19 of the address, if all zeros, are replaced with bits 1-19 of the prefix.
2. Bits 1-19 of the address, if equal to bits 1-19 of the prefix, are replaced with zeros.
3. Bits 1-19 of the address, if not all zeros and not equal to bits 1-19 of the prefix, remain unchanged.

In all cases, bits 20-31 of the address remain unchanged.

Only the address presented to storage is translated by prefixing. The contents of the source of the address remain unchanged.

The distinction between real and absolute addresses is made even when the prefix register contains all zeros, in which case a real address and its corresponding absolute address are identical.


LzLabs GmbH. Ex. 1008-71




(1) Real addresses in which bits 1-19 are equal to the prefix for this CPU (A or B).

(2) Absolute addresses of the block that contains for this CPU (A or B) the real locations 0-4095.

*Figure 3-5. Relationship between Real and Absolute Addresses*

## Address Spaces

An address space is a consecutive sequence of integer numbers (virtual addresses), together with the specific transformation parameters which allow each number to be associated with a byte location in storage. The sequence starts at zero and proceeds left to right.

When a virtual address is used by a CPU to access main storage, it is first converted, by means of dynamic address translation (DAT), to a real address, and then, by means of prefixing, to an absolute address. DAT uses two levels of tables (segment tables and page tables) as transformation parameters. The designation (origin and length) of a segment table is found for use by DAT in a control register or as specified by an access register.

DAT uses, at different times, the segment-table designations in different control registers or specified by the access registers. The choice is determined by the translation mode specified in the current PSW. Four translation modes are available: primary-space mode, secondary-space mode, access-register mode, and home-space mode. Different address spaces are addressable depending on the translation mode.

At any instant when the CPU is in the primary-space mode or secondary-space mode, the CPU can translate virtual addresses belonging to two address spaces — the primary address space and the secondary address space. At any instant when the CPU is in the access-register mode, it can translate virtual addresses of up to 16 address spaces — the primary address space and up to 15 AR-specified address spaces. At any instant when the CPU is in the home-space mode, it can translate virtual addresses of the home address space.

The primary address space is identified as such because it consists of primary virtual addresses, which are translated by means of the primary segment-table designation. Similarly, the secondary address space consists of secondary virtual addresses translated by means of the secondary segment-table designation, the



LzLabs GmbH. Ex. 1008-72

AR-specified address spaces consist of AR-specified virtual addresses translated by means of AR-specified segment-table designations, and the home address space consists of home virtual addresses translated by means of the home segment-table designation. The primary and secondary segment-table designations are in control registers 1 and 7, respectively. The AR-specified segment-table designations are in control registers 1 and 7 and in table entries called ASN-second-table entries. The home segment-table designation is in control register 13.

## Changing to Different Address Spaces

A program can cause different address spaces to be addressable by using the semiprivileged SET ADDRESS SPACE CONTROL or SET ADDRESS SPACE CONTROL FAST instruction to change the translation mode to the primary-space mode, secondary-space mode, access-register mode, or home-space mode. However, SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST can set the home-space mode only in the supervisor state. The program can cause still other address spaces to be addressable by using other semiprivileged instructions to change the segment-table designations in control registers 1 and 7 and by using unprivileged instructions to change the contents of the access registers. Only the privileged LOAD CONTROL instruction is available for changing the home segment-table designation in control register 13.

## Address-Space Number

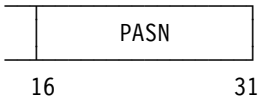
An address space may be assigned an address-space number (ASN) by the control program. The ASN designates, within a two-level table structure in main storage, an ASN-second-table entry containing information about the address space. If the ASN-second-table entry is marked as valid, it contains the segment-table designation that defines the address space.

Under certain circumstances, the semiprivileged instructions which place a new segment-table designation in control register 1 or 7 fetch this segment-table designation from an ASN-second-table entry. Some of these instructions use an ASN-translation mechanism which, given an ASN, can locate the designated ASN-second-table entry.

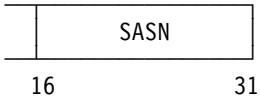
The 16-bit unsigned binary format of the ASN permits 64K unique ASNs.

The ASNs for the primary and secondary address spaces are assigned positions in control registers. The ASN for the primary address space, called the primary ASN, is assigned bits 16-31 of control register 4, and that for the secondary address space, called the secondary ASN, is assigned bits 16-31 of control register 3. The registers have the following formats:

Control Register 4




Control Register 3




An instruction that uses ASN translation and loads the primary or secondary segment-table designation into the appropriate control register also loads the corresponding ASN into the appropriate control register.

The ASN for the home address space is not assigned a position in a control register.

An access register containing the value 0 or 1 specifies the primary or secondary address space, respectively; and the segment-table designation specified by the access register is in control register 1 or 7, respectively. An access register containing any other value designates an entry in a table called an access list. The designated access-list entry contains the real address of an ASN-second-table entry for the address space specified by the access register. The segment-table designation specified by the access register is in the ASN-second-table entry. Translating the contents of an access register to obtain a segment-table designation for use by DAT does not involve the use of an ASN.

**Note:** Virtual storage consisting of byte locations ordered according to their virtual addresses in an address space is usually referred to as “storage.”

**Programming Note:** Because an ASN-second-table entry is located from an access-list entry by means of its address instead of by means of its ASN, the ASN-second-table entries designated by access-list entries can be


LzLabs GmbH. Ex. 1008-73

"pseudo" ASN-second-table entries, that is, entries which are not in the two-level structure able to be indexed by means of the ASN-translation process. The number of unique pseudo ASN-second-table entries can be greater than the number of unique ASNs and is limited only by the amount of storage available to be occupied by the ASN-second-table entries. Thus, in a sense, there is no limit on the number of possible address spaces.

## ASN Translation

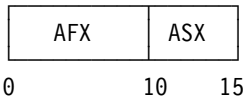
ASN translation is the process of translating a 16-bit ASN to locate the address-space-control parameters. ASN translation may be performed as part of PROGRAM CALL with space switching (PC-ss), it is performed as part of PROGRAM TRANSFER with space switching (PT-ss) and SET SECONDARY ASN with space switching (SSAR-ss), and it may be performed as part of LOAD ADDRESS SPACE PARAMETERS. For PC-ss and PT-ss, the ASN which is translated replaces the primary ASN in control register 4. For SSAR-ss, the ASN which is translated replaces the secondary ASN in control register 3. These two translation processes are called primary ASN translation and secondary ASN translation, respectively, and both can occur for LOAD ADDRESS SPACE PARAMETERS. The ASN-translation process is the same for both primary and secondary ASN translation; only the uses of the results of the process are different.

ASN translation may also be performed as part of PROGRAM RETURN. Primary ASN translation is performed as part of PROGRAM RETURN with space switching (PR-ss). Secondary ASN translation is performed if the secondary ASN restored by PROGRAM RETURN (PR-ss or PROGRAM RETURN to current primary) does not equal the primary ASN restored by PROGRAM RETURN.

The ASN-translation process uses two tables, the ASN first table and the ASN second table. They are used to locate the address-space-control parameters and a third table, the authority table, which is used when ASN authorization is performed.

For the purposes of this translation, the 16-bit ASN is considered to consist of two parts: the ASN-first-table index (AFX) is the leftmost 10 bits of the ASN, and the ASN-second-table index (ASX) is the six rightmost bits. The ASN has the following format:

ASN

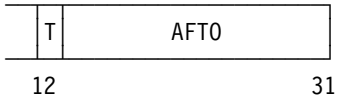



The AFX is used to select an entry from the ASN first table. The origin of the ASN first table is designated by the ASN-first-table origin in control register 14. The ASN-first-table entry contains the origin of the ASN second table. The ASX is used to select an entry from the ASN second table. This entry contains the address-space-control parameters.

## ASN-Translation Controls

ASN translation is controlled by the ASN-translation-control bit and the ASN-first-table origin, both of which reside in control register 14. It is also controlled by the address-space-function-control bit in control register 0.

### Control Register 14




***ASN-Translation Control (T):*** Bit 12 of control register 14 is the ASN-translation-control bit. This bit provides a mechanism whereby the control program can indicate whether ASN translation can occur while a particular program is being executed. Bit 12 must be one to allow completion of these instructions:

- LOAD ADDRESS SPACE PARAMETERS
- PROGRAM CALL with space switching
- PROGRAM RETURN with space switching or when the restored SASN does not equal the restored PASN
- PROGRAM TRANSFER with space switching
- SET SECONDARY ASN

Otherwise, a special-operation exception is recognized. The ASN-translation-control bit is examined in both the problem and the supervisor states.

When the address-space-function-control bit in control register 0 is one, PROGRAM CALL with



LzLabs GmbH. Ex. 1008-74

space switching (PC-ss) may omit performing ASN translation and instead obtain the address of an ASN-second-table entry directly from an entry-table entry. The ASN-translation control must be one whether or not PC-ss performs ASN translation; otherwise, a special-operation exception is recognized.

***ASN-First-Table Origin (AFTO):*** Bits 13-31 of control register 14, with 12 zeros appended on the right, form a 31-bit real address that designates the beginning of the ASN first table.

### Control Register 0

Bit 15 of control register 0 is the address-space-function (ASF) control. When the ASF control is zero, the ASN-second table begins on a 16-byte boundary, an ASN-second-table entry has a length of 16 bytes, and PROGRAM CALL with space switching (PC-ss) always performs ASN translation. When the ASF control is one, the ASN-second table begins on a 64-byte boundary, an ASN-second-table entry has a length of 64 bytes, and PC-ss may obtain an ASN-second-table-entry address from an entry-table entry instead of by performing ASN translation.

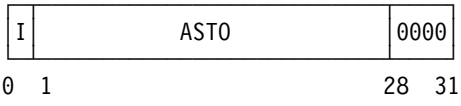
The ASF control has other effects also. A complete description of the effects of the ASF control is in “Address-Space-Function Control” on page 5-42.

## ASN-Translation Tables

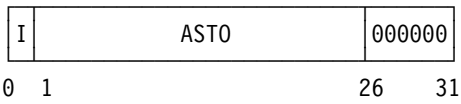
The ASN-translation process consists in a two-level lookup using two tables: an ASN first table and an ASN second table. These tables reside in real storage.

### ASN-First-Table Entries

When the ASF control, bit 15 of control register 0, is zero, an entry in the ASN first table has the following format:




When the ASF control is one, an entry has the following format:




The fields in the entry are allocated as follows:

***AFX-Invalid Bit (I):*** Bit 0 controls whether the ASN second table associated with the ASN-first-table entry is available. When bit 0 is zero, ASN translation proceeds by using the designated ASN second table. When the bit is one, the ASN translation cannot continue.

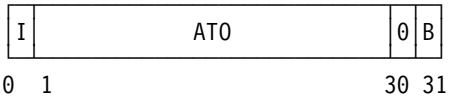
***ASN-Second-Table Origin (ASTO):*** Bits 1-27, with four zeros appended on the right, or bits 1-25, with six zeros appended on the right, are used to form a 31-bit real address that designates the beginning of the ASN second table.

Bits 28-31 of the AFT entry, or bits 26-31, must be zeros; otherwise, an ASN-translation-specification exception may be recognized as part of the execution of the instruction using that entry for ASN translation.

### ASN-Second-Table Entries

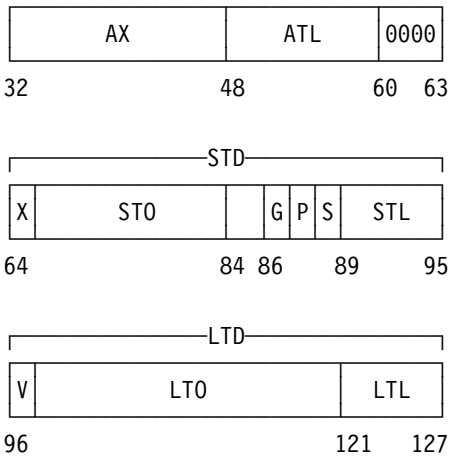
When the ASF control in control register 0 is zero, the ASN-second-table entry has a length of 16 bytes. When the ASF control is one, the entry has a length of 64 bytes. The format of the 16-byte ASN-second-table entry is identical to that of the first 16 bytes of the 64-byte entry. Only the first 16 bytes of the ASN-second-table entry (16-byte entry or 64-byte entry) are used in or as a result of ASN translation. The 16-byte ASN-second-table entry is described below. The 64-byte entry as used by access-register translation for other than the BRANCH IN SUBSPACE GROUP instruction is described in “Extended ASN-Second-Table Entries” on page 5-47. The 64-byte entry as used by BRANCH IN SUBSPACE GROUP is described in “Subspace-Group ASN-Second-Table Entries” on page 5-57.

The 16-byte ASN-second-table entry has the following format:





LzLabs GmbH. Ex. 1008-75




The fields in the entry are allocated as follows:

***ASX-Invalid Bit (I):*** Bit 0 controls whether the address space associated with the ASN-second-table entry is available. When bit 0 is zero, ASN translation proceeds. When the bit is one, the ASN translation cannot continue.

***Authority-Table Origin (ATO):*** Bits 1-29, with two zeros appended on the right, are used to form a 31-bit real address that designates the beginning of the authority table.

***Base-Space Bit (B):*** Bit 31 is ignored during ASN translation if the subspace-group facility is installed and the ASF control is one. If the subspace-group facility is not installed or the ASF control is zero, bit 31 must be zero; otherwise, an ASN-translation-specification exception may be recognized. Bit 31 is further described in "Subspace-Group ASN-Second-Table Entries" on page 5-57.

***Authorization Index (AX):*** Bits 32-47 are used as a result of primary ASN translation by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER and, possibly, LOAD ADDRESS SPACE PARAMETERS. The AX field is ignored for secondary ASN translation.

***Authority-Table Length (ATL):*** Bits 48-59 specify the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four bytes, is one more than the ATL value. The contents of the ATL field are used to establish whether the entry designated by a particular AX falls within the authority table.

***Segment-Table Designation (STD):*** Bits 64-95 are used as a result of ASN translation to replace the primary-segment-table designation (PSTD) or the secondary-segment-table designation (SSTD). For SET SECONDARY ASN, the STD field replaces the SSTD, bits 0-31 of control register 7. For PROGRAM CALL, the STD field replaces the PSTD, bits 0-31 of control register 1. Each of these actions may occur independently for LOAD ADDRESS SPACE PARAMETERS. For PROGRAM TRANSFER, the STD field replaces both the PSTD and the SSTD. For PROGRAM RETURN, as a result of primary ASN translation, the STD field replaces the PSTD, and, as a result of secondary ASN translation, the STD field replaces the SSTD. The contents of the entire STD field are placed in the appropriate control registers without being inspected for validity.

The subspace-group-control bit (G) (bit 86, or bit 22 of the STD field) is an extension provided by the subspace-group facility. The bit indicates, when one, that the STD specifies an address space that is the base space or a subspace of a subspace group. If (1) G is one in the STD placed in a control register as described above, (2) the current dispatchable unit last had control in a subspace of its subspace group instead of in the base space, as indicated by the subspace-active bit being one in the dispatchable-unit control table, and (3) the STD specifies the base space of the group, as indicated by the origin of this AST entry being equal to the base-AST-entry origin in the dispatchable-unit control table, then bits 1-23 and 25-31 of the STD in the control register are replaced by bits 1-23 and 25-31 of the STD for that last entered subspace. The STD for the subspace is obtained from the AST entry designated by the subspace-AST-entry origin in the dispatchable-unit control table.

The storage-alteration-event bit (S) (bit 88, or bit 24 of the STD field) is an extension provided by the program-event-recording-2 (PER-2) facility.

***Space-Switch-Event Control (X):*** Bit 0 of the segment-table designation is the space-switch-event-control bit. When, in PC-ss, PCF-ss, PR-ss, or PT-ss, this bit is one in control register 1 either before or after the execution of the instruction, a program interruption for a space-switch event occurs after the execution of the instruction is completed. A space-switch-event program inter-



LzLabs GmbH. Ex. 1008-76

ruption also occurs after the completion of a SET ADDRESS SPACE CONTROL, SET ADDRESS SPACE CONTROL FAST, or RESUME PROGRAM instruction that changes the translation mode either to or from the home-space mode when this bit is one in either control register 1 or control register 13. When, in LOAD ADDRESS SPACE PARAMETERS, this bit is one during primary ASN translation, this fact is indicated by the condition code.

***Linkage-Table Designation (LTD):*** Bits 96-127 may be used as a result of primary ASN translation and they are used in PC-number translation. The linkage-table-designation field contains the subsystem-linkage-control bit (V) (bit 96), the linkage-table origin (LTO) (bits 97-120), and the linkage-table length (LTL) (bits 121-127). When the ASF control is zero, the contents of the LTD field are placed in control register 5 as a result of primary ASN translation, and the PC-number-translation process obtains the LTD from control register 5. When the ASF control is one, control register 5 contains the origin of an ASN-second-table entry called the primary AST entry. The primary-AST-entry origin is replaced in control register 5 as a result of primary ASN translation, and PC-number translation obtains the LTD from the LTD field in the primary AST entry. PC-number translation is described in Chapter 5, “Program Execution.”

Bits 30, 31, and 60-63 of the AST entry must be zeros; otherwise, an ASN-translation-specification exception may be recognized as part of the execution of the instruction using that entry for ASN translation. However, ASN translation does not require bit 31 to be zero if the subspace-group facility is installed and the ASF control is one.

**Programming Note:** The unused portion of the STD field, bits 84 and 85 of the AST entry, which corresponds to bits 20 and 21 of the STD, should be set to zeros. These bits are reserved for future expansion, and programs which place nonzero values in these bit positions may not operate compatibly on future machines.

## ASN-Translation Process

This section describes the ASN-translation process as it is performed during the execution of the space-switching forms of PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, and SET SECONDARY ASN, and also in PROGRAM RETURN when the restored secondary ASN does not equal the restored primary ASN. ASN translation for LOAD ADDRESS SPACE PARAMETERS is the same except that AFX-translation and ASX-translation exceptions do not occur; such situations are instead indicated by the condition code. Translation of an ASN is performed by means of two tables, an ASN first table and an ASN second table, both of which reside in main storage.

The ASN first index is used to select an entry from the ASN first table. This entry designates the ASN second table to be used.

The ASN second index is used to select an entry from the ASN second table. This entry contains the address-space-control parameters. When the ASF control is one, the ASN second table begins on a 64-byte boundary, and its entries are each 64 bytes in length; otherwise, the table begins on a 16-byte boundary, and the entries are 16 bytes in length.

If the I bit is one in either the ASN-first-table entry or the ASN-second-table entry, the entry is invalid, and the ASN-translation process cannot be completed. An AFX-translation exception or ASX-translation exception is recognized.

Whenever access to main storage is made during the ASN-translation process for the purpose of fetching an entry from an ASN first table or ASN second table, key-controlled protection does not apply.

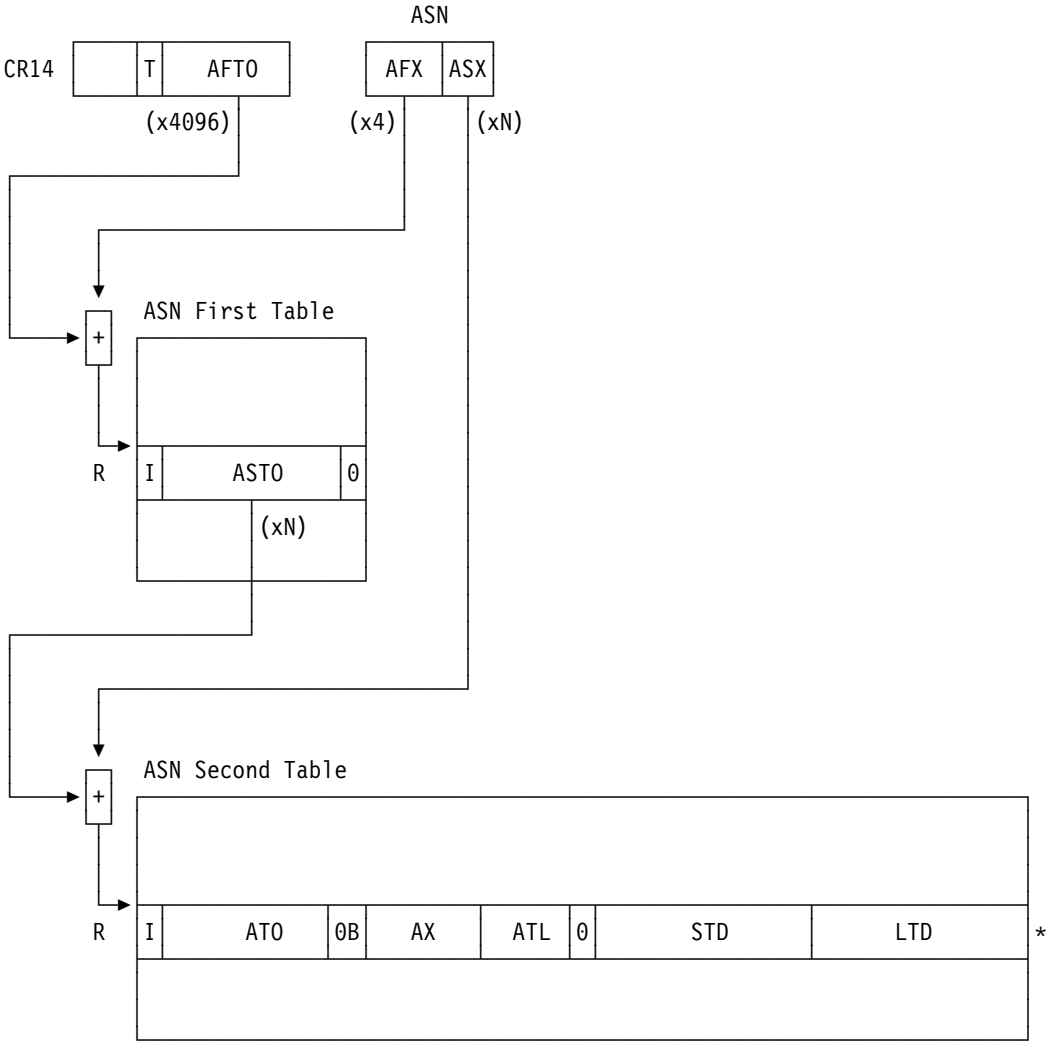
The ASN-translation process is shown in Figure 3-6 on page 3-22.

### ASN-First-Table Lookup

The AFX portion of the ASN, in conjunction with the ASN-first-table origin, is used to select an entry from the ASN first table.

The 31-bit real address of the ASN-first-table entry is obtained by appending 12 zeros on the right to the AFT origin contained in bit positions 13-31 of control register 14 and adding the AFX portion with two rightmost and 19 leftmost zeros appended. This addition cannot cause a carry into bit position 0. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.


LzLabs GmbH. Ex. 1008-77




N: 16 if ASF control, bit 15 of control register 0, is zero; 64 if ASF control is one
R: Address is real
*: ASTE is 64 bytes if ASF control is one; last 48 bytes are not shown

*Figure 3-6. ASN Translation*

All four bytes of the ASN-first-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address which is generated for fetching the ASN-first-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the four-byte AFT entry specifies whether the corresponding AST is available. If this bit is one, an AFX-translation exception is recognized. When the AST-entry size is 16 bytes and bit positions 28-31 of the AFT entry do not contain zeros, or when the AST-entry size is 64 bytes and bit positions 26-31 of the AFT entry do not contain zeros, an ASN-translation-specification exception may be recognized. When no exceptions are recognized, the entry fetched from the AFT is used to access the AST.

## ASN-Second-Table Lookup

The ASX portion of the ASN, in conjunction with the ASN-second-table origin contained in the ASN-first-table entry, is used to select an entry from the ASN second table.

When the address-space-function (ASF) control,



LzLabs GmbH. Ex. 1008-78

bit 15 of control register 0, is zero, the ASN second table begins on a 16-byte boundary, and its entries are each 16 bytes in length. When the ASF control is one, the ASN second table begins on a 64-byte boundary, and its entries are 64 bytes in length.

The 31-bit real address of the ASN-second-table entry is obtained as follows. When the AST-entry size is 16 bytes, the address is obtained by appending four zeros on the right to bits 1-27 of the ASN-first-table entry and adding the ASX with four rightmost and 21 leftmost zeros appended. When the AST-entry size is 64 bytes, the address is obtained by appending six zeros on the right to bits 1-25 of the ASN-first-table entry and adding the ASX with six rightmost and 19 leftmost zeros appended. In both of these cases, when a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31} - 1$ to zero. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

The fetch of the 16 or 64 bytes of the ASN-second-table entry appears to be word concurrent as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining 3 or 15 words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address which is generated for fetching the ASN-second-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the 16-byte or 64-byte ASN-second-table entry specifies whether the address space is accessible. If this bit is one, an ASX-translation exception is recognized. If bit positions 30, 31, and 60-63 of the ASN-second-table entry do not contain zeros, an ASN-translation-specification exception may be recognized. A one in bit position 31 does not cause an ASN-translation-specification exception to be recognized if the subspace-group facility is installed and the ASF control is one.

### Recognition of Exceptions during ASN Translation

The exceptions which can be encountered during the ASN-translation process are collectively referred to as ASN-translation exceptions. A list of these exceptions and their priorities is given in Chapter 6, “Interruptions.”

## ASN Authorization

ASN authorization is the process of testing whether the program associated with the current authorization index is permitted to establish a particular address space. The ASN authorization is performed as part of PROGRAM TRANSFER with space switching (PT-ss) and SET SECONDARY ASN with space switching (SSAR-ss) and may be performed as part of LOAD ADDRESS SPACE PARAMETERS. ASN authorization is performed after the ASN-translation process for these instructions.

ASN authorization is also performed as part of PROGRAM RETURN when the restored secondary ASN does not equal the restored primary ASN. ASN authorization of the restored secondary ASN is performed after ASN translation of the restored secondary ASN.

When performed as part of PT-ss, the ASN authorization tests whether the ASN can be established as the primary ASN and is called primary-ASN authorization. When performed as part of LOAD ADDRESS SPACE PARAMETERS, PROGRAM RETURN, or SSAR-ss, the ASN authorization tests whether the ASN can be established as the secondary ASN and is called secondary-ASN authorization.

The ASN authorization is performed by means of an authority table in real storage which is designated by the authority-table-origin and authority-table-length fields in the ASN-second-table entry.

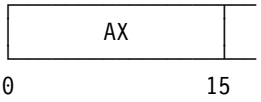
## ASN-Authorization Controls

ASN authorization uses the authority-table origin and the authority-table length from the ASN-second-table entry, together with an authorization index.


LzLabs GmbH. Ex. 1008-79

### Control Register 4

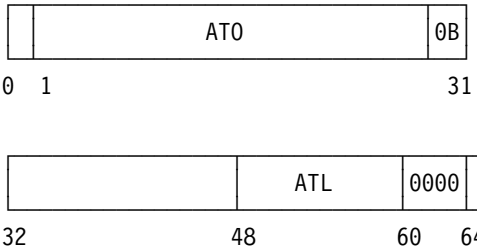
For PT-ss and SSAR-ss, the current contents of control register 4 include the authorization index. For LOAD ADDRESS SPACE PARAMETERS and PROGRAM RETURN, the value which will become the new contents of control register 4 is used. The register has the following format:




***Authorization Index (AX):*** Bits 0-15 of control register 4 are used as an index to locate the authority bits in the authority table.

### ASN-Second-Table Entry

The ASN-second-table entry which is fetched as part of the ASN translation process contains information which is used to designate the authority table. An entry in the ASN second table has the following format:

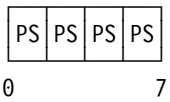



***Authority-Table Origin (ATO):*** Bits 1-29, with two zeros appended on the right, are used to form a 31-bit real address that designates the beginning of the authority table.

***Authority-Table Length (ATL):*** Bits 48-59 specify the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four bytes, is equal to one more than the ATL value. The contents of the length field are used to establish whether the entry designated by the authorization index falls within the authority table.

## Authority-Table Entries

The authority table consists of entries of two bits each; accordingly, each byte of the authority table contains four entries in the following format:




The fields are allocated as follows:

***Primary Authority (P):*** The left bit of an authority-table entry controls whether the program with the authorization index corresponding to the entry is permitted to establish the address space as a primary address space. If the P bit is one, the establishment is permitted. If the P bit is zero, the establishment is not permitted.

***Secondary Authority (S):*** The right bit of an authority-table entry controls whether the program with the corresponding authorization index is permitted to establish the address space as a secondary address space. If the S bit is one, the establishment is permitted. If the S bit is zero, the establishment is not permitted.

The authority table is also used in the extended-authorization process, as part of access-register translation. Extended authorization is described in “Authorizing the Use of the Access-List Entry” on page 5-52.

## ASN-Authorization Process

This section describes the ASN-authorization process as it is performed during the execution of PROGRAM TRANSFER with space switching and SET SECONDARY ASN with space switching. For these two instructions, the ASN-authorization process is performed by using the authorization index currently in control register 4. Secondary authorization for PROGRAM RETURN, when the restored secondary ASN does not equal the restored primary ASN, and for LOAD ADDRESS SPACE PARAMETERS is the same, except that the value which will become the new contents of control register 4 is used for the authorization index. Also, for LOAD ADDRESS SPACE PARAMETERS, a secondary-authority exception does not occur. Instead, such a condition is indicated by the condition code.

The ASN-authorization process is performed by using the authorization index, in conjunction with the authority-table origin and length from the AST entry, to select an authority-table entry. The entry is fetched, and either the primary- or the secondary-authority bit is examined, depending on



LzLabs GmbH. Ex. 1008-80

whether the primary- or secondary-ASN-authorization process is being performed. The ASN-authorization process is shown in Figure 3-7.

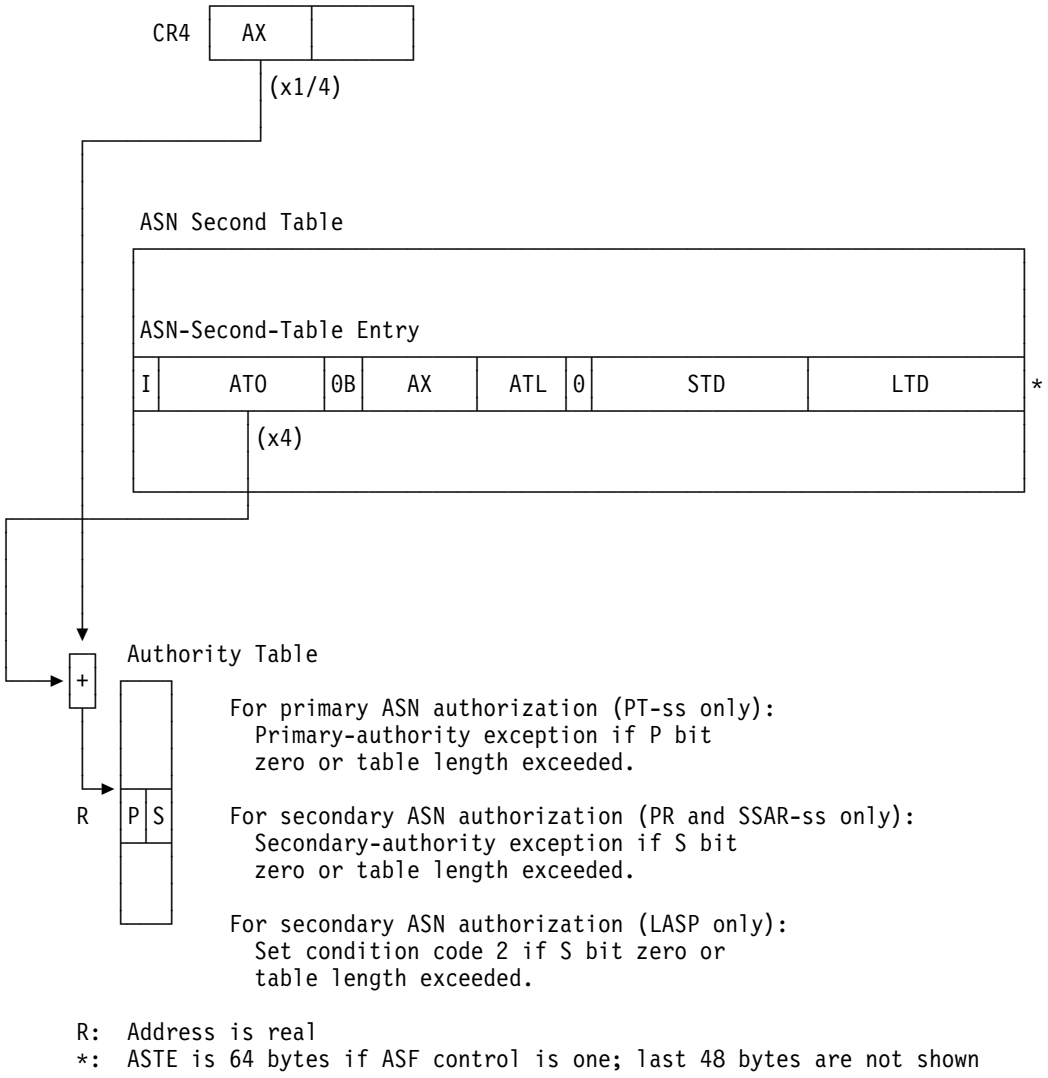



*Figure 3-7. ASN Authorization*

## Authority-Table Lookup

The authorization index, in conjunction with the authority-table origin contained in the ASN-second-table entry, is used to select an entry from the authority table.

The authorization index is contained in bit positions 0-15 of control register 4.

Bit positions 1-29 of the AST entry contain the leftmost 29 bits of the 31-bit real address of the authority table (ATO), and bit positions 48-59 contain the length of the authority table (ATL).

The 31-bit real address of a byte in the authority table is obtained by appending two zeros on the right to the authority-table origin and adding the 14 leftmost bits of the authorization index with 17 zeros appended on the left. when a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31}$ - 1 to zero. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

As part of the authority-table-entry-lookup process, bits 0-11 of the authorization index are compared against the authority-table length. If the compared


LzLabs GmbH. Ex. 1008-81

portion is greater than the authority-table length, a primary-authority exception or secondary-authority exception is recognized for PT-ss or SSAR-ss, respectively. For LOAD ADDRESS SPACE PARAMETERS, when the authority-table length is exceeded, condition code 2 is set.

The fetch access to the byte in the authority table is not subject to protection. When the storage address which is generated for fetching the byte designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

The byte contains four authority-table entries of two bits each. The rightmost two bits of the authorization index, bits 14 and 15 of control register 4, are used to select one of the four entries. The left or right bit of the entry is then tested, depending on whether the authorization test is for a primary ASN or a secondary ASN. The following table shows the bit which is selected from the byte as a function of bits 14 and 15 of the authorization index and the instruction PT-ss, SSAR-ss, PROGRAM RETURN, or LOAD ADDRESS SPACE PARAMETERS.

| Authorization-Index Bits | | Bit Selected from Authority-Table Byte for Test | |
|---|---|---|---|
| 14 | 15 | P Bit (PT-ss) | S Bit (SSAR-ss, PR, or LASP) |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 2 | 3 |
| 1 | 0 | 4 | 5 |
| 1 | 1 | 6 | 7 |

If the selected bit is one, the ASN is authorized, and the appropriate address-space-control parameters from the AST entry are loaded into the appropriate control registers. If the selected bit is zero, the ASN is not authorized, and a primary-authority exception is recognized for PT-ss or a secondary-authority exception is recognized for SSAR-ss or PROGRAM RETURN. For LOAD ADDRESS SPACE PARAMETERS, when the ASN is not authorized, condition code 2 is set.

### Recognition of Exceptions during ASN Authorization

The exceptions which can be encountered during the primary- and secondary-ASN-authorization processes and their priorities are described in the definitions of the instructions in which ASN authorization is performed.

**Programming Note:** The primary- and secondary-authority exceptions cause nullification in order to permit dynamic modification of the authority table. Thus, when an address space is created or "swapped in," the authority table can first be set to all zeros and the appropriate authority bits set to one only when required.

## Dynamic Address Translation

Dynamic address translation (DAT) provides the ability to interrupt the execution of a program at an arbitrary moment, record it and its data in auxiliary storage, such as a direct-access storage device, and at a later time return the program and the data to different main-storage locations for resumption of execution. The transfer of the program and its data between main and auxiliary storage may be performed piecemeal, and the return of the information to main storage may take place in response to an attempt by the CPU to access it at the time it is needed for execution. These functions may be performed without change or inspection of the program and its data, do not require any explicit programming convention for the relocated program, and do not disturb the execution of the program except for the time delay involved.

With appropriate support by an operating system, the dynamic-address-translation facility may be used to provide to a user a system wherein storage appears to be larger than the main storage which is available in the configuration. This apparent main storage is referred to as virtual storage, and the addresses used to designate locations in the virtual storage are referred to as virtual addresses. The virtual storage of a user may far exceed the size of the main storage which is available in the configuration and normally is maintained in auxiliary storage. The virtual storage is considered to be composed of blocks of addresses, called pages. Only the most recently referred-to pages of the virtual storage are assigned to occupy blocks of physical main storage. As the user refers to pages of virtual



LzLabs GmbH. Ex. 1008-82

storage that do not appear in main storage, they are brought in to replace pages in main storage that are less likely to be needed. The swapping of pages of storage may be performed by the operating system without the user's knowledge.

The sequence of virtual addresses associated with a virtual storage is called an address space. With appropriate support by an operating system, the dynamic-address-translation facility may be used to provide a number of address spaces. These address spaces may be used to provide degrees of isolation between users. Such support can consist of a completely different address space for each user, thus providing complete isolation, or a shared area may be provided by mapping a portion of each address space to a single common storage area. Also, instructions are provided which permit a semiprivileged program to access more than one such address space. Dynamic address translation provides for the translation of virtual addresses from multiple different address spaces without requiring that the translation parameters in the control registers be changed. These address spaces are called the primary address space, secondary address space, and AR-specified address spaces. A privileged program can also cause the home address space to be accessed.

In the process of replacing blocks of main storage by new information from an external medium, it must be determined which block to replace and whether the block being replaced should be recorded and preserved in auxiliary storage. To aid in this decision process, a reference bit and a change bit are associated with the storage key.

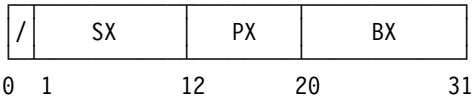
Dynamic address translation may be specified for instruction and data addresses generated by the CPU but is not available for the addressing of data and of CCWs and IDAWs in I/O operations. The CCW-indirect-data-addressing facility is provided to aid I/O operations in a virtual-storage environment.

Address computation can be carried out in either the 24-bit or 31-bit addressing mode. When address computation is performed in the 24-bit addressing mode, seven zeros are appended on the left to form a 31-bit address. Therefore, the resultant logical address is always 31 bits in length. All real and absolute addresses are 31 bits in length.

Dynamic address translation is the process of translating a virtual address during a storage reference into the corresponding real address. The virtual address may be a primary virtual address, secondary virtual address, AR-specified virtual address, or home virtual address. These addresses are translated by means of the primary, the secondary, an AR-specified, or the home segment-table designation, respectively. After selection of the appropriate segment-table designation, the translation process is the same for all of the four types of virtual address.

In the process of translation, two types of units of information are recognized — segments and pages. A segment is a block of sequential virtual addresses spanning 1M bytes and beginning at a 1M-byte boundary. A page is a block of sequential virtual addresses spanning 4K bytes and beginning at a 4K-byte boundary.

The virtual address, accordingly, is divided into three fields. Bits 1-11 are called the segment index (SX), bits 12-19 are called the page index (PX), and bits 20-31 are called the byte index (BX). The virtual address has the following format:




Virtual addresses are translated into real addresses by means of two translation tables: a segment table and a page table. These reflect the current assignment of real storage. The assignment of real storage occurs in units of pages, the real locations being assigned contiguously within a page. The pages need not be adjacent in real storage even though assigned to a set of sequential virtual addresses.

## Translation Control

Address translation is controlled by three bits in the PSW and by a set of bits referred to as the translation parameters. The translation parameters are in control registers 0, 1, 7, and 13. Additional controls are located in the translation tables.

Additional controls are provided as described in Chapter 5, “Program Execution.” These controls determine whether the contents of each access


LzLabs GmbH. Ex. 1008-83

register can be used to obtain a segment-table designation for use by DAT.

## Translation Modes

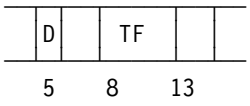
The three bits in the PSW that control dynamic address translation are bit 5, the DAT-mode bit, and bits 16 and 17, the address-space-control bits. When the DAT-mode bit is zero, then DAT is off, and the CPU is in the real mode. When the DAT-mode bit is one, then DAT is on, and the CPU is in the translation mode designated by the address-space-control bits: 00 designates the primary-space mode, 01 designates the access-register mode, 10 designates the secondary-space mode, and 11 designates the home-space mode. The various modes are shown in Figure 3-8, along with the handling of addresses in each mode.

| PSW Bit | | | | | Handling of Addresses | |
|---|---|---|---|---|---|---|
| 5 | 16 | 17 | DAT | Mode | Instruction Addresses | Logical Addresses |
| 0 | 0 | 0 | Off | Real mode | Real | Real |
| 0 | 0 | 1 | Off | Real mode | Real | Real |
| 0 | 1 | 0 | Off | Real mode | Real | Real |
| 0 | 1 | 1 | Off | Real mode | Real | Real |
| 1 | 0 | 0 | On | Primary-space mode | Primary virtual | Primary virtual |
| 1 | 0 | 1 | On | Access-register mode | Primary virtual | AR-specified virtual |
| 1 | 1 | 0 | On | Secondary-space mode | Primary virtual | Secondary virtual |
| 1 | 1 | 1 | On | Home-space mode | Home virtual | Home virtual |

*Figure 3-8. Translation Modes*

## Control Register 0

Six bits are provided in control register 0 for use in controlling dynamic address translation. The bits are assigned as follows:




***Secondary-Space Control (D):*** Bit 5 of control register 0 is the secondary-space-control bit. When this bit is zero and execution of MOVE TO PRIMARY, MOVE TO SECONDARY, or SET ADDRESS SPACE CONTROL is attempted, a special-operation exception is recognized. When this bit is one, it indicates that the secondary segment table is attached when the CPU is in the primary-space mode.

***Translation Format (TF):*** Bits 8-12 of control register 0 specify the translation format, with only one combination of the five control bits valid; all other combinations are invalid.

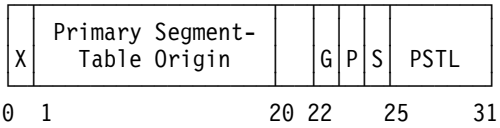
The control bits are encoded as follows:

| Bits of Control Register 0 | | | | | |
|---|---|---|---|---|---|
| 8 | 9 | 10 | 11 | 12 | Valid |
| 1 | 0 | 1 | 1 | 0 | Yes |
| All others | | | | | No |

When an invalid bit combination is detected in bit positions 8-12, a translation-specification exception is recognized as part of the execution of an instruction using address translation.

## Control Register 1

Control register 1 contains the primary segment-table designation (PSTD). The register has the following format:




***Primary Space-Switch-Event Control (X):*** When bit 0 of control register 1 is one:

- A space-switch-event program interruption occurs when execution of the space-switching form of PROGRAM CALL (PC-ss), PROGRAM CALL FAST (PCF-ss), PROGRAM RETURN (PR-ss), or PROGRAM TRANSFER (PT-ss) is completed. The interruption occurs if bit 0 is one either before or after the operation.
- A space-switch-event program interruption occurs upon completion of a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes the address space from which instructions are fetched either to or from the home address space; that is, when instructions are fetched from the home address space either before or after the operation but not both before and after the operation.
- Condition code 3 is set by LOAD ADDRESS SPACE PARAMETERS.



LzLabs GmbH. Ex. 1008-84

***Primary Segment-Table Origin (PSTO):*** Bits 1-19 of control register 1, with 12 zeros appended on the right, form an address that designates the beginning of the primary segment table. It is unpredictable whether the address is real or absolute. This table is called the primary segment table since it is used to translate virtual addresses in the primary address space.

***Primary Subspace-Group Control (G):*** Bit 22, when one, indicates that the address space specified by the STD is the base space or a subspace of a subspace group. When bit 22 is zero, the address space is not in a subspace group.

***Primary Private-Space Control (P):*** If bit 23 of control register 1 is one, then (1) a one value of the common-segment bit in a translation-lookaside-buffer (TLB) segment-table entry prevents the entry and the TLB page-table copy it designates from being used when translating references to the primary address space, even with a match of segment-table origins; (2) low-address protection and fetch-protection override do not apply to the primary address space; and (3) a translation-specification exception is recognized if a reference to the primary address space is translated by means of a segment-table entry in storage and the common-segment bit is one in the entry.

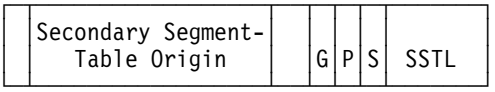
***Primary Storage-Alteration-Event Control (S):*** With PER 2 when the storage-alteration-space control in control register 9 is one, bit 24 of control register 1 specifies, when one, that the primary address space is one for which storage-alteration events can occur. Bit 24 is examined when the segment-table designation is used to perform dynamic-address translation for a storage-operand store reference. Bit 24 is ignored when the storage-alteration-space control is zero, and it is always ignored by PER 1.

***Primary Segment-Table Length (PSTL):*** Bits 25-31 of control register 1 specify the length of the primary segment table in units of 64 bytes, thus making the length of the segment table variable in multiples of 16 entries. The length of the primary segment table, in units of 64 bytes, is one more than the PSTL value. The contents of the length field are used to establish whether the entry designated by the segment-index portion of a primary virtual address falls within the primary segment table.

Bits 20 and 21 of control register 1 are not assigned and are ignored. Bit 22 is ignored if the subspace-group facility is not installed. Bit 24 is ignored if the PER-2 facility is not installed.

## Control Register 7

Control register 7 contains the secondary segment-table designation (SSTD). The register has the following format:

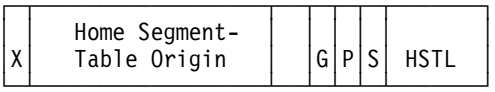



The secondary segment-table origin, secondary subspace-group control (G), secondary private-space control (P), secondary storage-alteration-event control (S), and secondary segment-table length (SSTL) in control register 7 are defined the same as the fields in the same bit positions in control register 1, except that control register 7 applies to the secondary address space.

Bits 0, 20, and 21 of control register 7 are not assigned and are ignored. Bit 22 is ignored if the subspace-group facility is not installed. Bit 24 is ignored if the PER-2 facility is not installed.

## Control Register 13

Control register 13 contains the home segment-table designation (HSTD). The register has the following format:




***Home Space-Switch-Event Control (X):*** When bit 0 of control register 13 is one, a space-switch-event program interruption occurs upon completion of a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes the address space from which instructions are fetched either to or from the home address space; that is, when instructions are fetched from the home address space either before or after the operation but not both before and after the operation.


LzLabs GmbH. Ex. 1008-85

The home segment-table origin, home private-space control (P), home storage-alteration-event control (S), and home segment-table length (HSTL) in control register 13 are defined the same as the fields in the same bit positions in control register 1, except that control register 13 applies to the home address space.

Bits 20 and 21 of control register 13 are not assigned and are ignored. Bit 22 (G) is ignored. Bit 24 is ignored if the PER-2 facility is not installed.

**Programming Notes:**

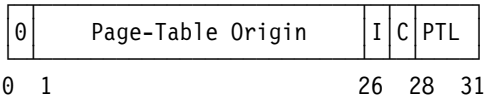
1. The validity of the information loaded into a control register, including that pertaining to dynamic address translation, is not checked at the time the register is loaded. This information is checked and the program exception, if any, is indicated at the time the information is used.
2. The information pertaining to dynamic address translation is considered to be used when an instruction is executed with DAT on or when INVALIDATE PAGE TABLE ENTRY or LOAD REAL ADDRESS is executed. The information is not considered to be used when the PSW specifies translation but an I/O, external, restart, or machine-check interruption occurs before an instruction is executed, or when the PSW specifies the wait state.

## Translation Tables

The translation process consists in a two-level lookup using two tables: a segment table and a page table. These tables reside in real or absolute storage.

### Segment-Table Entries

The entry fetched from the segment table has the following format:




The fields in the segment-table entry are allocated as follows:

***Page-Table Origin (PTO):*** Bits 1-25, with six zeros appended on the right, form the address that designates the beginning of a page table. It is unpredictable whether the address is real or absolute.

***Segment-Invalid Bit (I):*** Bit 26 controls whether the segment associated with the segment-table entry is available. When the bit is zero, address translation proceeds by using the segment-table entry. When the bit is one, the segment-table entry cannot be used for translation.

***Common-Segment Bit (C):*** Bit 27 controls the use of the translation-lookaside-buffer (TLB) copies of the segment-table entry and of the page table which it designates. A zero identifies a private segment; in this case, the segment-table entry and the page table it designates may be used only in association with the segment-table origin that designates the segment table in which the segment-table entry resides. A one identifies a common segment; in this case, the segment-table entry and the page table it designates may continue to be used for translating addresses corresponding to the segment index, even though a different segment table is specified. However, TLB copies of the segment-table entry and page table for a common segment are not usable if the private-space control, bit 23, is one in the segment-table designation used in the translation. The common-segment bit must be zero if the segment-table entry is fetched from storage during a translation when the private-space control is one in the segment-table designation being used; otherwise, a translation-specification exception is recognized.

***Page-Table Length (PTL):*** Bits 28-31 specify the length of the page table in units of 64 bytes (16 entries). The length of the page table, in units of 64 bytes, is one more than the PTL value. The contents of the length field are used to establish whether the entry designated by the page-index portion of the virtual address falls within the page table.

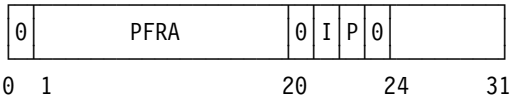
Bit 0 of the segment-table entry must be zero; if it is not zero, a translation-specification exception is recognized as part of the execution of an instruction using that entry for address translation.

### Page-Table Entries

The entry fetched from the page table has the following format:



LzLabs GmbH. Ex. 1008-86




The fields in the page-table entry are allocated as follows:

***Page-Frame Real Address (PFRA):*** Bits 1-19 provide the leftmost bits of a real storage address. When these bits are concatenated with the 12-bit byte-index field of the virtual address on the right, a 31-bit real address is obtained.

***Page-Invalid Bit (I):*** Bit 21 controls whether the page associated with the page-table entry is available. When the bit is zero, address translation proceeds by using the page-table entry. When the bit is one, the page-table entry cannot be used for translation.

***Page-Protection Bit (P):*** Bit 22 controls whether store accesses can be made in the page. This protection mechanism is in addition to the key-controlled-protection and low-address-protection mechanisms. The bit has no effect on fetch accesses. If the bit is zero, stores are permitted to the page, subject to the other protection mechanisms. If the bit is one, stores are disallowed. An attempt to store when the page-protection bit is one causes a protection exception to be recognized.

Bit positions 0, 20, and 23 of the entry must contain zeros; otherwise, a translation-specification exception is recognized as part of the execution of an instruction using that entry for address translation. Bit positions 24-31 are not assigned and are ignored.

### Summary of Segment-Table and Page-Table Sizes

The sizes of segment tables and page tables are summarized in Figure 3-9.

| Segment-Table Parameters | | | | |
|---|---|---|---|---|
| Virtual Address Size (Bits) | Number of Addressable Segments | Corresponding Segment Table | | Segment-Table Increment (Bytes) |
| | | Maximum Size (Bytes) | Usable Length Code | |
| 24[1] | 16 | 64 | 0 | -- |
| 31 | 2,048 | 8,192 | 127 | 64 |

| Page-Table Parameters[2] | | | |
|---|---|---|---|
| Number of Pages in Segment | Corresponding Page Table | | Page-Table Increment (Bytes) |
| | Maximum Size (Bytes) | Usable Length Code | |
| 256 | 1,024 | 15 | 64 |

**Explanation**:

1 A virtual address specified by the program in the 24-bit addressing mode consists of a 24-bit value embedded in a 31-bit address.

2 The page-table size is independent of the virtual address size.

*Figure 3-9. Sizes of Segment Tables and Page Tables*

## Translation Process

This section describes the translation process as it is performed implicitly before a virtual address is used to access main storage. Explicit translation, which is the process of translating the operand address of LOAD REAL ADDRESS and TEST PROTECTION, is the same, except that segment-translation and page-translation exceptions do not occur; such conditions are instead indicated by the condition code. Translation of the operand address of LOAD REAL ADDRESS also differs in that the CPU may be in the real mode and, on models without z/Architecture installed, the translation-lookaside buffer is not used.

Translation of a virtual address is performed by means of a segment table and a page table, both of which reside in real or absolute storage. It is controlled by the DAT-mode bit and the address-space-control bits, all in the PSW. The translation tables are designated by fields in control registers 1, 7, and 13 and as specified by the access registers.


LzLabs GmbH. Ex. 1008-87

## Effective Segment-Table Designation

The segment-table designation used for a particular address translation is called the effective segment-table designation. Accordingly, when a primary virtual address is translated, the contents of control register 1 are used as the effective segment-table designation. Similarly, for a secondary virtual address, the contents of control register 7 are used; for an AR-specified virtual address, the segment-table designation specified by the access register is used; and for a home virtual address, the contents of control register 13 are used.

The segment-index portion of the virtual address is used to select an entry from the segment table, the starting address and length of which are specified by the effective segment-table designation. This entry designates the page table to be used.

The page-index portion of the virtual address is used to select an entry from the page table. This entry contains the leftmost bits of the real address that represents the translation of the virtual address and provides the page-protection bit.

The byte-index field of the virtual address is used unchanged as the rightmost bit positions of the real address.

If the I bit is one in either the segment-table entry or the page-table entry, the entry is invalid, and the translation process cannot be completed for this virtual address. A segment-translation or page-translation exception is recognized.

In order to eliminate the delay associated with references to translation tables in real or absolute storage, the information fetched from the tables normally is also placed in a special buffer, the translation-lookaside buffer (TLB), and subsequent translations involving the same table entries may be performed by using the information recorded in the TLB. The operation of the TLB is described in "Translation-Lookaside Buffer" on page 3-35.

Whenever access to real or absolute storage is made during the address-translation process for the purpose of fetching an entry from a segment table or page table, key-controlled protection does not apply.

The translation process, including the effect of the TLB, is shown graphically in Figure 3-10 on page 3-33.



LzLabs GmbH. Ex. 1008-88

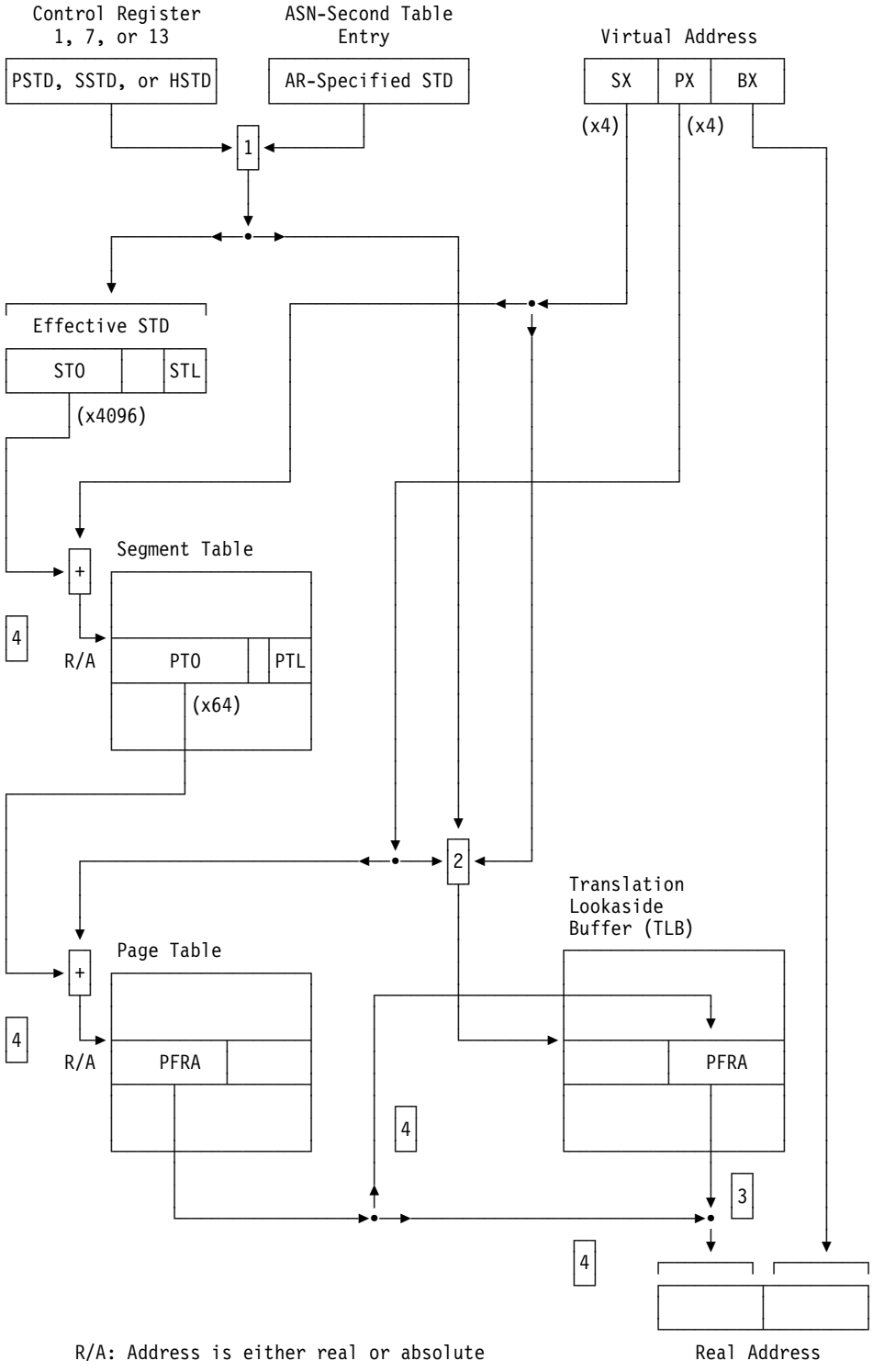



*Figure 3-10 (Part 1 of 2). Translation Process*


LzLabs GmbH. Ex. 1008-89

1 Control register 1 provides the primary segment-table designation for translation of a primary virtual address, control register 7 provides the secondary segment-table designation for translation of a secondary virtual address, and control register 13 provides the home segment-table designation for translation of a home virtual address. An ASN-second-table entry provides an AR-specified (access-register-specified) segment-table designation for translation of an AR-specified virtual address.

2 Information, which may include portions of the virtual address and the effective segment-table origin, is used to search the TLB.

3 If a match exists, the page-frame real address from the TLB is used in forming the real address.

4 If no match exists, table entries in real or absolute storage are fetched. The resulting fetched entries, in conjunction with the search information, are used to translate the address and may be used to form an entry in the TLB.

*Figure 3-10 (Part 2 of 2). Translation Process*

## Inspection of Control Register 0

The interpretation of the virtual address for translation purposes requires that there be a valid translation format specified by bits 8-12 of control register 0. If bits 8-12 contain an invalid code, a translation-specification exception is recognized.

## Segment-Table Lookup

The segment-index portion of the virtual address, in conjunction with the segment-table origin contained in the effective segment-table designation, is used to select an entry from the segment table.

The 31-bit address of the segment-table entry in real or absolute storage is obtained by appending 12 zeros to the right of bits 1-19 of the effective segment-table designation and adding the segment index with two rightmost and 18 leftmost zeros appended. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31} - 1$ to zero. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

As part of the segment-table-lookup process, bits 1-7 of the virtual address are compared against the segment-table length in bit positions 25-31 of the effective segment-table designation to establish whether the addressed entry is within the segment table. If the value in the segment-table-length field is less than the value in the corresponding bit positions of the virtual address, a segment-translation exception is recognized. The comparison against the segment-table length may be omitted if a segment-table entry in the translation-lookaside buffer is used in the translation.

All four bytes of the segment-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address generated for fetching the segment-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the unit of operation is suppressed.

Bit 26 of the entry fetched from the segment table specifies whether the corresponding segment is available. This bit is inspected, and, if it is one, a segment-translation exception is recognized. If bit 0 of the entry is one, a translation-specification exception is recognized. A translation-specification exception is also recognized if (1) the private-space control, bit 23, in the effective segment-table designation is one and (2) the common-segment bit, bit 27, in the entry fetched from the segment table is one.

When no exceptions are recognized in the process of segment-table lookup, the entry fetched from the segment table designates the beginning and specifies the length of the corresponding page table.



LzLabs GmbH. Ex. 1008-90

The common-segment bit in the entry fetched from the segment table is further used only for the purpose of forming a TLB entry (see “Use of TLB Entries” on page 3-37).

### Page-Table Lookup

The page-index portion of the virtual address, in conjunction with the page-table origin contained in the segment-table entry, is used to select an entry from the page table.

The 31-bit address of the page-table entry in real or absolute storage is obtained by appending six zeros to the right of the page-table origin and adding the page index, with two rightmost and 21 leftmost zeros appended. A carry into bit position 0 may cause an addressing exception to be recognized, or the carry may be ignored, causing the page table to wrap from $2^{31} - 1$ to zero. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

As part of the page-table-lookup process, the four leftmost bits of the page index are compared against the page-table length, bits 28-31 of the segment-table entry, to establish whether the addressed entry is within the table. If the value in the page-table-length field is less than the value in the four leftmost bit positions of the page-index field, a page-translation exception is recognized.

All four bytes of the page-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address generated for fetching the page-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the unit of operation is suppressed.

The entry fetched from the page table indicates the availability of the page and contains the leftmost bits of the page-frame real address. The page-invalid bit is inspected to establish whether the corresponding page is available. If this bit is one, a page-translation exception is recognized. If bit position 0, 20, or 23 contains a one, a translation-specification exception is recognized.

### Formation of the Real Address

When no exceptions in the translation process are encountered, the page-frame real address obtained from the page-table entry and the byte-index portion of the virtual address are concatenated, with the page-frame real address forming the leftmost part. The result is the real storage address which corresponds to the virtual address. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

### Recognition of Exceptions during Translation

Invalid addresses and invalid formats can cause exceptions to be recognized during the translation process. Exceptions are recognized when information contained in control registers or table entries is used for translation and is found to be incorrect.

The information pertaining to DAT is considered to be used when an instruction is executed with DAT on or when INVALIDATE PAGE TABLE ENTRY or LOAD REAL ADDRESS is executed. The information is not considered to be used when the PSW specifies DAT on but an I/O, external, restart, or machine-check interruption occurs before an instruction is executed, or when the PSW specifies the wait state. Only that information required in order to translate a virtual address is considered to be in use during the translation of that address, and, in particular, addressing exceptions that would be caused by the use of a segment-table designation are not recognized when that segment-table designation is not the one actually used in the translation.

A list of translation exceptions, with the action taken for each exception and the priority in which the exceptions are recognized when more than one is applicable, is provided in “Recognition of Access Exceptions” on page 6-35.

## Translation-Lookaside Buffer

To enhance performance, the dynamic-address-translation mechanism normally is implemented such that some of the information specified in the segment and page tables is maintained in a special buffer, referred to as the translation-lookaside buffer (TLB). The CPU necessarily refers to a DAT-table entry in real or absolute


LzLabs GmbH. Ex. 1008-91

storage only for the initial access to that entry. This information may be placed in the TLB, and subsequent translations may be performed by using the information in the TLB. The presence of the TLB affects the translation process to the extent that (1) a modification of the contents of a table entry in real or absolute storage does not necessarily have an immediate effect, if any, on the translation, and (2) the comparison against the segment-table length in the effective segment-table designation may be omitted if a TLB segment-table entry is used. In a multiple-CPU configuration, each CPU has its own TLB.

Entries within the TLB are not explicitly addressable by the program.

Information is not necessarily retained in the TLB under all conditions for which such retention is permissible. Furthermore, information in the TLB may be cleared under conditions additional to those for which clearing is mandatory.

## TLB Structure

The description of the logical structure of the TLB covers the implementation by all systems operating as defined by ESA/390. The TLB entries are considered as being of two types: TLB segment-table entries and TLB page-table entries. A TLB entry is considered as containing within it both the information obtained from the table entry in real or absolute storage and the attributes used to fetch the entry from storage.

**Note:** The following sections describe the conditions under which information may be placed in the TLB, the conditions under which information from the TLB may be used for address translation, and how changes to the translation tables affect the translation process.

## Formation of TLB Entries

The formation of TLB entries and the effect of any manipulation of the contents of a table entry in real or absolute storage by the program depend on whether the entry is attached to a particular CPU and on whether the entry is valid.

The *attached* state of a table entry denotes that the CPU to which it is attached can attempt to use the table entry for implicit address translation, except that a table entry for the primary or home address space may be attached even when the CPU does not fetch from either of those spaces. A table entry may be attached to more than one CPU at a time.

The *valid* state of a table entry denotes that the segment or page associated with the table entry is available. An entry is valid when the segment-invalid bit or page-invalid bit in the entry is zero.

A segment-table entry or a page-table entry may be placed in the TLB whenever the entry is attached and valid and would not cause a translation-specification exception if used for translation.

A segment-table entry is attached when all of the following conditions are met:

1. The current PSW specifies DAT on.
2. The current PSW contains no errors that would cause an early specification exception to be recognized.
3. The current translation format, bits 8-12 in control register 0, is valid.
4. The entry meets the requirements in a, b, c, or d below.
   a. The entry is within the segment table designated by the primary segment-table designation in control register 1.
   b. The entry is within the segment table designated by the secondary segment-table designation in control register 7 and either of the following requirements is met:
      - The CPU is in the secondary-space mode or access-register mode.
      - The CPU is in the primary-space mode, and the secondary-space control, bit 5 of control register 0, is one.
   c. The entry is within a segment table for which the designation is in either an attached and valid ASN-second-table entry (ASTE) or a usable ALB ASTE, and the CPU is in the access-register mode. See “ART-Lookaside Buffer” on page 5-53 for the meaning of the terminology used here.
   d. The entry is within the segment table specified by the home segment-table designation in control register 13.



LzLabs GmbH. Ex. 1008-92

A page-table entry is attached when it is within the page table designated by either an attached and valid segment-table entry that would not cause a translation-specification exception if used for translation or a usable TLB segment-table entry. A usable TLB segment-table entry is explained in the next section.

## Use of TLB Entries

The *usable* state of a TLB entry denotes that the CPU can attempt to use the TLB entry for implicit address translation. A usable TLB entry attaches the next-lower-level table, if any, and may be usable for a particular instance of implicit address translation.

A TLB segment-table entry is in the usable state when all of the following conditions are met:

1. The current PSW specifies DAT on.
2. The current PSW contains no errors that would cause an early specification exception to be recognized.
3. The current translation format, bits 8-12 in control register 0, is valid.
4. The TLB segment-table entry meets at least one of the following requirements:
   a. The common-segment bit is one in the TLB entry.
   b. The segment-table-origin field in the TLB entry matches the current PSTO, and the CPU is not in the home-space mode.
   c. The segment-table-origin field in the TLB entry matches the current SSTO, and either of the following requirements is met:
      - The CPU is in the secondary-space mode or access-register mode.
      - The CPU is in the primary-space mode, and the secondary-space control, bit 5 of control register 0, is one.
   d. The segment-table-origin field in the TLB entry matches the segment-table-origin field in an attached and valid ASN-second-table entry (ASTE) or a usable ALB ASTE, and the CPU is in the access-register mode.
   e. The segment-table-origin field in the TLB entry matches the current HSTO, and the CPU is not in the secondary-space mode.

A TLB segment-table entry may be used for a particular instance of implicit address translation only when the entry is in the usable state, either the common-segment bit is one in the TLB entry or the segment-table-origin field in the TLB entry matches the segment-table origin being used in the translation, and the segment-index field in the TLB entry matches the segment index of the virtual address being translated. However, a TLB segment-table entry is not used if the common-segment bit is one in the entry and the private-space-control bit is one in the segment-table designation, even if the segment-table-origin fields in the entry and the designation match.

A TLB page-table entry may be used for a particular instance of implicit address translation only when the page-table-origin field in the TLB page-table entry matches the page-table-origin field in the segment-table entry or TLB segment-table entry being used in the translation, the value of the page-index field in the TLB page-table entry is within the range permitted by the page-table-length field in the segment-table entry or TLB segment-table entry, and the page-index field in the TLB page-table entry matches the page index of the virtual address being translated.

On a model without z/Architecture installed, the operand address of LOAD REAL ADDRESS is translated without the use of the TLB contents. Translation in this case is performed by the use of the designated tables in real or absolute storage. If z/Architecture is installed, LOAD REAL ADDRESS may use the TLB whether DAT is on or off, but TLB entries still are formed only if DAT is on.

**Programming Notes:**

1. Although a table entry may be copied into the TLB only when the table entry is both attached and valid, the copy may remain in the TLB even when the table entry itself is no longer attached or valid.
2. No entries can be copied into the TLB when DAT is off because the table entries at this time are not attached. In particular, translation of the operand address of LOAD REAL ADDRESS with DAT off does not cause entries to be placed in the TLB.

   Conversely, when DAT is on, information may be copied into the TLB from all translation-


LzLabs GmbH. Ex. 1008-93

table entries that could be used for address translation, given the current translation parameters, the setting of the address-space-control bits, the setting of the secondary-space-control bit, and the contents of the access registers. The loading of the TLB does not depend on whether the entry is used for translation as part of the execution of the current instruction, and such loading can occur when the CPU is in the wait state.

3. More than one copy of a table entry may exist in the TLB. For example, some implementations may cause a copy of a valid table entry to be placed in the TLB for each segment-table origin by which the entry becomes attached.

## Modification of Translation Tables

When an attached and invalid table entry is made valid and no usable entry for the associated virtual address is in the TLB, the change takes effect no later than the end of the current unit of operation. Similarly, when an unattached and valid table entry is made attached and no usable entry for the associated virtual address is in the TLB, the change takes effect no later than the end of the current unit of operation.

When a valid and attached table entry is changed, and when, before the TLB is cleared of entries that qualify for substitution for that entry, an attempt is made to refer to storage by using a virtual address requiring that entry for translation, unpredictable results may occur, to the following extent. The use of the new value may begin between instructions or during the execution of an instruction, including the instruction that caused the change. Moreover, until the TLB is cleared of entries that qualify for substitution for that entry, the TLB may contain both the old and the new values, and it is unpredictable whether the old or new value is selected for a particular access. If both old and new values of a segment-table entry are present in the TLB, a page-table entry may be fetched by using one value and placed in the TLB associated with the other value. If the new value of the entry is a value that would cause an exception, the exception may or may not cause an interruption to occur. If an interruption does occur, the result fields of the instruction may be changed even though the exception would normally cause suppression or nullification.

Entries are cleared from the TLB in accordance with the following rules:

1. All entries are cleared from the TLB by the execution of PURGE TLB and SET PREFIX and by CPU reset.
2. All entries may be cleared from all TLBs in the configuration by the execution of COMPARE AND SWAP AND PURGE by any of the CPUs in the configuration, depending on a bit in a general register used by the instruction.
3. Selected entries are cleared from all TLBs in the configuration by the execution of INVALIDATE PAGE TABLE ENTRY by any of the CPUs in the configuration.
4. Some or all TLB entries may be cleared at times other than those required by the preceding rules.

**Programming Notes:**

1. Entries in the TLB may continue to be used for translation after the table entries from which they have been formed have become unattached or invalid. These TLB entries are not necessarily removed unless explicitly cleared from the TLB.

   A change made to an attached and valid entry or a change made to a table entry that causes the entry to become attached and valid is reflected in the translation process for the next instruction, or earlier than the next instruction, unless a TLB entry qualifies for substitution for that table entry. However, a change made to a table entry that causes the entry to become unattached or invalid is not necessarily reflected in the translation process until the TLB is cleared of entries that qualify for substitution for that table entry.

2. Exceptions associated with dynamic address translation may be established by a pretest for operand accessibility that is performed as part of the initiation of instruction execution. Consequently, a segment-translation or page-translation exception may be indicated when a table entry is invalid at the start of execution even if the instruction would have validated the table entry it uses and the table entry would have appeared valid if the instruction was considered to process the operands one byte at a time.



LzLabs GmbH. Ex. 1008-94

3. A change made to an attached table entry, except to set the I bit to zero or to alter the rightmost byte of a page-table entry, may produce unpredictable results if that entry is used for translation before the TLB is cleared of all copies of that entry. The use of the new value may begin between instructions or during the execution of an instruction, including the instruction that caused the change. When an instruction, such as MOVE (MVC), makes a change to an attached table entry, including a change that makes the entry invalid, and subsequently uses the entry for translation, a changed entry is being used without a prior clearing of the entry from the TLB, and the associated unpredictability of result values and of exception recognition applies.

   Manipulation of attached table entries may cause spurious table-entry values to be recorded in a TLB. For example, if changes are made piecemeal, modification of a valid attached entry may cause a partially updated entry to be recorded, or, if an intermediate value is introduced in the process of the change, a supposedly invalid entry may temporarily appear valid and may be recorded in the TLB. Such an intermediate value may be introduced if the change is made by an I/O operation that is retried, or if an intermediate value is introduced during the execution of a single instruction.

   As another example, if a segment-table entry is changed to designate a different page table and used without clearing the TLB, the new page-table entries may be fetched and associated with the old page-table origin. In such a case, execution of INVALIDATE PAGE TABLE ENTRY designating the new page-table origin will not necessarily clear the page-table entries fetched from the new page table.

4. To facilitate the manipulation of translation tables, INVALIDATE PAGE TABLE ENTRY is provided, which sets the I bit in a page-table entry to one and clears all TLBs in the configuration of entries formed from that table entry.

   INVALIDATE PAGE TABLE ENTRY is useful for setting the I bit to one in a page-table entry and causing TLB copies of the entry to be cleared from the TLB of each CPU in the configuration. The following aspects of the TLB operation should be considered when using INVALIDATE PAGE TABLE ENTRY. (See also the programming notes for INVALIDATE PAGE TABLE ENTRY.)

   a. INVALIDATE PAGE TABLE ENTRY should be executed before making any change to a page-table entry other than changing the rightmost byte; otherwise, the selective-clearing portion of INVALIDATE PAGE TABLE ENTRY may not clear the TLB copies of the entry.

   b. Invalidation of all the page-table entries within a page table by means of INVALIDATE PAGE TABLE ENTRY does not necessarily clear the TLB of the copies, if any, of the segment-table entry designating the page table. When it is desired to invalidate and clear the TLB of a segment-table entry, the rules in note 5 below must be followed.

   c. When a large number of page-table entries are to be invalidated at a single time, the overhead involved in using PURGE TLB or COMPARE AND SWAP AND PURGE (one that purges the TLB) and in following the rules in note 5 below may be less than in issuing INVALIDATE PAGE TABLE ENTRY for each page-table entry.

5. Manipulation of table entries should be in accordance with the following rules. If these rules are complied with, translation is performed as if the table entries from real or absolute storage were always used in the translation process.

   a. A valid table entry must not be changed while it is attached to any CPU and may be used for translation by that CPU except to (1) invalidate the entry by using INVALIDATE PAGE TABLE ENTRY, (2) alter bits 24-31 of a page-table entry, or (3) make a change by means of a COMPARE AND SWAP AND PURGE instruction that purges the TLB.

   b. When any change is made to an attached and valid or unattached or invalid table entry other than a change to bits 24-31 of a page-table entry, each CPU which may have a TLB entry formed from that entry must be caused to purge its TLB after the change occurs and prior to the use of that entry for implicit translation by that CPU.


LzLabs GmbH. Ex. 1008-95

(Note that a separate purge is unnecessary if the change was made by using INVALIDATE PAGE TABLE ENTRY or a COMPARE AND SWAP AND PURGE instruction that purges the TLB.) In the case when the table entry is attached and valid, this rule applies when it is known that a program is not being executed that may require the entry for translation.

c. When any change is made to an invalid table entry in such a way as to allow intermediate valid values to appear in the entry, each CPU to which the entry is attached must be caused to purge its TLB after the change occurs and prior to the use of the entry for implicit address translation by that CPU.

d. When any change is made to a length specified for a segment table or page table, each CPU which may have a TLB entry formed from a table entry that no longer lies within its table must be caused to purge its TLB after the change occurs and prior to the use of the table for implicit translation by that CPU.

Note that when an invalid page-table entry is made valid without introducing intermediate valid values, the TLB need not be cleared in a CPU which does not have any TLB entries formed from that entry. Similarly, when an invalid segment-table entry is made valid without introducing intermediate valid values, the TLB need not be cleared in a CPU which does not have any TLB entries formed from that segment-table entry and which does not have any TLB entries formed from entries in a page table attached by the segment-table entry.

The execution of PURGE TLB, COMPARE AND SWAP AND PURGE, or SET PREFIX may have an adverse effect on the performance of some models. Use of these instructions should, therefore, be minimized in conformance with the above rules.

## Address Summary

## Addresses Translated

Most addresses that are explicitly specified by the program and are used by the CPU to refer to storage are instruction or logical addresses and are subject to implicit translation when DAT is on. Analogously, the corresponding addresses indicated to the program on an interruption or as the result of executing an instruction are instruction or logical addresses. The operand address of LOAD REAL ADDRESS is explicitly translated, regardless of whether the PSW specifies DAT on or off.

Translation is not applied to quantities that are formed from the values specified in the B and D fields of an instruction but that are not used to address storage. This includes operand addresses in LOAD ADDRESS, LOAD ADDRESS EXTENDED, MONITOR CALL, and the shifting instructions. This also includes the addresses in control registers 10 and 11 designating the starting and ending locations for PER.

With the exception of INSERT VIRTUAL STORAGE KEY and TEST PROTECTION, the addresses explicitly designating storage keys (operand addresses in SET STORAGE KEY EXTENDED, INSERT STORAGE KEY EXTENDED, and RESET REFERENCE BIT EXTENDED) are real addresses. Similarly, the addresses implicitly used by the CPU for such sequences as interruptions are real addresses.

The addresses used by channel programs to transfer data and to refer to CCWs or IDAWs are absolute addresses.

The handling of storage addresses associated with DIAGNOSE is model-dependent.

The processing of addresses, including dynamic address translation and prefixing, is discussed in “Address Types” on page 3-3. Prefixing, when provided, is applied after the address has been translated by means of the dynamic-address-translation facility. For a description of prefixing, see “Prefixing” on page 3-15.

## Handling of Addresses

The handling of addresses is summarized in Figure 3-11 on page 3-41. This figure lists all addresses that are encountered by the program and specifies the address type.



LzLabs GmbH. Ex. 1008-96

Virtual Addresses

- Address of storage operand for INSERT VIRTUAL STORAGE KEY
- Operand address in LOAD REAL ADDRESS
- Addresses of storage operands for MOVE TO PRIMARY and MOVE TO SECONDARY
- Address stored in the word at real location 144 on a program interruption for page-translation or segment-translation exception
- Linkage-stack-entry address in control register 15
- Backward stack-entry address in linkage-stack header entry
- Forward-section-header address in linkage-stack trailer entry
- Trap-control-block address in dispatchable-unit-control table
- Trap-save-area address and trap-program address in trap control block

Instruction Addresses

- Instruction address in PSW
- Branch address
- Target of EXECUTE
- Address stored in the word at real location 152 on a program interruption for PER
- Address placed in general register by BRANCH AND LINK, BRANCH AND SAVE, BRANCH AND SAVE AND SET MODE, BRANCH AND STACK, BRANCH IN SUBSPACE GROUP, BRANCH RELATIVE AND SAVE, BRANCH RELATIVE AND SAVE LONG, and PROGRAM CALL
- Address used in general register by BRANCH AND STACK.
- Address placed in general register by BRANCH AND SET AUTHORITY executed in reduced-authority state

Logical Addresses

- Addresses of storage operands for instructions not otherwise specified
- Address placed in general register 1 by EDIT AND MARK and TRANSLATE AND TEST
- Addresses in general registers updated by MOVE LONG, MOVE LONG EXTENDED, COMPARE LOGICAL LONG, and COMPARE LOGICAL LONG EXTENDED
- Addresses in general registers updated by CHECKSUM, COMPARE AND FORM CODEWORD, and UPDATE TREE
- Address for TEST PENDING INTERRUPTION when the second-operand address is nonzero
- Address of parameter list of RESUME PROGRAM

*Figure 3-11 (Part 1 of 3). Handling of Addresses*


LzLabs GmbH. Ex. 1008-97

Real Addresses

- Address of storage key for INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED
- Address of storage operand for LOAD USING REAL ADDRESS, STORE USING REAL ADDRESS, and TEST BLOCK
- The translated address generated by LOAD REAL ADDRESS
- Page-table origin in INVALIDATE PAGE TABLE ENTRY
- Page-frame real address in page-table entry
- Trace-entry address in control register 12
- ASN-first-table origin in control register 14
- ASN-second-table origin in ASN-first-table entry
- Authority-table origin in ASN-second-table entry, except when used by access-register translation
- Linkage-table origin in control register 5 or primary ASN-second-table entry[1]
- Entry-table origin in linkage-table entry
- Dispatchable-unit-control-table origin in control register 2
- Primary-ASN-second-table-entry origin in control register 5[1]
- Base-ASN-second-table-entry origin and subspace-ASN-second-table-entry origin in dispatchable-unit control table
- ASN-second-table-entry address in entry-table entry and access-list entry
- PCF-entry-table origin at real locations 196-199

Permanently Assigned Real Addresses

- Address of the doubleword into which TEST PENDING INTERRUPTION stores when the second-operand address is zero
- Addresses of PSWs, interruption codes, and the associated information used during interruption
- Addresses used for machine-check logout and save areas
- Address of PCF-entry-table origin
- Address of STORE FACILITY LIST operand

Addresses which Are Unpredictably Real or Absolute

- Segment-table origin in control registers 1, 7, and 13 and in access-register-specified segment-table designation
- Page-table origin in segment-table entry
- Address of segment-table entry or page-table entry provided by LOAD REAL ADDRESS
- The dispatchable-unit or primary-space access-list origin and the authority-table origin (in the ASTE designated by the ALE used) used by access-register translation

*Figure 3-11 (Part 2 of 3). Handling of Addresses*



LzLabs GmbH. Ex. 1008-98

Absolute Addresses

- Prefix value
- Channel-program address in ORB
- Data address in CCW
- IDAW address in a CCW specifying indirect data addressing
- CCW address in a CCW specifying transfer in channel
- Data address in IDAW
- Measurement-block origin specified by SET CHANNEL MONITOR
- Address limit specified by SET ADDRESS LIMIT
- Addresses used by the store-status-at-address SIGNAL PROCESSOR order
- Failing-storage address stored in the word at real location 248
- CCW address in SCSW

Permanently Assigned Absolute Addresses

- Addresses used for the store-status function
- Addresses of PSW and first two CCWs used for initial program loading

Addresses Not Used to Reference Storage

- PER starting address in control register 10
- PER ending address in control register 11
- Address stored in the word at real location 156 for a monitor event
- Address in shift instructions and other instructions specified not to use the address to reference storage

**Explanation:**

[1] When the address-space-function (ASF) control, bit 15 of control register 0, is zero, control register 5 contains the linkage-table origin. When the ASF control is one, control register 5 contains the primary-ASN-second-table-entry origin, and the linkage-table origin is in the primary ASN-second-table entry.

*Figure 3-11 (Part 3 of 3). Handling of Addresses*

## Assigned Storage Locations

Figure 3-12 on page 3-50 shows the format and extent of the assigned locations in storage. The locations are used as follows.

0-7 (Absolute Address)

*Initial-Program-Loading PSW*: The first eight bytes read during the initial-program-loading (IPL) initial-read operation are stored at locations 0-7. The contents of these locations are used as the new PSW at the completion of the IPL operation. These locations may also be used for temporary storage at the initiation of the IPL operation.

0-7 (Real Address)

*Restart New PSW*: The new PSW is fetched from locations 0-7 during a restart interruption.

8-15 (Absolute Address)

*Initial-Program-Loading CCW1*: Bytes 8-15 read during the initial-program-loading (IPL) initial-read operation are stored at locations 8-15. The contents of these locations are ordinarily used as the next CCW in an IPL CCW chain after completion of the IPL initial-read operation.

8-15 (Real Address)

*Restart Old PSW*: The current PSW is stored as the old PSW at locations 8-15 during a restart interruption.

16-23 (Absolute Address)

*Initial-Program-Loading CCW2*: Bytes 16-23 read during the initial-program loading (IPL) initial-read operation are stored at locations 16-23. The contents of these locations may be used as


LzLabs GmbH. Ex. 1008-99

another CCW in the IPL CCW chain to follow IPL CCW1.

24-31 (Real Address)

*External Old PSW*: The current PSW is stored as the old PSW at locations 24-31 during an external interruption.

32-39 (Real Address)

*Supervisor-Call Old PSW*: The current PSW is stored as the old PSW at locations 32-39 during a supervisor-call interruption.

40-47 (Real Address)

*Program Old PSW*: The current PSW is stored as the old PSW at locations 40-47 during a program interruption.

48-55 (Real Address)

*Machine-Check Old PSW*: The current PSW is stored as the old PSW at locations 48-55 during a machine-check interruption.

56-63 (Real Address)

*Input/Output Old PSW*: The current PSW is stored as the old PSW at locations 56-63 during an I/O interruption.

88-95 (Real Address)

*External New PSW*: The new PSW is fetched from locations 88-95 during an external interruption.

96-103 (Real Address)

*Supervisor-Call New PSW*: The new PSW is fetched from locations 96-103 during a supervisor-call interruption.

104-111 (Real Address)

*Program New PSW*: The new PSW is fetched from locations 104-111 during a program interruption.

112-119 (Real Address)

*Machine-Check New PSW*: The new PSW is fetched from locations 112-119 during a machine-check interruption.

120-127 (Real Address)

*Input/Output New PSW*: The new PSW is fetched from locations 120-127 during an I/O interruption.

128-131 (Real Address)

*External-Interruption Parameter*: During an external interruption due to service signal or the external time reference (ETR), the parameter associated with the interruption is stored at locations 128-131.

132-133 (Real Address)

*CPU Address*: During an external interruption due to malfunction alert, emergency signal, or external call, the CPU address associated with the source of the interruption is stored at locations 132-133. The CPU address is a 16-bit unsigned binary integer. For all other external-interruption conditions, zeros are stored at locations 132-133.

134-135 (Real Address)

*External-Interruption Code*: During an external interruption, the interruption code is stored at locations 134-135.

136-139 (Real Address)

*Supervisor-Call-Interruption Identification*: During a supervisor-call interruption, the instruction-length code is stored in bit positions 5 and 6 of location 137, and the interruption code is stored at locations 138-139. Zeros are stored at location 136 and in the remaining bit positions of location 137.

140-143 (Real Address)

*Program-Interruption Identification*: During a program interruption, the instruction-length code is stored in bit positions 5 and 6 of location 141, and the interruption code is stored at locations 142-143. Zeros are stored at location 140 and in the remaining bit positions of location 141.

144-147 (Real Address)

*Data-Exception Code (DXC)*: If the basic-floating-point-extensions facility is installed, then, during a program interruption due to a data exception, the data-exception code is stored at location 147, and zeros are stored at locations 144-146. The DXC is described in



LzLabs GmbH. Ex. 1008-100

"Data-Exception Code (DXC)" on page 6-15.

*Translation-Exception Identification*: During a program interruption due to a segment-translation exception or a page-translation exception, the segment-index and page-index portion of the virtual address causing the exception is stored at locations 144-147. This address is sometimes referred to as the translation-exception address. Bits 20-28 of the address are unpredictable. If the exception was a page-translation exception that was recognized during the execution of MOVE PAGE (facility 2), bit 29 of the address is set to one. If the exception was a page-translation exception recognized during the execution of an instruction other than MOVE PAGE (facility 2), bit 29 is set to zero. If the exception was a segment-translation exception, bit 29 of the address is unpredictable. See the definition of real location 162 for related information.

Bits 30-31 of the address are set to identify the segment-table designation (STD) used in the translation, as follows:

| Bit 30 | Bit 31 | Meaning |
|---|---|---|
| 0 | 0 | Primary STD was used. |
| 0 | 1 | CPU was in the access-register mode, and either the access was an instruction fetch or it was a storage-operand reference that used an AR-specified STD (the access was not an implicit reference to the linkage stack). The exception access id, real location 160, can be examined to determine the STD used. However, if the primary, secondary, or home STD was used, bits 30 and 31 may be set to 00, 10, or 11, respectively, instead of to 01. |
| 1 | 0 | Secondary STD was used. |
| 1 | 1 | Home STD was used (includes the case of an implicit reference to the linkage stack). |

The CPU may avoid setting bits 30 and 31 to 01 by recognizing that the access was an instruction fetch, that access-list-entry token 00000000 or 00000001 hex was used, or that the access-list-entry token designated, through an access-list entry, an ASN-second-table entry containing an STD equal to the primary STD, secondary STD, or home STD.

Bit 0 of location 144 is set to one if the CPU was in either the primary-space mode or the secondary-space mode and the secondary STD was used; otherwise, bit 0 is set to zero.

During a program interruption due to an AFX-translation, ASX-translation, primary-authority, or secondary-authority exception, the ASN being translated is stored at locations 146-147. Zeros are stored at locations 144-145.

During a program interruption due to a space-switch event, an identification of the old instruction space is stored at locations 146-147, and the old instruction-space space-switch-event-control bit is placed in bit position 0 and zeros are placed in bit positions 1-15 of locations 144-145. The identification and bit stored are as follows:

- If the CPU was in the primary-space, secondary-space, or access-register mode before the operation, the old PASN, bits 16-31 of control register 4 before the operation, is stored at locations 146-147, and the old primary space-switch-event-control bit, bit 0 of control register 1 before the operation, is placed in bit position 0 of locations 144-145.
- If the CPU was in the home-space mode before the operation, zeros are stored at locations 146-147, and the home space-switch-event-control bit, bit 0 of control register 13, is placed in bit position 0 of locations 144-145.

During a program interruption due to an LX-translation or EX-translation exception recognized by PROGRAM CALL, the PC number is stored in bit positions


LzLabs GmbH. Ex. 1008-101

12-31 of the word at locations 144-147. Bits 0-11 are set to zeros.

During a program interruption due to an EX-translation exception recognized by PROGRAM CALL FAST, the PC number is stored in bit positions 12-31 of the word at locations 144-147. Bits 0-10 are set to zeros, and bit 11 is set to one.

If the suppression-on-protection facility is installed, then, during a program interruption due to a protection exception, information is stored at locations 144-147 as described in "Suppression on Protection" on page 3-12.

148-149 (Real Address)

*Monitor-Class Number*: During a program interruption due to a monitor event, the monitor-class number is stored at location 149, and zeros are stored at location 148.

150-151 (Real Address)

*PER Code*: During a program interruption due to a PER event with PER 1, the PER code is stored in bit positions 0-3 of locations 150-151, and zeros are stored in bit positions 4-15. With PER 2, the PER code is stored in bit positions 0-2 and 4 of locations 150-151, and other information is or may be stored as described in "Identification of Cause" on page 4-17.

152-155 (Real Address)

*PER Address*: During a program interruption due to a PER event, the PER address is stored at locations 152-155. Bit 0 of location 152 is set to zero.

156-159 (Real Address)

*Monitor Code*: During a program interruption due to a monitor event, the monitor code is stored at locations 156-159.

160 (Real Address)

*Exception Access Identification*: During a program interruption due to a segment-translation exception or a page-translation exception, an indication of the address space to which the exception applies may be stored at location 160. If the CPU was in the access-register mode and the access was an instruction fetch, including a fetch of the target of an EXECUTE instruction, zeros are stored at location 160. If the CPU was in the access-register mode and the access was a storage-operand reference that used an AR-specified segment-table designation, the number of the access register used is stored in bit positions 4-7 of location 160, and zeros are stored in bit positions 0-3. (In either of the two cases described so far, storing at location 160 occurs regardless of the value stored in bit positions 30 and 31 of real locations 144-147.) If the CPU was in the access-register mode but the access was an implicit reference to the linkage stack, or if the CPU was not in the access-register mode, the contents of location 160 are unpredictable.

During a program interruption due to an ALEN-translation, ALE-sequence, ASTE-validity, ASTE-sequence, or extended-authority exception recognized during access-register translation, the number of the access register used is stored in bit positions 4-7 of location 160, and zeros are stored in bit positions 0-3. During a program interruption due to an ASTE-validity or ASTE-sequence exception recognized during a subspace-replacement operation, all zeros are stored at location 160.

If the suppression-on-protection facility is installed, then, during a program interruption due to a protection exception, information is stored at location 160 as described in "Suppression on Protection" on page 3-12.

161 (Real Address)

*PER Access Identification*: During a program interruption due to a PER storage-alteration event, an indication of the address space to which the event applies may be stored at location 161. If the access used an AR-specified segment-table designation, the number of the access register used is stored in bit positions 4-7 of location 161, and zeros are stored in bit positions 0-3.



LzLabs GmbH. Ex. 1008-102

However, with PER 1, the contents of location 161 are unpredictable if the instruction that caused the event turned DAT off. Also, with PER 1 or PER 2, the contents of location 161 are unpredictable if (1) the CPU was in the access-register mode but the access was an implicit reference to the linkage stack, (2) the CPU was not in the access-register mode, or (3) bit 2 of the PER code is one but indicates a store-using-real-address event instead of a storage-alteration event.

162 (Real Address)

*Operand Access Identification*: During a program interruption due to a page-translation exception recognized by the MOVE PAGE (facility 2) instruction, the contents of the $R_1$ field of the instruction are stored in bit positions 0-3 of location 162, and the contents of the $R_2$ field are stored in bit positions 4-7. If the page-translation exception was recognized during the execution of an instruction other than MOVE PAGE (facility 2), or if a segment-translation exception was recognized, the contents of location 162 are unpredictable.

163 (Absolute Address)

*Store-Status Architectural-Mode Identification*: During the execution of the store-status operation if z/Architecture is installed, zeros are stored in bit positions 0-7 of location 163. A zero stored in bit position 7 indicates the ESA/390 architectural mode, and a one indicates the z/Architecture architectural mode.

163 (Real Address)

*Machine-Check Architectural-Mode Identification*: During a machine-check interruption if z/Architecture is installed, zeros are stored in bit positions 0-7 of location 163. A zero stored in bit position 7 indicates the ESA/390 architectural mode, and a one indicates the z/Architecture architectural mode.

184-187 (Real Address)

*Subsystem-Identification Word*: During an I/O interruption, the subsystem-identification word is stored at locations 184-187.

188-191 (Real Address)

*I/O-Interruption Parameter*: During an I/O interruption, the interruption parameter from the associated subchannel is stored at locations 188-191.

192-195 (Real Address)

*I/O-Interruption-Identification Word*: During an I/O interruption, the I/O-interruption-identification word, which further identifies the source of the I/O interruption, is stored at locations 192-195.

196-199 (Real Address)

*PCF-Entry-Table Origin*: The real origin of the PCF entry table is obtained by PROGRAM CALL FAST from real locations 196-199.

200-203 (Real Address)

*STFL Facility List*: The STORE FACILITY LIST instruction stores information at real locations 200-203. The information describes which facilities are provided by the CPU. See the definition of STORE FACILITY LIST in Chapter 10, "Control Instructions," for a description of the information stored.

212-215 (Absolute Address)

*Store-Status Extended-Save-Area Address*: During the execution of the store-status operation when the basic-floating-point-extensions facility is installed and the extended-save-area control, bit 2 of control register 14, is one, bits 1-19 of locations 212-215, with 12 zeros appended on the right, are used as the absolute address of a 4,096-byte extended save area. Bits 0 and 20-31 of the locations are reserved and should be zeros. They are ignored when forming the address of the extended save area. Bits 1-19 must not be all zeros; otherwise, storing is not performed in the extended save area.

212-215 (Real Address)

*Machine-Check Extended-Save-Area Address:* During a machine-check inter-


LzLabs GmbH. Ex. 1008-103

ruption when the basic-floating-point-extensions facility is installed and the extended-save-area control, bit 2 of control register 14, is one, bits 1-19 of locations 212-215, with 12 zeros appended on the right, are used as the absolute address of a 4,096-byte extended save area. Bits 0 and 20-31 of the locations are reserved and should be zeros. They are ignored when forming the address of the extended save area. Bits 1-19 must not be all zeros; otherwise, storing is not performed in the extended save area.

216-223 (Absolute Address)

*Store-Status CPU-Timer Save Area*: During the execution of the store-status operation, the contents of the CPU timer are stored at locations 216-223.

216-223 (Real Address)

*Machine-Check CPU-Timer Save Area*: During a machine-check interruption, the contents of the CPU timer are stored at locations 216-223.

224-231 (Absolute Address)

*Store-Status Clock-Comparator Save Area*: During the execution of the store-status operation, the contents of the clock comparator are stored at locations 224-231.

224-231 (Real Address)

*Machine-Check Clock-Comparator Save Area*: During a machine-check interruption, the contents of the clock comparator are stored at locations 224-231.

232-239 (Real Address)

*Machine-Check-Interruption Code*: During a machine-check interruption, the machine-check-interruption code is stored at locations 232-239.

244-247 (Real Address)

*External-Damage Code*: During a machine-check interruption due to certain external-damage conditions, depending on the model, an external-damage code may be stored at locations 244-247.

248-251 (Real Address)

*Failing-Storage Address*: During a machine-check interruption, a failing-storage address may be stored at locations 248-251. Bit 0 of location 248 is set to zero.

256-263 (Absolute Address)

*Store-Status PSW Save Area*: During the execution of the store-status operation, the contents of the current PSW are stored at locations 256-263.

256-271 (Real Address)

*Fixed-Logout Area*: Depending on the model, logout information may be stored at locations 256-271 during a machine-check interruption.

264-267 (Absolute Address)

*Store-Status Prefix Save Area*: During the execution of the store-status operation, the contents of the prefix register are stored at locations 264-267.

288-351 (Absolute Address)

*Store-Status Access-Register Save Area*: During the execution of the store-status operation, the contents of the access registers are stored at locations 288-351.

288-351 (Real Address)

*Machine-Check Access-Register Save Area*: During a machine-check interruption, the contents of the access registers are stored at locations 288-351.

352-383 (Absolute Address)

*Store-Status Floating-Point-Register Save Area*: During the execution of the store-status operation, the contents of floating-point registers 0, 2, 4, and 6 are stored at locations 352-383.

352-383 (Real Address)

*Machine-Check Floating-Point-Register Save Area*: During a machine-check interruption, the contents of floating-point registers 0, 2, 4, and 6 are stored at locations 352-383.

384-447 (Absolute Address)

*Store-Status General-Register Save Area*: During the execution of the store-



LzLabs GmbH. Ex. 1008-104

status operation, the contents of the general registers are stored at locations 384-447.

384-447 (Real Address)

*Machine-Check General-Register Save Area*: During a machine-check interruption, the contents of the general registers are stored at locations 384-447.

448-511 (Absolute Address)

*Store-Status Control-Register Save Area*: During the execution of the store-status operation, the contents of the control registers are stored at locations 448-511.

448-511 (Real Address)

*Machine-Check Control-Register Save Area*: During a machine-check interruption, the contents of the control registers are stored at locations 448-511.

**Programming Notes:**

1. When the CPU is in the access-register mode, some instructions, such as MVCL, which address operands in more than one address space, may cause a storage-alteration PER event in one address space concurrently with a segment-translation exception or a page-translation exception in another address space. The access registers used to cause these conditions in such a case are different. In order to identify both access registers, two access identifications, namely the exception access identification and the PER access identification, are provided.
2. STORE THEN AND SYSTEM MASK can cause a PER storage-alteration event and turn DAT off, in which case, with PER 1, the PER access identification at real location 161 is unpredictable.


LzLabs GmbH. Ex. 1008-105

| Hex | Dec | |
|---|---|---|
| 0 | 0 | Initial-Program-Loading PSW; or Restart New PSW |
| 4 | 4 | |
| 8 | 8 | Initial-Program-Loading CCW1; or Restart Old PSW |
| C | 12 | |
| 10 | 16 | Initial-Program Loading CCW2 |
| 14 | 20 | |
| 18 | 24 | External Old PSW |
| 1C | 28 | |
| 20 | 32 | Supervisor-Call Old PSW |
| 24 | 36 | |
| 28 | 40 | Program Old PSW |
| 2C | 44 | |
| 30 | 48 | Machine-Check Old PSW |
| 34 | 52 | |
| 38 | 56 | Input/Output Old PSW |
| 3C | 60 | |
| 40 | 64 | |
| 44 | 68 | |
| 48 | 72 | |
| 4C | 76 | |
| 50 | 80 | |
| 54 | 84 | |
| 58 | 88 | External New PSW |
| 5C | 92 | |
| 60 | 96 | Supervisor-Call New PSW |
| 64 | 100 | |
| 68 | 104 | Program New PSW |
| 6C | 108 | |
| 70 | 112 | Machine-Check New PSW |
| 74 | 116 | |
| 78 | 120 | Input/Output New PSW |
| 7C | 124 | |

*Figure 3-12 (Part 1 of 4). Assigned Storage Locations*



LzLabs GmbH. Ex. 1008-106

| Hex | Dec | | | | | |
|---|---|---|---|---|---|---|
| 80 | 128 | External-Interruption Parameter | | | | |
| 84 | 132 | CPU Address | | | External-Interruption Code | |
| 88 | 136 | 0 0 0 0 0 0 0 0 0 0 0 0 0 | ILC | 0 | SVC-Interruption Code | |
| 8C | 140 | 0 0 0 0 0 0 0 0 0 0 0 0 0 | ILC | 0 | Program-Interruption Code | |
| 90 | 144 | Data-Exception Code or Translation-Exception Identification | | | | |
| 94 | 148 | Monitor-Class Number | PER Cde | ATMID | SI | |
| 98 | 152 | PER Address | | | | |
| 9C | 156 | Monitor Code | | | | |
| A0 | 160 | Exc. Access ID | PER Access ID | Op. Access ID | SS/MC Ar-Md Id | |
| A4 | 164 | | | | | |
| A8 | 168 | | | | | |
| AC | 172 | | | | | |
| B0 | 176 | | | | | |
| B4 | 180 | | | | | |
| B8 | 184 | Subsystem-Identification Word | | | | |
| BC | 188 | I/O-Interruption Parameter | | | | |
| C0 | 192 | I/O-Interruption-Identification Word | | | | |
| C4 | 196 | PCF-Entry-Table Origin | | | | |
| C8 | 200 | STFL Facility List | | | | |
| CC | 204 | | | | | |
| D0 | 208 | | | | | |
| D4 | 212 | Store-Status Extended-Save-Area Address; or Machine-Check Extended-Save-Area Address | | | | |
| D8 | 216 | Store-Status CPU-Timer Save Area; or Machine-Check CPU-Timer Save Area | | | | |
| DC | 220 | | | | | |
| E0 | 224 | Store-Status Clock-Comparator Save Area; or Machine-Check Clock-Comparator Save Area | | | | |
| E4 | 228 | | | | | |
| E8 | 232 | Machine-Check Interruption Code | | | | |
| EC | 236 | | | | | |
| F0 | 240 | | | | | |
| F4 | 244 | External-Damage Code | | | | |
| F8 | 248 | Failing-Storage Address | | | | |
| FC | 252 | | | | | |

*Figure 3-12 (Part 2 of 4). Assigned Storage Locations*



LzLabs GmbH. Ex. 1008-107

| Hex | Dec | |
|---|---|---|
| 100 | 256 | Store-Status PSW Save Area; or Fixed-Logout Area (Part 1) |
| 104 | 260 | |
| 108 | 264 | Store-Status Prefix Save Area; or Fixed-Logout Area (Part 2) |
| 10C | 268 | Fixed-Logout Area (Part 3) |
| 110 | 272 | |
| | | / / |
| 11C | 284 | |
| 120 | 288 | Store-Status Access-Register Save Area; or Machine-Check Access-Register Save Area |
| 124 | 292 | |
| 128 | 296 | |
| 12C | 300 | |
| | | / / |
| 154 | 340 | |
| 158 | 344 | |
| 15C | 348 | |
| 160 | 352 | Store-Status Floating-Point-Register Save Area; or Machine-Check Floating-Point-Register Save Area |
| 164 | 356 | |
| 168 | 360 | |
| 16C | 364 | |
| 170 | 368 | |
| 174 | 372 | |
| 178 | 376 | |
| 17C | 380 | |
| 180 | 384 | Store-Status General-Register Save Area; or Machine-Check General-Register Save Area |
| 184 | 388 | |
| 188 | 392 | |
| 18C | 396 | |
| | | / / |
| 1B4 | 436 | |
| 1B8 | 440 | |
| 1BC | 444 | |

*Figure 3-12 (Part 3 of 4). Assigned Storage Locations*



LzLabs GmbH. Ex. 1008-108

| Hex | Dec | |
|---|---|---|
| 1C0 | 448 | Store-Status Control-Register Save Area; or Machine-Check Control-Register Save Area |
| 1C4 | 452 | |
| 1C8 | 456 | |
| 1CC | 460 | |
| | / | / |
| 1F4 | 500 | |
| 1F8 | 504 | |
| 1FC | 508 | |

*Figure 3-12 (Part 4 of 4). Assigned Storage Locations*


LzLabs GmbH. Ex. 1008-109

LzLabs GmbH. Ex. 1008-110

# Chapter 4. Control


Stopped, Operating, Load, and Check-Stop States . . . . 4-1
Stopped State . . . . 4-2
Operating State . . . . 4-2
Load State . . . . 4-2
Check-Stop State . . . . 4-3
Program-Status Word . . . . 4-3
Program-Status-Word Format . . . . 4-5
Control Registers . . . . 4-6
Tracing . . . . 4-10
Control-Register Allocation . . . . 4-10
Trace Entries . . . . 4-11
Operation . . . . 4-14
Program-Event Recording . . . . 4-14
Control-Register Allocation and Segment-Table Designation . . . . 4-15
Operation . . . . 4-16
Identification of Cause . . . . 4-17
Priority of Indication . . . . 4-20
Storage-Area Designation . . . . 4-21
PER Events . . . . 4-22
Successful Branching . . . . 4-22
Instruction Fetching . . . . 4-22
Storage Alteration . . . . 4-22
General-Register Alteration . . . . 4-23
Store Using Real Address . . . . 4-24
Indication of PER Events Concurrently with Other Interruption Conditions . . . 4-24
Timing . . . . 4-29
Time-of-Day Clock . . . . 4-29
Format . . . . 4-29
States . . . . 4-30
Changes in Clock State . . . . 4-31
Setting and Inspecting the Clock . . . . 4-31
TOD Programmable Register . . . . 4-32
TOD-Clock Synchronization . . . . 4-34
Clock Comparator . . . . 4-35
CPU Timer . . . . 4-35
Externally Initiated Functions . . . . 4-37
Resets . . . . 4-37
CPU Reset . . . . 4-40
Initial CPU Reset . . . . 4-41
Subsystem Reset . . . . 4-41
Clear Reset . . . . 4-42
Power-On Reset . . . . 4-42
Initial Program Loading . . . . 4-43
Store Status . . . . 4-43
Multiprocessing . . . . 4-44
Shared Main Storage . . . . 4-44
CPU-Address Identification . . . . 4-45
CPU Signaling and Response . . . . 4-45
Signal-Processor Orders . . . . 4-45
Conditions Determining Response . . . . 4-50
Conditions Precluding Interpretation of the Order Code . . . . 4-50
Status Bits . . . . 4-51


This chapter describes in detail the facilities for controlling, measuring, and recording the operation of one or more CPUs.

## Stopped, Operating, Load, and Check-Stop States

The stopped, operating, load, and check-stop states are four mutually exclusive states of the CPU. When the CPU is in the stopped state, instructions and interruptions, other than the restart interruption, are not executed. In the operating state, the CPU executes instructions and takes interruptions, subject to the control of the program-status word (PSW) and control registers, and in the manner specified by the setting of the operator-facility rate control. The CPU is in the load state during the initial-program-loading operation. The CPU enters the check-stop state only as the result of machine malfunctions.

A change between these four CPU states can be effected by use of the operator facilities or by acceptance of certain SIGNAL PROCESSOR orders addressed to that CPU. The states are not controlled or identified by bits in the PSW. The stopped, load, and check-stop states are indicated to the operator by means of the manual indicator, load indicator, and check-stop indicator, respectively. These three indicators are off when the CPU is in the operating state.

The CPU timer is updated when the CPU is in the operating state or the load state. The TOD clock is not affected by the state of any CPU.

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-111

## Stopped State

The CPU changes from the operating state to the stopped state by means of the stop function. The stop function is performed when:

- The stop key is activated while the CPU is in the operating state.
- The CPU accepts a stop or stop-and-store-status order specified by a SIGNAL PROCESSOR instruction addressed to this CPU while it is in the operating state.
- The CPU has finished the execution of a unit of operation initiated by performing the start function with the rate control set to the instruction-step position.

When the stop function is performed, the transition from the operating to the stopped state occurs at the end of the current unit of operation. When the wait-state bit of the PSW is one, the transition takes place immediately, provided no interruptions are pending for which the CPU is enabled. In the case of interruptible instructions, the amount of data processed in a unit of operation depends on the particular instruction and may depend on the model.

Before entering the stopped state by means of the stop function, all pending allowed interruptions occur while the CPU is still in the operating state. They cause the old PSW to be stored and the new PSW to be fetched before the stopped state is entered. While the CPU is in the stopped state, interruption conditions remain pending.

The CPU is also placed in the stopped state when:

- CPU reset is completed. However, when the reset operation is performed as part of initial program loading for this CPU, then the CPU is placed in the load state and does not necessarily enter the stopped state.
- An address comparison indicates equality and stopping on the match is specified.

The execution of resets is described in “Resets” on page 4-37, and address comparison is described in “Address-Compare Controls” on page 12-1.

If the CPU is in the stopped state when an INVALIDATE PAGE TABLE ENTRY instruction is executed on another CPU in the configuration, the clearing of TLB entries is completed before the CPU leaves the stopped state.

## Operating State

The CPU changes from the stopped state to the operating state by means of the start function or when a restart interruption (see Chapter 6, “Interruptions”) occurs.

The start function is performed if the CPU is in the stopped state and (1) the start key associated with that CPU is activated or (2) that CPU accepts the start order specified by a SIGNAL PROCESSOR instruction addressed to that CPU. The effect of performing the start function is unpredictable when the stopped state has been entered by means of a reset.

When the rate control is set to the process position and the start function is performed, the CPU starts operating at normal speed. When the rate control is set to the instruction-step position and the wait-state bit is zero, one instruction or, for interruptible instructions, one unit of operation is executed, and all pending allowed interruptions occur before the CPU returns to the stopped state. When the rate control is set to the instruction-step position and the wait-state bit is one, the start function does not cause an instruction to be executed, but all pending allowed interruptions occur before the CPU returns to the stopped state.

## Load State

The CPU enters the load state when the load-normal or load-clear key is activated. (See “Initial Program Loading” on page 4-43. See also “Initial Program Loading” on page 17-17.) If the initial-program-loading operation is completed successfully, the CPU changes from the load state to the operating state, provided the rate control is set to the process position; if the rate control is set to the instruction-step position, the CPU changes from the load state to the stopped state.



LzLabs GmbH. Ex. 1008-112

## Check-Stop State

The check-stop state, which the CPU enters on certain types of machine malfunction, is described in Chapter 11, "Machine-Check Handling." The CPU leaves the check-stop state when CPU reset is performed.

**Programming Notes:**

1. Except for the relationship between execution time and real time, the execution of a program is not affected by stopping the CPU.
2. When, because of a machine malfunction, the CPU is unable to end the execution of an instruction, the stop function is ineffective, and a reset function has to be invoked instead. A similar situation occurs when an unending string of interruptions results from a PSW with a PSW-format error of the type that is recognized early, or from a persistent interruption condition, such as one due to the CPU timer.
3. Pending I/O operations may be initiated, and active I/O operations continue to suspension or completion, after the CPU enters the stopped state. The interruption conditions due to suspension or completion of I/O operations remain pending when the CPU is in the stopped state.

## Program-Status Word

The current program-status word (PSW) in the CPU contains information required for the execution of the currently active program. The PSW is 64 bits in length and includes the instruction address, condition code, and other control fields. In general, the PSW is used to control instruction sequencing and to hold and indicate much of the status of the CPU in relation to the program currently being executed. Additional control and status information is contained in control registers and permanently assigned storage locations.

The status of the CPU can be changed by loading a new PSW or part of a PSW.

Control is switched during an interruption of the CPU by storing the current PSW, so as to preserve the status of the CPU, and then loading a new PSW.

Execution of LOAD PSW, or the successful conclusion of the initial-program-loading sequence, introduces a new PSW. The instruction address is updated by sequential instruction execution and replaced by successful branches. Other instructions are provided which operate on a portion of the PSW. Figure 4-1 on page 4-4 summarizes these instructions.

A new or modified PSW becomes active (that is, the information introduced into the current PSW assumes control over the CPU) when the interruption or the execution of an instruction that changes the PSW is completed. The interruption for PER associated with an instruction that changes the PSW occurs under control of the PER mask that is effective at the beginning of the operation.

Bits 0-7 of the PSW are collectively referred to as the system mask.


LzLabs GmbH. Ex. 1008-113

| Instruction | System Mask (PSW Bits 0-7) | | PSW Key (PSW Bits 8-11) | | Problem State (PSW Bit 15) | | Address-Space Control (PSW Bits 16-17) | | Condition Code and Program Mask (PSW Bits 18-23) | | Addressing Mode (PSW Bit 32) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Saved | Set | Saved | Set | Saved | Set | Saved | Set | Saved | Set | Saved | Set |
| BRANCH AND LINK | - | - | - | - | - | - | - | - | AM | - | AM | - |
| BRANCH AND SAVE | - | - | - | - | - | - | - | - | - | - | Yes | - |
| BRANCH AND SAVE AND SET MODE | - | - | - | - | - | - | - | - | - | - | Yes | Yes[1] |
| BRANCH AND SET AUTHORITY | - | - | Yes | Yes | Yes | Yes | - | - | - | - | Yes | Yes |
| BRANCH AND SET MODE | - | - | - | - | - | - | - | - | - | - | Yes[1] | Yes[1] |
| BRANCH AND STACK | Yes | - | Yes | - | Yes | - | Yes | - | Yes | - | Yes[2] | - |
| BRANCH IN SUBSPACE GROUP | - | - | - | - | - | - | - | - | - | - | Yes[1] | Yes |
| BRANCH RELATIVE AND SAVE | - | - | - | - | - | - | - | - | - | - | Yes | - |
| BRANCH RELATIVE AND SAVE LONG | - | - | - | - | - | - | - | - | - | - | Yes | - |
| EXTRACT PSW | Yes | - | Yes | - | Yes | - | Yes | - | Yes | - | Yes | - |
| INSERT PROGRAM MASK | - | - | - | - | - | - | - | - | Yes | - | - | - |
| INSERT PSW KEY | - | - | Yes | - | - | - | - | - | - | - | - | - |
| INSERT ADDRESS SPACE CONTROL | - | - | - | - | - | - | Yes | - | - | - | - | - |
| Basic PROGRAM CALL | - | - | - | - | Yes | Yes | - | - | - | - | Yes | Yes |
| Stacking PROGRAM CALL | Yes | - | Yes | PKC | Yes | Yes | Yes | Yes | Yes | - | Yes | Yes |
| PROGRAM CALL FAST | Yes | - | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| PROGRAM RETURN | - | Yes[3] | - | Yes | - | Yes | - | Yes | - | Yes[4] | - | Yes |
| PROGRAM TRANSFER | - | - | - | - | - | Yes[5] | - | - | - | - | - | Yes |
| RESUME PROGRAM | - | - | - | - | - | - | - | Yes | - | Yes | - | Yes |
| SET ADDRESS SPACE CONTROL | - | - | - | - | - | - | - | Yes | - | - | - | - |
| SET PROGRAM MASK | - | - | - | - | - | - | - | - | - | Yes | - | - |
| SET PSW KEY FROM ADDRESS | - | - | - | Yes | - | - | - | - | - | - | - | - |
| SET SYSTEM MASK | - | Yes | - | - | - | - | - | - | - | - | - | - |
| STORE THEN AND SYSTEM MASK | Yes | ANDs | - | - | - | - | - | - | - | - | - | - |
| STORE THEN OR SYSTEM MASK | Yes | ORs | - | - | - | - | - | - | - | - | - | - |
| TRAP | - | - | - | - | Yes | - | Yes | Yes | Yes | - | Yes | Yes |

**Explanation:**

- No.

1 The action takes place only if the associated R field in the instruction is nonzero.

2 The action takes place only if the associated R field in the instruction is zero.

3 PROGRAM RETURN does not change the PER mask.

4 The condition code set by PROGRAM RETURN is unpredictable.

5 PROGRAM TRANSFER does not change the problem-state bit from one to zero.

AM The action depends on the addressing mode, bit 32 of the current PSW. In the 24-bit addressing mode, the condition code and program mask are saved in the leftmost byte of the general register. In the 31-bit addressing mode, the addressing mode and bits 1-7 of the 31-bit address replace the leftmost byte of the register.

*Figure 4-1 (Part 1 of 2). Operations on PSW Fields*



LzLabs GmbH. Ex. 1008-114

**Explanation** (Continued):

ANDs The logical AND of the immediate field in the instruction and the current system mask replaces the current system mask.

ORs The logical OR of the immediate field in the instruction and the current system mask replaces the current system mask.

PKC When the PSW-key-control bit, bit 131 of the 32-byte entry-table entry, is zero, the PSW key remains unchanged. When the PSW-key-control bit is one, the PSW key is set with the entry key, bits 136-139 of the entry-table entry.

*Figure 4-1 (Part 2 of 2). Operations on PSW Fields*

**Programming Note:** A summary of the operations which save or set the problem state, addressing mode, and instruction address is contained in "Subroutine Linkage without the Linkage Stack" on page 5-10.

## Program-Status-Word Format

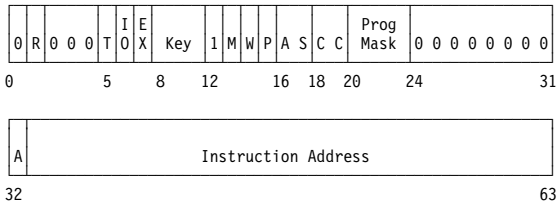



*Figure 4-2. PSW Format*

The following is a summary of the functions of the PSW fields. (See Figure 4-2.)

***PER Mask (R):*** Bit 1 controls whether the CPU is enabled for interruptions associated with program-event recording (PER). When the bit is zero, no PER event can cause an interruption. When the bit is one, interruptions are permitted, subject to the PER-event-mask bits in control register 9.

***DAT Mode (T):*** Bit 5 controls whether implicit dynamic address translation of logical and instruction addresses used to access storage takes place. When the bit is zero, DAT is off, and logical and instruction addresses are treated as real addresses. When the bit is one, DAT is on, and the dynamic-address-translation mechanism is invoked.

***I/O Mask (IO):*** Bit 6 controls whether the CPU is enabled for I/O interruptions. When the bit is zero, an I/O interruption cannot occur. When the bit is one, I/O interruptions are subject to the I/O-interruption subclass-mask bits in control register 6. When an I/O-interruption subclass-mask bit is zero, an I/O interruption for that I/O-interruption subclass cannot occur; when the I/O-interruption subclass-mask bit is one, an I/O interruption for that I/O-interruption subclass can occur.

***External Mask (EX):*** Bit 7 controls whether the CPU is enabled for interruption by conditions included in the external class. When the bit is zero, an external interruption cannot occur. When the bit is one, an external interruption is subject to the corresponding external subclass-mask bits in control register 0; when the subclass-mask bit is zero, conditions associated with the subclass cannot cause an interruption; when the subclass-mask bit is one, an interruption in that subclass can occur.

***PSW Key:*** Bits 8-11 form the access key for storage references by the CPU. If the reference is subject to key-controlled protection, the PSW key is matched with a storage key when information is stored or when information is fetched from a location that is protected against fetching. However, for one of the operands of each of MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH KEY, MOVE WITH SOURCE KEY, and MOVE WITH DESTINATION KEY, an access key specified as an operand is used instead of the PSW key.

***Machine-Check Mask (M):*** Bit 13 controls whether the CPU is enabled for interruption by machine-check conditions. When the bit is zero, a machine-check interruption cannot occur. When the bit is one, machine-check interruptions due to system damage and instruction-processing damage are permitted, but interruptions due to other machine-check-subclass conditions are subject to the subclass-mask bits in control register 14.


LzLabs GmbH. Ex. 1008-115

***Wait State (W):*** When bit 14 is one, the CPU is waiting; that is, no instructions are processed by the CPU, but interruptions may take place. When bit 14 is zero, instruction fetching and execution occur in the normal manner. The wait indicator is on when the bit is one.

***Problem State (P):*** When bit 15 is one, the CPU is in the problem state. When bit 15 is zero, the CPU is in the supervisor state. In the supervisor state, all instructions are valid. In the problem state, only those instructions are valid that provide meaningful information to the problem program and that cannot affect system integrity; such instructions are called unprivileged instructions. The instructions that are never valid in the problem state are called privileged instructions. When a CPU in the problem state attempts to execute a privileged instruction, a privileged-operation exception is recognized. Another group of instructions, called semiprivileged instructions, are executed by a CPU in the problem state only if specific authority tests are met; otherwise, a privileged-operation exception or a special-operation exception is recognized.

***Address-Space Control (AS):*** Bits 16 and 17, in conjunction with PSW bit 5, control the translation mode. See "Translation Modes" on page 3-28.

***Condition Code (CC):*** Bits 18 and 19 are the two bits of the condition code. The condition code is set to 0, 1, 2, or 3, depending on the result obtained in executing certain instructions. Most arithmetic and logical operations, as well as some other operations, set the condition code. The instruction BRANCH ON CONDITION can specify any selection of the condition-code values as a criterion for branching. A table in Appendix C summarizes the condition-code values that may be set for all instructions which set the condition code of the PSW.

***Program Mask:*** Bits 20-23 are the four program-mask bits. Each bit is associated with a program exception, as follows:

| Program-Mask Bit | Program Exception |
|---|---|
| 20 | Fixed-point overflow |
| 21 | Decimal overflow |
| 22 | HFP exponent underflow |
| 23 | HFP significance |

When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The setting of the HFP-exponent-underflow-mask bit or the HFP-significance-mask bit also determines the manner in which the operation is completed when the corresponding exception occurs.

***Addressing Mode (A):*** Bit 32 controls the size of effective addresses and effective-address generation. When the bit is zero, 24-bit addressing is specified. When the bit is one, 31-bit addressing is specified. The addressing mode does not control the size of PER addresses or of addresses used to access DAT, ASN, dispatchable-unit-control, linkage, entry, and trace tables or access lists or the linkage stack. See "Address Generation" on page 5-7 and "Address Size and Wraparound" on page 3-5.

***Instruction Address:*** Bits 33-63 form the instruction address. This address designates the location of the leftmost byte of the next instruction to be executed, unless the CPU is in the wait state (bit 14 of the PSW is one).

Bit positions 0, 2-4, and 24-31 are unassigned and must contain zeros. A specification exception is recognized when these bit positions do not contain zeros. When bit 32 of the PSW specifies the 24-bit addressing mode, bits 33-39 of the instruction address must be zeros; otherwise, a specification exception is recognized. A specification exception is also recognized when bit position 12 does not contain a one.

## Control Registers

The control registers provide for maintaining and manipulating control information outside the PSW. There are sixteen 32-bit control registers.

All control-register bit positions in all 16 control registers are installed, regardless of whether the bit position is assigned to a facility. One or more specific bit positions in control registers are assigned to each facility requiring such register space.

The LOAD CONTROL instruction causes all control-register bit positions within those registers designated by the instruction to be loaded from storage. The instructions BRANCH AND SET AUTHORITY, BRANCH IN SUBSPACE GROUP,



LzLabs GmbH. Ex. 1008-116

LOAD ADDRESS SPACE PARAMETERS, SET SECONDARY ASN, BRANCH AND STACK, PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, and PROGRAM TRANSFER provide specialized functions to place information into certain control-register bit positions.

Information loaded into the control registers becomes active (that is, assumes control over the system) at the completion of the instruction that causes the information to be loaded.

At the time the registers are loaded, the information is not checked for exceptions, such as invalid translation-format code or an address designating an unavailable or protected location. The validity of the information is checked and the exceptions, if any, are indicated at the time the information is used.

The STORE CONTROL instruction causes the contents of all control-register bit positions, within those registers designated by the instruction, to be placed in storage. The instructions EXTRACT PRIMARY ASN, EXTRACT SECONDARY ASN, and PROGRAM CALL provide specialized functions to obtain information from certain control-register bit positions.

Only the general structure of the control registers is described here; the definition of a particular control-register bit position appears in the description of the facility with which the position is associated. Figure 4-3 on page 4-8 shows the control-register bit positions which are assigned and the initial values of the positions upon execution of initial CPU reset. All control-register bit positions not listed in the figure are initialized to zero.

**Programming Notes:**

1. The detailed definition of a particular control-register bit position can be located by referring to the entry “control-register assignment” in the Index.
2. To ensure that existing programs operate correctly if and when new facilities using additional control-register positions are installed, the program should load zeros in unassigned control-register positions.


LzLabs GmbH. Ex. 1008-117

| Ctrl Reg | Bits | Name of Field | Associated with | Initial Value |
|---|---|---|---|---|
| 0 | 1 | SSM-suppression control | SET SYSTEM MASK | 0 |
| 0 | 2 | TOD-clock-sync control | TOD clock | 0 |
| 0 | 3 | Low-address-protection control | Low-address protection | 0 |
| 0 | 4 | Extraction-authority control | Instruction authorization | 0 |
| 0 | 5 | Secondary-space control | Instruction authorization | 0 |
| 0 | 6 | Fetch-protection-override control | Key-controlled protection | 0 |
| 0 | 7 | Storage-protection-override control | Key-controlled protection | 0 |
| 0 | 8-12 | Translation format | Dynamic address translation | 0 |
| 0 | 13 | AFP-register control | Floating point | 0 |
| 0 | 14 | Vector control[1] | Vector operations | 0 |
| 0 | 15 | Address-space-function control | Instruction authorization | 0 |
| 0 | 16 | Malfunction-alert subclass mask | External interruptions | 0 |
| 0 | 17 | Emergency-signal subclass mask | External interruptions | 0 |
| 0 | 18 | External-call subclass mask | External interruptions | 0 |
| 0 | 19 | TOD-clock sync-check subclass mask | External interruptions | 0 |
| 0 | 20 | Clock-comparator subclass mask | External interruptions | 0 |
| 0 | 21 | CPU-timer subclass mask | External interruptions | 0 |
| 0 | 22 | Service-signal subclass mask | External interruptions | 0 |
| 0 | 24 | Unused[2] | | 1 |
| 0 | 25 | Interrupt-key subclass mask | External interruptions | 1 |
| 0 | 26 | Unused[2] | | 1 |
| 0 | 27 | ETR subclass mask | External interruptions | 0 |
| 0 | 28 | Program-call-fast control | PROGRAM CALL FAST | 0 |
| 0 | 29 | Crypto control | Cryptography | 0 |
| 1 | 0 | Primary space-switch-event control | Program interruptions | 0 |
| 1 | 1-19 | Primary segment-table origin | Dynamic address translation | 0 |
| 1 | 22 | Primary subspace-group control | Subspace groups | 0 |
| 1 | 23 | Primary private-space control | Dynamic address translation | 0 |
| 1 | 24 | Primary storage-alteration-event control | Program-event rec. 2 only | 0 |
| 1 | 25-31 | Primary segment-table length | Dynamic address translation | 0 |
| 2 | 1-25 | Dispatchable-unit-control-table origin | Access-register translation | 0 |
| 3 | 0-15 | PSW-key mask | Instruction authorization | 0 |
| 3 | 16-31 | Secondary ASN | Address spaces | 0 |
| 4 | 0-15 | Authorization index | Instruction authorization | 0 |
| 4 | 16-31 | Primary ASN | Address spaces | 0 |
| 5 | 0 | Subsystem-linkage control[3] | Instruction authorization | 0 |
| 5 | 1-24 | Linkage-table origin[3] | PC-number translation | 0 |
| 5 | 25-31 | Linkage-table length[3] | PC-number translation | 0 |
| 5 | 1-25 | Primary-ASN-second-table-entry origin[4] | Access-register translation | 0 |

*Figure 4-3 (Part 1 of 3). Assignment of Control-Register Fields*



LzLabs GmbH. Ex. 1008-118

| Ctrl Reg | Bits | Name of Field | Associated with | Initial Value |
|---|---|---|---|---|
| 6 | 0-7 | I/O-interruption subclass mask | I/O interruptions | 0 |
| 7 | 1-19 | Secondary segment-table origin | Dynamic address translation | 0 |
| 7 | 22 | Secondary subspace-group control | Subspace groups | 0 |
| 7 | 23 | Secondary private-space control | Dynamic address translation | 0 |
| 7 | 24 | Secondary storage-alteration-event control | Program-event rec. 2 only | 0 |
| 7 | 25-31 | Secondary segment-table length | Dynamic address translation | 0 |
| 8 | 0-15 | Extended authorization index | Access-register translation | 0 |
| 8 | 16-31 | Monitor masks | MONITOR CALL | 0 |
| 9 | 0 | Successful-branching-event mask | Program-event recording | 0 |
| 9 | 1 | Instruction-fetching-event mask | Program-event recording | 0 |
| 9 | 2 | Storage-alteration-event mask | Program-event recording | 0 |
| 9 | 3 | GR-alteration-event mask | Program-event rec. 1 only | 0 |
| 9 | 4 | Store-using-real-address-event mask | Program-event recording | 0 |
| 9 | 8 | Branch-address control | Program-event rec. 2 only | 0 |
| 9 | 10 | Storage-alteration-space control | Program-event rec. 2 only | 0 |
| 9 | 16-31 | PER general-register masks | Program-event rec. 1 only | 0 |
| 10 | 1-31 | PER starting address | Program-event recording | 0 |
| 11 | 1-31 | PER ending address | Program-event recording | 0 |
| 12 | 0 | Branch-trace control | Tracing | 0 |
| 12 | 1-29 | Trace-entry address | Tracing | 0 |
| 12 | 30 | ASN-trace control | Tracing | 0 |
| 12 | 31 | Explicit-trace control | Tracing | 0 |
| 13 | 0 | Home space-switch-event control | Program interruptions | 0 |
| 13 | 1-19 | Home segment-table origin | Dynamic address translation | 0 |
| 13 | 22 | Ignored | | 0 |
| 13 | 23 | Home private-space control | Dynamic address translation | 0 |
| 13 | 24 | Home storage-alteration-event control | Program-event rec. 2 only | 0 |
| 13 | 25-31 | Home segment-table length | Dynamic address translation | 0 |
| 14 | 0 | Unused[2] | | 1 |
| 14 | 1 | Unused[2] | | 1 |
| 14 | 2 | Extended-save-area control | Floating point | 0 |
| 14 | 3 | Channel-report-pending subclass mask | I/O machine-check handling | 0 |
| 14 | 4 | Recovery subclass mask | Machine-check handling | 0 |
| 14 | 5 | Degradation subclass mask | Machine-check handling | 0 |
| 14 | 6 | External-damage subclass mask | Machine-check handling | 1 |
| 14 | 7 | Warning subclass mask | Machine-check handling | 0 |
| 14 | 10 | TOD-clock-control-override control | TOD clock | 0 |
| 14 | 12 | ASN-translation control | Instruction authorization | 0 |
| 14 | 13-31 | ASN-first-table origin | ASN translation | 0 |
| 15 | 1-28 | Linkage-stack-entry address | Linkage-stack operations | 0 |

*Figure 4-3 (Part 2 of 3). Assignment of Control-Register Fields*


LzLabs GmbH. Ex. 1008-119

**Explanation:**

The fields not listed are unassigned. The initial value for all unlisted control-register positions is zero.

1 Bit 14 of control register 0, the vector-control bit, is described in the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207.

2 This bit is not used but is initialized to one for consistency with the System/370 definition.

3 When the address-space-function control in control register 0 is zero, LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, and PROGRAM TRANSFER treat control register 5 as containing the linkage-table designation (LTD) (subsystem-linkage control, linkage-table origin, and linkage-table length).

4 When the address-space-function control is one, control register 5 is treated as containing the primary-ASN-second-table-entry (PASTE) origin, and PROGRAM CALL and PROGRAM TRANSFER obtain the LTD from the PASTE.

*Figure 4-3 (Part 3 of 3). Assignment of Control-Register Fields*

## Tracing

Tracing assists in the determination of system problems by providing an ongoing record in storage of significant events. Tracing consists of three separately controllable functions which cause entries to be made in a trace table: branch tracing, ASN tracing, and explicit tracing. Branch tracing and ASN tracing together are referred to as implicit tracing.

When branch tracing is on, an entry is made in the trace table for each execution of certain branch instructions when they cause branching. The branch address is placed in the trace entry. The trace entry also indicates the addressing mode in effect after branching. The branch instructions that are traced are:

- BRANCH AND LINK (BALR only) when the $R_2$ field is not zero
- BRANCH AND SAVE (BASR only) when the $R_2$ field is not zero
- BRANCH AND SAVE AND SET MODE when the $R_2$ field is not zero
- BRANCH AND SET AUTHORITY
- BRANCH AND STACK when the $R_2$ field is not zero
- BRANCH IN SUBSPACE GROUP
- RESUME PROGRAM
- TRAP

However, a branch trace entry is made for BRANCH IN SUBSPACE GROUP only if ASN tracing is not on.

When ASN tracing is on, an entry is made in the trace table for each execution of the following instructions:

- BRANCH IN SUBSPACE GROUP
- PROGRAM CALL
- PROGRAM RETURN
- PROGRAM TRANSFER
- SET SECONDARY ASN

However, the entry for PROGRAM RETURN is made only when PROGRAM RETURN unstacks a linkage-stack state entry that was formed by PROGRAM CALL or PROGRAM CALL FAST, not when PROGRAM RETURN unstacks an entry formed by BRANCH AND STACK.

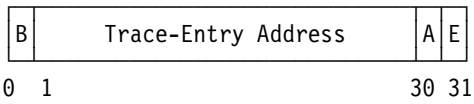
When explicit tracing is on, execution of TRACE causes an entry to be made in the trace table. This entry includes bits 16-63 from the TOD clock, the second operand of the TRACE instruction, and the contents of a range of general registers.

## Control-Register Allocation

The information to control tracing is contained in control register 12 and has the following format:



LzLabs GmbH. Ex. 1008-120




***Branch-Trace-Control Bit (B):*** Bit 0 of control register 12 controls whether branch tracing is turned on or off. If the bit is zero, branch tracing is off; if the bit is one, branch tracing is on.

***Trace-Entry Address:*** Bits 1-29 of control register 12, with two zero bits appended on the right, form the real address of the next trace entry to be made.

***ASN-Trace-Control Bit (A):*** Bit 30 of control register 12 controls whether ASN tracing is turned on or off. If the bit is zero, ASN tracing is off; if the bit is one, ASN tracing is on.

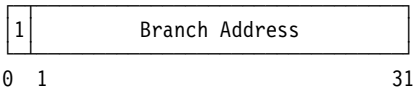
***Explicit-Trace-Control Bit (E):*** Bit 31 of control register 12 controls whether explicit tracing is turned on or off. If the bit is zero, explicit tracing is off, which causes the TRACE instruction to be executed as a no-operation; if the bit is one, the execution of the TRACE instruction creates an entry in the trace table, except that no entry is made when bit 0 of the second operand of the TRACE instruction is one.

## Trace Entries

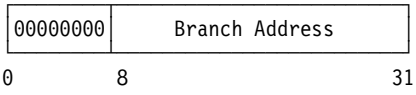
Trace entries are of eight types, as shown in Figure 4-4 on page 4-12.


LzLabs GmbH. Ex. 1008-121

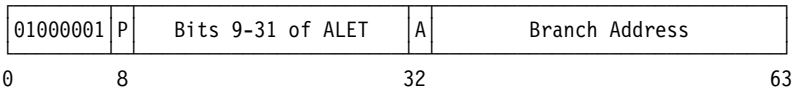
31-Bit Branch



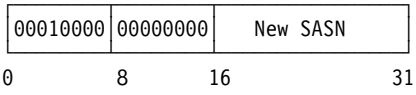

24-Bit Branch



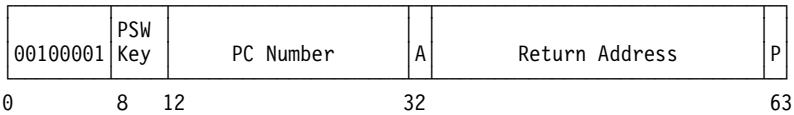

BRANCH IN SUBSPACE GROUP (if ASN Tracing Is On)



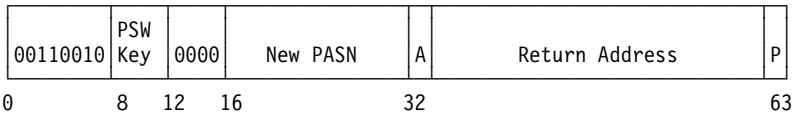

SET SECONDARY ASN

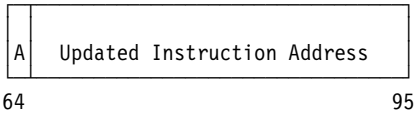



PROGRAM CALL




PROGRAM RETURN




PROGRAM TRANSFER

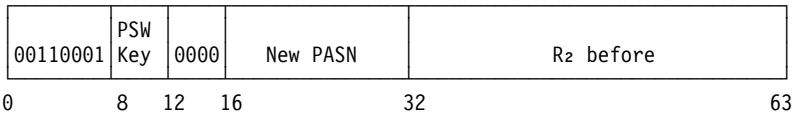
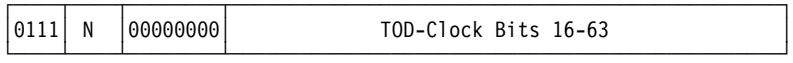



*Figure 4-4 (Part 1 of 2). Trace-Entry Formats*

TRACE



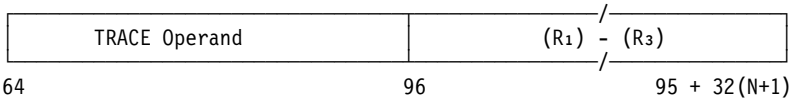




*Figure 4-4 (Part 2 of 2). Trace-Entry Formats*



LzLabs GmbH. Ex. 1008-122

***Branch Address:*** The branch address is the address of the next instruction to be executed when the branch is taken. In a branch trace entry when the 31-bit addressing mode is in effect after branching, bit positions 1-31 of the trace entry contain the branch address. When the 24-bit addressing mode is in effect after branching, bit positions 8-31 contain the branch address. In a trace entry made on execution of BRANCH IN SUBSPACE GROUP when ASN tracing is on, bit positions 33-63 of the trace entry contain the branch address.

***Primary-List Bit (P) and Bits 9-31 of ALET:*** Bit position 8 of the trace entry made on execution of BRANCH IN SUBSPACE GROUP when ASN tracing is on contains bit 7 of the access-list-entry token (ALET) in the access register designated by the $R_2$ field of the instruction. Bit positions 9-31 of the trace entry contain bits 9-31 of the ALET.

***New SASN:*** Bit positions 16-31 of the trace entry for SET SECONDARY ASN contain the ASN value loaded into control register 3 by the instruction.

***PSW Key:*** Bit positions 8-11 of the trace entries made on execution of PROGRAM CALL, PROGRAM RETURN, and PROGRAM TRANSFER contain the PSW key from the current PSW.

***PC Number:*** Bit positions 12-31 of the trace entry made on execution of PROGRAM CALL contain the value of the rightmost 20 bits of the second-operand address.

***Addressing-Mode Bit (A):*** Bit position 32 of the trace entry made on execution of PROGRAM CALL contains the addressing-mode bit from the current PSW. Bit position 32 of the trace entry made on execution of PROGRAM RETURN contains the addressing-mode bit that replaces bit 32 of the PSW, and bit position 64 of the trace entry contains bit 32 from the PSW before bit 32 is replaced. Bit position 32 of the trace entry made on execution of BRANCH IN SUBSPACE GROUP when ASN tracing is on contains the addressing-mode bit that replaces bit 32 of the PSW.

***Return Address:*** Bit positions 33-62 of the trace entry made on execution of PROGRAM CALL contain bits 1-30 of the updated instruction address in the PSW before that address is replaced from the entry-table entry. Bit positions 33-62 of the trace entry made on execution of PROGRAM RETURN contain bits 1-30 of the instruction address that replaces bits 33-63 of the PSW.

***Problem-State Bit (P):*** Bit position 63 of the trace entry made on execution of PROGRAM CALL contains the problem-state bit from the current PSW. Bit position 63 of the trace entry made on execution of PROGRAM RETURN contains the problem-state bit that replaces bit 15 of the PSW.

***New PASN:*** Bit positions 16-31 of the trace entry made on execution of PROGRAM RETURN contain the new PASN that is restored from the linkage-stack state entry. Bit positions 16-31 of the trace entry made on execution of PROGRAM TRANSFER contain the new PASN (which may be zero) specified in bit positions 16-31 of general register $R_1$.

***Updated Instruction Address:*** Bit positions 65-95 of the trace entry made on execution of PROGRAM RETURN contain bits 1-31 of the updated instruction address in the PSW before that address is replaced from the linkage-stack state entry.

***$R_2$ before:*** Bit positions 32-63 of the trace entry made on execution of PROGRAM TRANSFER contain the contents of the general register designated by the $R_2$ field of the instruction. Bits 0-30 of the general register designated by the $R_2$ field replace bits 32-62 of the PSW. Bit 31 of the same general register replaces the problem-state bit of the PSW.

***Number of Registers (N):*** Bits 4-7 of the trace entry for TRACE contain a value which is one less than the number of general registers which have been provided in the trace entry. The value of N ranges from zero, meaning the contents of one general register are provided in the trace entry, to 15, meaning the contents of all 16 general registers are provided.

***TOD-Clock Bits 16-63:*** Bits 16-63 of the trace entry for TRACE are obtained from bit positions 16-63 of the TOD clock, as would be provided by a STORE CLOCK instruction executed at the time the TRACE instruction was executed.


LzLabs GmbH. Ex. 1008-123

***TRACE Operand:*** Bits 64-95 of the trace entry for TRACE contain a copy of the 32 bits of the second operand of the TRACE instruction for which the entry is made.

***($R_1$) - ($R_3$):*** The four-byte fields starting with bit 96 of the trace entry for TRACE contain the contents of the general registers whose range is specified by the $R_1$ and $R_3$ fields of the TRACE instruction. The general registers are stored in ascending order of register numbers, starting with general register $R_1$ and continuing up to and including general register $R_3$, with general register 0 following general register 15.

**Programming Note:** The size of the trace entry for TRACE in units of words is 3 + (N + 1). The maximum size of an entry is 19 words, or 76 bytes.

## Operation

When an instruction which is subject to tracing is executed and the corresponding tracing function is turned on, a trace entry of the appropriate format is made. The real address of the trace entry is formed by appending two zero bits on the right to the value in bit positions 1-29 of control register 12. The address in control register 12 is subsequently increased by the size of the entry created.

No trace entry is stored if the incrementing of the address in control register 12 would cause a carry to be propagated into bit position 19 (that is, if the trace-entry address would be in the next 4K-byte block). If this would be the case for the entry to be made, a trace-table exception is recognized. For the purpose of recognizing the trace-table exception in the case of a TRACE instruction, the maximum length of 76 bytes is used instead of the actual length.

The storing of a trace entry is not subject to key-controlled protection (nor, since the trace-entry address is real, is it subject to access-list-controlled protection or page protection), but it is subject to low-address protection; that is, if the address of the trace entry due to be created is in the range 0-511 and bit 3 of control register 0 is one, a protection exception is recognized, and instruction execution is suppressed. If the address of a trace entry is invalid, an addressing exception is recognized, and instruction execution is suppressed.

The three exceptions associated with storing a trace entry (addressing, protection, and trace table) are collectively referred to as trace exceptions.

If a program interruption takes place for a condition which is not a trace-exception condition and for which execution of an instruction is not completed, it is unpredictable whether part or all of any trace entry due to be made for such an interrupted instruction is stored in the trace table. Thus, for a condition which would ordinarily cause nullification or suppression of instruction execution, storage locations may have been altered beginning at the location designated by control register 12 and extending up to the length of the entry that would have been created.

When PROGRAM RETURN unstacks a linkage-stack state entry that was formed by BRANCH AND STACK and ASN tracing is on, trace exceptions may be recognized, even though a trace entry is not made and no part of a trace entry is stored.

The order in which information is placed in a trace entry is unpredictable. Furthermore, as observed by other CPUs and by channel programs, the contents of a byte of a trace entry may appear to change more than once before completion of the instruction for which the entry is made.

The trace-entry address in control register 12 is updated only on completion of execution of an instruction for which a trace entry is made.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

## Program-Event Recording

There are two versions of the program-event-recording (PER) facility. The version which is the same as PER in ESA/370 is named PER 1, and the other version is named PER 2. A model provides either PER 1 or PER 2.

Unless otherwise noted, the descriptions in this section apply to both PER 1 and PER 2. The differences between PER 1 and PER 2 are pointed out in the section.



LzLabs GmbH. Ex. 1008-124

The purpose of PER (PER 1 or PER 2) is to assist in debugging programs. It permits the program to be alerted to the following types of events:

- Execution of a successful branch instruction. PER 2 provides the option of having an event occur only when the branch-target location is within the designated storage area.
- Fetching of an instruction from the designated storage area.
- Alteration of the contents of the designated storage area. PER 2 provides the option of having an event occur only when the storage area is within designated address spaces.
- Alteration of the contents of designated general registers. This type of event can occur only with PER 1, not with PER 2.
- Execution of the STORE USING REAL ADDRESS instruction.

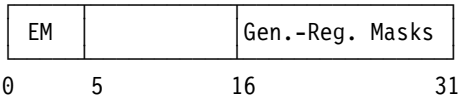
The program can selectively specify that one or more of the above types of events be recognized, except that the event for STORE USING REAL ADDRESS can be specified only along with the storage-alteration event. The information concerning a PER event is provided to the program by means of a program interruption, with the cause of the interruption being identified in the interruption code.

If a model implements ESA/390 with PER 2 and also System/370, general-register-alteration events may be omitted in System/370, depending on the model.

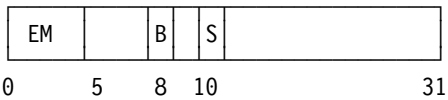
## Control-Register Allocation and Segment-Table Designation

The information for controlling PER resides in control registers 9, 10, and 11 and the segment-table designation. The information in the control registers has the following format:

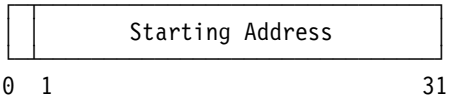
PER-1 Control Register 9



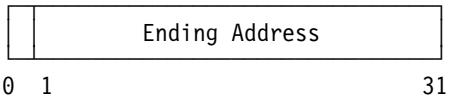

PER-2 Control Register 9




Control Register 10




Control Register 11




***PER-Event Masks (EM):*** With PER 1, bits 0-4 of control register 9 specify which types of events are recognized. With PER 2, bits 0-2 and 4 provide this specification. The bits are assigned as follows:

Bit 0: Successful-branching event
Bit 1: Instruction-fetching event
Bit 2: Storage-alteration event
Bit 3: General-register-alteration event (PER 1 only)
Bit 4: Store-using-real-address event (bit 2 must be one also)

Bits 0-4, when ones, specify that the corresponding types of events be recognized. However, bit 4 is effective for this purpose only when bit 2 is also one. When bit 2 is one, the storage-alteration event is recognized. When bits 2 and 4 are ones, both the storage-alteration event and the store-using-real-address event are recognized. When a bit is zero, the corresponding type of event is not recognized. When bit 2 is zero, both the storage-alteration event and the store-using-real-address event are not recognized. With PER 2, no type of event corresponds to bit 3, and bit 3 is ignored.

***Branch-Address Control (B):*** With PER 2, bit 8 of control register 9 specifies, when one, that successful-branching events occur only for branches that are to a location within the designated storage area. With PER 1, or with PER 2 when bit 8 is zero, successful-branching events occur regardless of the branch-target address. Bit 8 is ignored by PER 1.

***Storage-Alteration-Space Control (S):*** With PER 2, bit 10 of control register 9 specifies, when one, that storage-alteration events occur as a result of references to the designated storage area only within designated address spaces. An address space is designated as one for which


LzLabs GmbH. Ex. 1008-125

storage-alteration events occur by means of the storage-alteration-event bit in the segment-table designation that is used to translate references to the address space. Bit 10 is ignored when DAT is off. With PER 1, or with PER 2 when DAT is off or bit 10 is zero, storage-alteration events are not restricted to occurring for only particular address spaces. Bit 10 is ignored by PER 1.

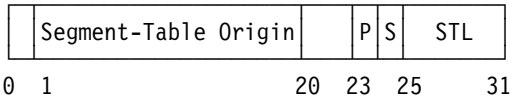
***PER General-Register Masks:*** With PER 1, bits 16-31 of control register 9 specify which general registers are designated for recognition of the alteration of their contents. The 16 bits, in the sequence of ascending bit numbers, correspond one for one with the 16 registers, in the sequence of ascending register numbers. When a bit is one, the alteration of the associated register is recognized; when it is zero, the alteration of the register is not recognized. With PER 2, general-register-alteration events do not occur, and bits 16-31 are ignored.

***PER Starting Address:*** Bits 1-31 of control register 10 are the address of the beginning of the designated storage area.

***PER Ending Address:*** Bits 1-31 of control register 11 are the address of the end of the designated storage area.

The segment-table designation has the following format:

Segment-Table Designation




***Storage-Alteration-Event Bit (S):*** With PER 2, when the storage-alteration-space control in control register 9 is one, bit 24 of the segment-table designation specifies, when one, that the address space defined by the segment-table designation is one for which storage-alteration events can occur. Bit 24 is examined when the segment-table designation is used to perform dynamic-address translation for a storage-operand store reference. The segment-table designation may be the PSTD, SSTD, or HSTD in control register 1, 7, or 13, respectively, or it may be obtained from an ASN-second-table entry during access-register translation. Instead of being obtained from an ASN-second-table entry in main storage, bit 24 may be obtained from an ASN-second-table entry in the ART-lookaside buffer (ALB). Bit 24 is ignored when the storage-alteration-space control is zero, and it is always ignored by PER 1.

**Programming Notes:**

1. Models may operate at reduced performance while the CPU is enabled for PER events. In order to ensure that CPU performance is not degraded because of the operation of the PER facility, programs that do not use it should disable the CPU for PER events by setting either the PER mask in the PSW to zero or the PER-event masks in control register 9 to zero, or both. No degradation due to PER occurs when either of these fields is zero.
2. Some degradation may be experienced on some models every time control registers 9, 10, and 11 are loaded, even when the CPU is disabled for PER events (see the programming note under “Storage-Area Designation”).

## Operation

PER is under control of bit 1 of the PSW, the PER mask. When the PER mask, a particular PER-event mask bit, and, for general-register-alteration events (PER 1 only), a particular general-register mask bit are all ones, the CPU is enabled for the corresponding type of event; otherwise, it is disabled. However, the CPU is enabled for the store-using-real-address event only when the storage-alteration mask bit and the store-using-real-address mask bit are both ones.

An interruption due to a PER event normally occurs after the execution of the instruction responsible for the event. The occurrence of the event does not affect the execution of the instruction, which may be completed, partially completed, terminated, suppressed, or nullified. However, on a model on which z/Architecture is installed, recognition of a storage-alteration event causes no more than 4K bytes to be stored beginning with the byte that caused the event, and this may result in partial completion of an interruptible instruction.

When the CPU is disabled for a particular PER event at the time it occurs, either by the PER



LzLabs GmbH. Ex. 1008-126

mask in the PSW or by the masks in control register 9, the event is not recognized.

A change to the PER mask in the PSW or to the PER control fields in control registers 9, 10, and 11 affects PER starting with the execution of the immediately following instruction.

A change to the storage-alteration-event bit in a segment-table designation in control register 1, 7, or 13 also affects PER starting with the execution of the immediately following instruction. A change to the storage-alteration-event bit in a segment-table designation that may be obtained, during access-register translation, from an ASN-second-table entry in either main storage or the ALB does not necessarily have an immediate, if any, effect on PER. However, PER is affected immediately after PURGE ALB is executed.

If a PER event occurs during the execution of an instruction which changes the CPU from being enabled to being disabled for that type of event, that PER event is recognized.

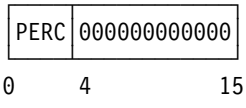
PER events may be recognized in a trial execution of an instruction, and subsequently the instruction, DAT-table entries, and operands may be refetched for the actual execution. If any refetched field was modified by another CPU or by a channel program between the trial execution and the actual execution, it is unpredictable whether the PER events indicated are for the trial or the actual execution.

For special-purpose instructions that are not described in this publication, the operation of PER may not be exactly as described in this section.

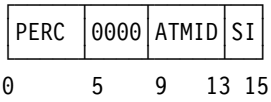
## Identification of Cause

A program interruption for PER sets bit 8 of the interruption code to one and places identifying information in real storage locations 150-155, and in location 161 if the PER event is a storage-alteration event. Additional information is provided by means of the instruction address in the program old PSW and the ILC. The information stored in real locations 150-155 and 161 has the following format:

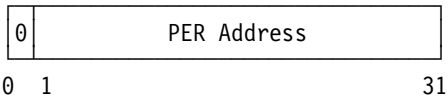
PER-1 Locations 150-151:



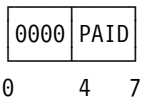

PER-2 Locations 150-151:




Locations 152-155:




Location 161:




***PER Code (PERC):*** With PER 1, the occurrence of PER events is indicated by ones in bit positions 0-3 of real location 150, the PER code. With PER 2, the PER code is bits 0-2 and 4. The bit position in the PER code for a particular type of event is the same as the bit position for that event in the PER-event-mask field in control register 9, except as follows:

- With PER 1, when bits 2 and 4 in control register 9 are both ones, a one in bit position 2 of location 150 indicates the occurrence of either a storage-alteration event or a store-using-real-address event.
- With PER 2, a one in bit position 2 and a zero in bit position 4 of location 150 indicate a storage-alteration event, while ones in bit positions 2 and 4 indicate a store-using-real-address event.

When a program interruption occurs, more than one type of PER event can be concurrently indicated. Additionally, if another program-interruption condition exists, the interruption code for the program interruption may indicate both the PER events and the other condition.

***Addressing-and-Translation-Mode Identification (ATMID):*** With PER 2, during a program interruption when a PER event is indicated, bits 32, 5, 16, and 17 of the PSW at the beginning of the execution of the instruction that caused the event may be stored in bit positions 10-13, respectively, of real locations 150-151. If bits 32, 5, 16, and 17 are stored, then a one bit is stored in bit position 9 of locations 150-151. If bits 32, 5,


LzLabs GmbH. Ex. 1008-127

16, and 17 are not stored, then zero bits are stored in bit positions 9-13 of locations 150-151.

Bits 9-13 of real locations 150-151 are named the addressing-and-translation-mode identification (ATMID). Bit 9 is named the ATMID-validity bit. When bit 9 is zero, it indicates that an invalid ATMID (all zeros) was stored.

The meanings of the bits of a valid ATMID are as follows:

| Bit | Meaning |
|---|---|
| 9 | ATMID-validity bit |
| 10 | PSW bit 32 |
| 11 | PSW bit 5 |
| 12 | PSW bit 16 |
| 13 | PSW bit 17 |

A valid ATMID is necessarily stored only if the PER event was caused by one of the following instructions:

- BRANCH AND SAVE AND SET MODE (BASSM)
- BRANCH AND SET AUTHORITY (BSA)
- BRANCH AND SET MODE (BSM)
- BRANCH IN SUBSPACE GROUP (BSG)
- LOAD PSW (LPSW)
- PROGRAM CALL (PC)
- PROGRAM CALL FAST (PCF)
- PROGRAM RETURN (PR)
- PROGRAM TRANSFER (PT)
- RESUME PROGRAM (RP)
- SET ADDRESS SPACE CONTROL (SAC)
- SET ADDRESS SPACE CONTROL FAST (SACF)
- SET SYSTEM MASK (SSM)
- STORE THEN AND SYSTEM MASK (STNSM)
- STORE THEN OR SYSTEM MASK (STOSM)
- SUPERVISOR CALL (SVC)
- TRAP (TRAP2, TRAP4)

It is unpredictable whether a valid ATMID is stored if the PER event was caused by any other instruction.

In the case of an instruction-fetching PER event caused by SET ADDRESS SPACE CONTROL or SET ADDRESS SPACE CONTROL FAST, bits 12 and 13 of the ATMID, which correspond to bits 16 and 17 of the PSW, may indicate that the CPU was in the primary-space mode when it actually was in the primary-space, secondary-space, or access-register mode. In any of those modes, the instruction fetch is from the primary address space.

***PER STD Identification (SI):*** With PER 2, if a storage-alteration event is indicated in the PER code (bit 2 is one and bit 4 is zero) and this event occurred when DAT was on, bits 14 and 15 of locations 150-151 are set to identify the segment-table designation (STD) that was used to translate the reference that caused the event, as follows:

| Bits 14-15 | Meaning |
|---|---|
| 00 | Primary STD was used. |
| 01 | An AR-specified STD was used. The PER access id, real location 161, can be examined to determine the STD used. However, if the primary, secondary, or home STD was used, bits 14 and 15 may be set to 00, 10, or 11, respectively, instead of to 01. |
| 10 | Secondary STD was used. |
| 11 | Home STD was used. |

The CPU may avoid setting bits 14 and 15 to 01 by recognizing that access-list-entry token (ALET) 00000000 or 00000001 hex was used or that the ALET designated, through an access-list entry, an ASN-second-table entry containing an STD equal to the primary STD, secondary STD, or home STD.

If a storage-alteration event is not indicated in the PER code (bit 2 is zero or bit 4 is one) or DAT was off, zeros are stored in bit positions 14 and 15.

With PER 1, zeros are stored in bit positions 4-15 of locations 150-151. With PER 2, zeros are stored in bit positions 3 and 5-8 of locations 150-151.

***PER Address:*** The PER-address field at locations 152-155 contains the instruction address used to fetch the instruction in execution when one or more PER events were recognized. When the instruction is the target of EXECUTE, the instruction address used to fetch the EXECUTE instruction is placed in the PER-address field. A zero is stored in bit position 0 of real location 152.

***PER Access Identification (PAID):*** If a storage-alteration event is indicated in the PER code, an indication of the address space to which the event applies may be stored at location 161. If the



LzLabs GmbH. Ex. 1008-128

access used an AR-specified segment-table designation, the number of the access register used is stored in bit positions 4-7 of location 161, and zeros are stored in bit positions 0-3. However, with PER 1 only, the contents of location 161 are unpredictable if the instruction that caused the event turned DAT off. With PER 1 or PER 2, the contents of location 161 are also unpredictable if (1) the CPU was in the access-register mode but the access was an implicit reference to the linkage stack, (2) the CPU was not in the access-register mode, or (3) a store-using-real-address event instead of a storage-alteration event occurred. If bit 2 of the PER code is zero, location 161 remains unchanged.

***Instruction Address:*** The instruction address in the program old PSW is the address of the instruction which would have been executed next, unless another program condition is also indicated, in which case the instruction address is that determined by the instruction ending due to that condition.

***ILC:*** The ILC indicates the length of the instruction designated by the PER address, except when a concurrent specification exception for the PSW introduced by LOAD PSW, PROGRAM RETURN, or a supervisor-call interruption sets an ILC of 0.

**Programming Notes:**

1. PSW bit 32 is the addressing-mode bit (24-bit mode if the bit is zero, or 31-bit mode if the bit is one), PSW bit 5 is the DAT-mode bit, and PSW bits 16 and 17 are the address-space-control bits. For the handling of instruction and logical addresses in the different translation modes, see “Translation Modes” on page 3-28. The following notes apply to PER 2.

2. A valid ATMID allows the program handling the PER event to determine the address space from which the instruction that caused the event was fetched and also to determine which translation mode applied to the storage-operand references of the instruction, if any. Each of the instructions for which a valid ATMID is necessarily stored can change one or more of PSW bits 5, 16, and 17, with the result that the values of those bits in the program old PSW that is stored because of the PER event are not necessarily the values that existed at the beginning of the execution of the instruction that caused the event. The instructions for which a valid ATMID is necessarily stored are the only instructions that can change any of PSW bits 5, 16, and 17.

3. If a storage-alteration PER event is indicated and DAT was on when the event occurred, an indication of the segment-table designation that was used to translate the reference that caused the event is given by the PER STD identification, bits 14 and 15 of real locations 150-151. If bits 14 and 15 indicate that an AR-specified segment-table designation was used, the PER access identification in real location 161 can be used to determine the address space that was referenced. To determine if DAT was on, the program handling the PER event should first examine the ATMID-validity bit to determine whether a valid ATMID was stored and, if it was stored, then examine the DAT-mode bit in the ATMID. If a valid ATMID was not stored, the program should examine the DAT-mode bit in the program old PSW.

4. If a valid ATMID is stored, it also allows the program handling the PER event to determine the addressing mode (24-bit or 31-bit) that existed for the instruction that caused the PER event. This knowledge of the addressing mode allows the program to determine, without any chance of error, the meaning of one bits in bit positions 1-7 of the addresses of the instruction and of the storage operands, if any, of the instruction and, thus, to determine accurately the locations of the instruction and operands. Note that the address of the instruction is not necessarily provided without error by the PER address in real locations 152-155 because that address may be the address of an EXECUTE instruction, with the address of the target instruction still to be determined from the fields that specify the second-operand address of the EXECUTE instruction. Also note that another possible source of error is that, in the 24-bit addressing mode, an instruction or operand may wrap around in storage by beginning just below the 16M-byte boundary.

5. A valid ATMID is necessarily stored for all instructions that can change the addressing-mode bit. However, the ATMID mechanism does not provide complete assurance that the instruction causing a PER event and the


LzLabs GmbH. Ex. 1008-129

instruction's operands can be located accurately because LOAD CONTROL and LOAD ADDRESS SPACE PARAMETERS can change the segment-table designation that was used to fetch the instruction.

## Priority of Indication

When a program interruption occurs and more than one PER event has been recognized, all recognized PER events are concurrently indicated in the PER code. Additionally, if another program-interruption condition concurrently exists, the interruption code for the program interruption indicates both the PER condition and the other condition.

In the case of an instruction-fetching event for SUPERVISOR CALL, the program interruption occurs immediately after the supervisor-call interruption.

If a PER event is recognized during the execution of an instruction which also introduces a new PSW with the type of PSW-format error which is recognized early (see "Exceptions Associated with the PSW" on page 6-9), both the specification exception and PER are indicated concurrently in the interruption code of the program interruption. If the PSW-format error is of the type which is recognized late, only PER is indicated in the interruption code. In both cases, the invalid PSW is stored as the program old PSW.

Recognition of a PER event does not normally affect the ending of instruction execution. However, in the following cases, execution of an interruptible instruction is not completed normally:

1. When the instruction is due to be interrupted for an asynchronous condition (I/O, external, restart, or repressible machine-check condition), a program interruption for the PER event occurs first, and the other interruptions occur subsequently (subject to the mask bits in the new PSW) in the normal priority order.
2. When the stop function is performed, a program interruption indicating the PER event occurs before the CPU enters the stopped state.
3. When any program exception is recognized, PER events recognized for that instruction execution are indicated concurrently.
4. Depending on the model, in certain situations, recognition of a PER event may appear to cause the instruction to be interrupted prematurely without concurrent indication of a program exception, without an interruption for any asynchronous condition, and without the CPU entering the stopped state. In particular, on a model on which z/Architecture is installed, recognition of a storage-alteration event causes no more than 4K bytes to be stored beginning with the byte that caused the event.

In cases 1 and 2 above, if the only PER event that has been recognized is an instruction-fetching event and another unit of operation of the instruction remains to be executed, the event may be discarded, with the result that a program interruption does not occur. Whether the event is discarded is unpredictable.

**Programming Notes:**

1. In the following cases, an instruction can both cause a program interruption for a PER event and change the value of fields controlling an interruption for PER events. The original field values determine whether a program interruption takes place for the PER event.
   a. The instructions LOAD PSW, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and SUPERVISOR CALL can cause an instruction-fetching event and disable the CPU for PER interruptions. Additionally, STORE THEN AND SYSTEM MASK can cause a storage-alteration event to be indicated. In all these cases, the program old PSW associated with the program interruption for the PER event may indicate that the CPU was disabled for PER events.
   b. An instruction-fetching event may be recognized during execution of a LOAD CONTROL instruction that changes the value of the PER-event masks in control register 9 or the addresses in control registers 10 and 11 controlling indication of instruction-fetching events.
   c. In the access-register mode, a storage-alteration event that is permitted by a one value of the storage-alteration-event bit in a segment-table designation in an ASN-second-table entry (designated by an access-list entry) may be caused by any



LzLabs GmbH. Ex. 1008-130

store-type instruction that changes the value of the bit from one to zero.

2. No instruction can both change the values of general-register-alteration masks (PER 1 only) and cause a general-register-alteration event to be recognized.
3. When a PER interruption occurs during the execution of an interruptible instruction, the ILC indicates the length of that instruction or EXECUTE, as appropriate. When a PER interruption occurs as a result of LOAD PSW, PROGRAM RETURN, or SUPERVISOR CALL, the ILC indicates the length of these instructions or EXECUTE, as appropriate, unless a concurrent specification exception on LOAD PSW or PROGRAM RETURN calls for an ILC of 0.
4. When a PER interruption is caused by branching, the PER address identifies the branch instruction (or EXECUTE, as appropriate), whereas the old PSW points to the next instruction to be executed. When the interruption occurs during the execution of an interruptible instruction, the PER address and the instruction address in the old PSW are the same.

## Storage-Area Designation

Two types of PER events. — instruction fetching and storage alteration. — always involve the designation of an area in storage. With PER 2, successful-branching events may involve this designation. The storage area starts at the location designated by the starting address in control register 10 and extends up to and including the location designated by the ending address in control register 11. The area extends to the right of the starting address.

An instruction-fetching event occurs whenever the first byte of an instruction or the first byte of the target of an EXECUTE instruction is fetched from the designated area. A storage-alteration event occurs when a store access is made to the designated area by using an operand address that is defined to be a logical or a virtual address. However, with PER 2, when DAT is on and the storage-alteration-space control in control register 9 is one, a storage-alteration event occurs only when the storage area is within an address space for which the storage-alteration-event bit in the segment-table designation is one. A storage-alteration event does not occur for a store access made with an operand address defined to be a real address. With PER 2, when the branch-address control in control register 9 is one, a successful-branching event occurs when the first byte of the branch-target instruction is within the designated area.

The set of addresses designated for successful-branching, instruction-fetching, and storage-alteration events wraps around at address 2,147,483,647; that is, address 0 is considered to follow address 2,147,483,647. When the starting address is less than the ending address, the area is contiguous. When the starting address is greater than the ending address, the set of locations designated includes the area from the starting address to address 2,147,483,647 and the area from address 0 to, and including, the ending address. When the starting address is equal to the ending address, only that one location is designated.

Address comparison for successful-branching, instruction-fetching, and storage-alteration events is always performed using 31-bit addresses. This is accomplished in the 24-bit addressing mode by extending the virtual, logical, or instruction address on the left with seven zero bits before comparing it with the starting and ending addresses.

**Programming Note:** In some models, performance of address-range checking is assisted by means of an extension to each page-table entry in the TLB. In such an implementation, changing the contents of control registers 10 and 11 when the successful-branching, instruction-fetching, or storage-alteration-event mask is one, or setting any of these PER-event masks to one, may cause the TLB to be cleared of entries. This degradation may be experienced even when the CPU is disabled for PER events. Thus, when possible, the program should avoid loading control registers 9, 10, or 11.


LzLabs GmbH. Ex. 1008-131

## PER Events

### Successful Branching

With PER 1, or with PER 2 when the branch-address control in control register 9 is zero, a successful-branching event occurs independent of the branch-target address. With PER 2 when the branch-address control is one, a successful-branching event occurs only when the first byte of the branch-target instruction is in the storage area designated by control registers 10 and 11.

Subject to the effect of the branch-address control, a successful-branching event occurs whenever one of the following instructions causes branching:

- BRANCH AND LINK (BAL, BALR)
- BRANCH AND SAVE (BAS, BASR)
- BRANCH AND SAVE AND SET MODE (BASSM)
- BRANCH AND SET AUTHORITY (BSA)
- BRANCH AND SET MODE (BSM)
- BRANCH AND STACK (BAKR)
- BRANCH IN SUBSPACE GROUP (BSG)
- BRANCH ON CONDITION (BC, BCR)
- BRANCH ON COUNT (BCT, BCTR)
- BRANCH ON INDEX HIGH (BXH)
- BRANCH ON INDEX LOW OR EQUAL (BXLE)
- BRANCH RELATIVE AND SAVE (BRAS)
- BRANCH RELATIVE AND SAVE LONG (BRASL)
- BRANCH RELATIVE ON CONDITION (BRC)
- BRANCH RELATIVE ON CONDITION LONG (BRCL)
- BRANCH RELATIVE ON COUNT (BRCT)
- BRANCH RELATIVE ON INDEX HIGH (BRXH)
- BRANCH RELATIVE ON INDEX LOW OR EQUAL (BRXLE)
- RESUME PROGRAM (RP)
- TRAP (TRAP2, TRAP4)

Subject to the effect of the branch-address control, a successful-branching event also occurs whenever one of the following instructions causes branching:

- PROGRAM CALL (PC)
- PROGRAM CALL FAST (PCF)
- PROGRAM RETURN (PR)
- PROGRAM TRANSFER (PT)

For PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, and PROGRAM TRANSFER, the branch-target address is considered to be the new instruction address that is placed in the PSW by the instruction.

A successful-branching event causes a PER successful-branching event to be recognized if bit 0 of the PER-event masks is one and the PER mask in the PSW is one.

A PER successful-branching event is indicated by setting bit 0 of the PER code to one.

### Instruction Fetching

An instruction-fetching event occurs if the first byte of the instruction is within the storage area designated by control registers 10 and 11. An instruction-fetching event also occurs if the first byte of the target of EXECUTE is within the designated storage area.

An instruction-fetching event causes a PER instruction-fetching event to be recognized if bit 1 of the PER-event masks is one and the PER mask in the PSW is one.

If an instruction-fetching event is the only PER event recognized for an interruptible instruction that is to be interrupted because of an asynchronous condition (I/O, external, restart, or repressible machine-check condition) or the performance of the stop function, and if a unit of operation of the instruction remains to be executed, the instruction-fetching event may be discarded, and whether it is discarded is unpredictable.

The PER instruction-fetching event is indicated by setting bit 1 of the PER code to one.

### Storage Alteration

A storage-alteration event occurs whenever a CPU, by using a logical or virtual address, makes a store access without an access exception to the storage area designated by control registers 10 and 11. However, with PER 2 when DAT is on and the storage-alteration-space control in control register 9 is one, the event occurs only if the storage-alteration-event bit is one in the segment-table designation that is used by DAT to translate the reference to the storage location.

The contents of storage are considered to have been altered whenever the CPU executes an instruction that causes all or part of an operand to



LzLabs GmbH. Ex. 1008-132

be stored within the designated storage area. Alteration is considered to take place whenever storing is considered to take place for purposes of indicating protection exceptions, except that recognition does not occur for the storing of data by a channel program. (See "Recognition of Access Exceptions" on page 6-35.) Storing constitutes alteration for PER purposes even if the value stored is the same as the original value.

Implied locations that are referred to by the CPU in the process of performing an interruption are not monitored. Such locations include PSW and interruption-code locations. These locations, however, are monitored when information is stored there explicitly by an instruction. Similarly, monitoring does not apply to the storing of data by a channel program. Implied locations in the linkage stack, which are stored in by instructions that operate on the linkage stack, are monitored.

The I/O instructions are considered to alter the second-operand location only when storing actually occurs.

When an interruptible vector instruction which performs storing is interrupted, and PER storage alteration applies to storage locations corresponding to elements due to be changed beyond the point of interruption, PER storage alteration is indicated if any such store actually occurred and may be indicated even if such a store did not occur. PER storage alteration is reported for such locations only if no access exception exists at the time that the instruction is executed.

Storage alteration does not apply to instructions whose operands are specified to have real addresses. Thus, storage alteration does not apply to INVALIDATE PAGE TABLE ENTRY, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY EXTENDED, STORE USING REAL ADDRESS, TEST BLOCK, and TEST PENDING INTERRUPTION (when the effective address is zero).

A storage-alteration event causes a PER storage-alteration event to be recognized if bit 2 of the PER-event masks is one and the PER mask in the PSW is one. Bit 4 of the PER-event masks is ignored when determining whether a PER storage-alteration event is to be recognized.

With PER 1, a PER storage-alteration event is indicated by setting bit 2 of the PER code to one. However, when bit 2 of the PER code and bit 4 of the PER-event masks are both ones, a store-using-real-address event, instead of a storage-alteration event, may have occurred. With PER 2, a PER storage-alteration event is indicated by setting bit 2 of the PER code to one and bit 4 of the PER code to zero.

## General-Register Alteration

With PER 1, a general-register-alteration event occurs whenever the contents of a general register are replaced. With PER 2, general-register-alteration events do not occur. The remainder of this description applies only to PER 1.

The contents of a general register are considered to have been altered whenever a new value is placed in the register. Recognition of the event is not contingent on the new value being different from the previous one. The execution of an RR-format arithmetic, logical, or movement instruction is considered to fetch the contents of the register, perform the indicated operation, if any, and then replace the value in the register. A register can be designated by an RR, RRE, RS, or RX instruction or implicitly, such as in TRANSLATE AND TEST and EDIT AND MARK.

The instructions MOVE LONG and COMPARE LOGICAL LONG are always considered to alter the contents of the four registers specifying the two operands, including the cases where the padding byte is used, when both operands have zero length. However, when condition code 3 is set for MOVE LONG, the general registers containing the operand lengths may or may not be considered as having been altered.

The instruction COMPARE UNTIL SUBSTRING EQUAL is always considered to alter the contents of the even-numbered registers specifying the two operands. When the operand length or the substring length is zero, the odd-numbered register specifying an operand may or may not be considered as having been altered.

The instruction INSERT CHARACTERS UNDER MASK is not considered to alter the general register when the mask is zero.


LzLabs GmbH. Ex. 1008-133

The instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP are considered to alter the general register, or general-register pair, designated by $R_1$, only when the contents are actually replaced, that is, when the first and second operands are not equal.

It is unpredictable whether general-register-alteration events are indicated for instructions of the vector facility.

A general-register-alteration event causes a PER general-register-alteration event to be recognized if bit 3 of the PER-event masks is one, the PER mask in the PSW is one, and the corresponding bit in the PER general-register mask is one.

The PER general-register-alteration event is indicated by setting bit 3 of the PER code to one.

**Programming Note:** The following are some examples of general-register alteration:

1. Register-to-register load instructions are considered to alter the register contents even when both operand addresses designate the same register.
2. Addition or subtraction of zero and multiplication or division by one are considered to constitute alteration.
3. Logical and fixed-point shift operations are considered to alter the register contents even for shift amounts of zero.
4. The branching instructions BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL are considered to alter the first operand even when zero is added to its value.

### Store Using Real Address

A store-using-real-address event occurs whenever the STORE USING REAL ADDRESS instruction is executed.

There is no relationship between the store-using-real-address event and the designated storage area.

A store-using-real-address event causes a PER store-using-real-address event to be recognized if bits 2 and 4 of the PER-event mask are ones and the PER mask in the PSW is one.

With PER 1, a PER store-using-real-address event is indicated by setting bit 2 of the PER code to one. However, when bit 2 of the PER code is one, a storage-alteration event, instead of a store-using-real-address event, may have occurred. With PER 2, a PER store-using-real-address event is indicated by setting bits 2 and 4 of the PER code to one.

## Indication of PER Events Concurrently with Other Interruption Conditions

The following rules govern the indication of PER events caused by an instruction that also causes a program exception, a monitor event, a space-switch event, or a supervisor-call interruption.

1. The indication of an instruction-fetching event does not depend on whether the execution of the instruction was completed, terminated, suppressed, or nullified. However, special cases of suppression and nullification are as follows:
   a. When the instruction is designated by an odd instruction address in the PSW, the instruction-fetching event is not indicated.
   b. When an access exception applies to the first, second, or third halfword of the instruction, it is unpredictable whether the instruction-fetching event is indicated.
   c. When the target address of EXECUTE is odd or an access exception applies to the first, second, or third halfword of the target instruction, it is unpredictable whether the instruction-fetching event is indicated for the target instruction, and it is also unpredictable whether the event is indicated for the EXECUTE instruction.
2. When the operation is completed or partially completed, the event is indicated, regardless of whether any program exception, space-switch event, or monitor event is also recognized.
3. Successful branching, storage alteration, general-register alteration, and store using real address are not indicated for an operation or, in case the instruction is interruptible, for a unit of operation that is suppressed or nullified.
4. When the execution of the instruction is terminated, general-register or storage alteration is indicated whenever the event has occurred,



LzLabs GmbH. Ex. 1008-134

and a model may indicate the event if the event would have occurred had the execution of the instruction been completed, even if altering the contents of the result field is contingent on operand values. For purposes of this definition, the occurrence of those exceptions which permit termination (addressing, protection, and data) is considered to cause termination, even if no result area is changed.

5. When LOAD PSW, PROGRAM RETURN, SET SYSTEM MASK, STORE THEN OR SYSTEM MASK, or SUPERVISOR CALL causes a PER condition and at the same time introduces a new PSW with the type of PSW-format error that is recognized immediately after the PSW becomes active, the interruption code identifies both the PER condition and the specification exception.

6. When LOAD PSW, PROGRAM RETURN, or SUPERVISOR CALL causes a PER condition and at the same time introduces a new PSW with the type of PSW-format error that is recognized as part of the execution of the following instruction, the introduced PSW is stored as the old PSW without the following instruction being fetched or executed and without the specification exception being recognized.

The indication of PER events concurrently with other program-interruption conditions for the same instruction, as described in cases 1-4 above, is summarized in Figure 4-5 on page 4-26. Cases 5 and 6 are shown in Figure 4-6 on page 4-28.

**Programming Notes:**

1. The execution of the interruptible instructions MOVE LONG, TEST BLOCK, and COMPARE LOGICAL LONG can cause events for general-register alteration and instruction fetching. Additionally, MOVE LONG can cause the storage-alteration event.

   Interruption of such an instruction may cause a PER event to be indicated more than once. It may be necessary, therefore, for a program to remove the redundant event indications from the PER data. The following rules govern the indication of the applicable events during execution of these instructions:

   a. The instruction-fetching event is indicated whenever the instruction is fetched for execution, regardless of whether it is the initial execution or a resumption, except that the event may be discarded (not indicated) if it is the only PER event to be indicated, the interruption is due to an asynchronous interruption condition or the performance of the stop function, and a unit of operation of the instruction remains to be executed.

   b. The general-register-alteration event is indicated on the initial execution and on each resumption and does not depend on whether or not the register actually is changed.

   c. The storage-alteration event is indicated only when data has been stored in the designated storage area by the portion of the operation starting with the last initiation and ending with the last byte transferred before the interruption. No special indication is provided on premature interruptions as to whether the event will occur again upon the resumption of the operation. When the designated storage area is a single byte location, a storage-alteration event can be recognized only once in the execution of MOVE LONG.

2. The following is an outline of the general action a program must take to delete multiple entries in the PER data for an interruptible instruction so that only one entry for each complete execution of the instruction is obtained:

   a. Check to see if the PER address is equal to the instruction address in the old PSW and if the last instruction executed was interruptible.

   b. If both conditions are met, delete instruction-fetching and register-alteration events.

   c. If both conditions are met and the event is storage alteration, delete the event if some part of the remaining destination operand is within the designated storage area.

3. An example of the indication of an instruction-fetching PER event caused by either a LOAD PSW instruction or the following instruction, in connection with an early PSW-format error or odd instruction address introduced by the


LzLabs GmbH. Ex. 1008-135

LOAD PSW instruction, is shown in Figure 4-6 on page 4-28.

| Concurrent Condition | Type of Ending | PER Event | | | | |
|---|---|---|---|---|---|---|
| | | Branch | Instr Fetch | Storage Alter. | GR Alter.[1] | STURA |
| Specification | | | | | | |
| Odd instruction address in the PSW | S | No | No | No | No | No |
| Instruction access | N or S | No | U | No | No | No |
| Specification | | | | | | |
| EXECUTE target address odd | S | No | U[2] | No | No | No |
| EXECUTE target access | N or S | No | U[2] | No | No | No |
| Other nullifying | N | No | Yes | No[3] | No[3] | - |
| Other suppressing | S | No | Yes | No[3] | No[3] | No |
| All terminating | T | No | Yes | Yes[4] | Yes[4] | - |
| All completing | C | Yes | Yes | Yes | Yes | - |

**Explanation:**

- The condition does not apply.

[1] With PER 2, PER general-register-alteration events do not occur and are not indicated.

[2] It is unpredictable whether the PER event is indicated for the target instruction, and it is unpredictable whether the PER event is indicated for the EXECUTE instruction.

[3] Although PER events of this type are not indicated for the current unit of operation of an interruptible instruction, PER events of this type that were recognized on completed units of operation of the interruptible instruction are indicated.

[4] This event may be indicated, depending on the model, if the event has not occurred but would have been indicated if execution had been completed.

C The operation or, in the case of the interruptible instructions, the unit of operation is completed.

N The operation or, in the case of the interruptible instructions, the unit of operation is nullified.

S The operation or, in the case of the interruptible instructions, the unit of operation is suppressed.

T The execution of the instruction is terminated.

Yes The PER event is indicated with the other program-interruption condition if the event has occurred; that is, the instruction address in the PSW was replaced and the branch-address control and designated storage area allow the event occurrence, an attempt was made to execute an instruction whose first byte is located in the designated storage area, or the contents of the the designated storage area or a designated general register were altered.

*Figure 4-5 (Part 1 of 2). Indication of PER Events with Other Concurrent Conditions*



LzLabs GmbH. Ex. 1008-136

**Explanation** (Continued):

No  The PER event is not indicated.

U  It is unpredictable whether the PER event is indicated.

*Figure 4-5 (Part 2 of 2). Indication of PER Events with Other Concurrent Conditions*



LzLabs GmbH. Ex. 1008-137

| LPSW at 4000 Loads a PSW | | Designated Storage Area Includes | | Two-Byte Instruction Is at 6000 | | | |
|---|---|---|---|---|---|---|---|
| PSW Has Early PSW-Format Error | Instruction Address in PSW | 4000 | 6000-6001 | Interruption Code | Address in Program Old PSW | ILC | PER Address |
| N | 6000 | N | N | None | - | - | - |
| N | 6000 | N | Y | P | 6002 | 1 | 6000 |
| N | 6000 | Y | - | P | 6000 | 2 | 4000 |
| N | 6001 | N | N | S | 6001+J[1] | K[1] | None |
| N | 6001 | N | Y | S[2] | 6001+J[1] | K[1] | None |
| N | 6001 | Y | - | P[2] [3] | 6001[2] [3] | 2 | 4000 |
| Y | 6000 | N | - | S | 6000 | 0[4] | None |
| Y | 6000 | Y | - | P,S[2] [3] | 6000[2] [3] | 0[4] | 4000 |
| Y | 6001 | N | - | S | 6001 | 0[4] | None |
| Y | 6001 | Y | - | P,S[2] [3] | 6001[2] [3] | 0[4] | 4000 |

**Explanation:**

- \- Immaterial or not applicable.
- [1] See "ILC on Instruction-Fetching Exceptions" on page 6-7.
- [2] See "Indication of PER Events Concurrently with Other Interruption Conditions" on page 4-24.
- [3] See "Priority of Indication" on page 4-20.
- [4] See "Zero ILC" on page 6-7.
- J Unpredictably 2, 4, or 6.
- K 1, 2, or 3 depending on whether J is 2, 4, or 6, respectively.
- N No.
- P PER event (instruction-fetching).
- S Specification exception.
- Y Yes.

*Figure 4-6. Example of Instruction-Fetching PER Event and Early PSW-Format Error or Odd Instruction Address*



LzLabs GmbH. Ex. 1008-138

## Timing

The timing facilities include three facilities for measuring time: the TOD clock, the clock comparator, and the CPU timer.

In a multiprocessing configuration, a single TOD clock may be shared by more than one CPU, or each CPU may have a separate TOD clock. Each CPU has its own clock comparator and CPU timer.

The extended-TOD-clock facility and the TOD-clock-control-override facility may be installed.

The extended-TOD-clock facility includes an extension in length of the TOD clock, a TOD programmable register for each CPU, and the instructions SET CLOCK PROGRAMMABLE FIELD and STORE CLOCK EXTENDED.

The TOD-clock-control-override facility includes the TOD-clock-control-override control, bit 10 of control register 14.

## Time-of-Day Clock

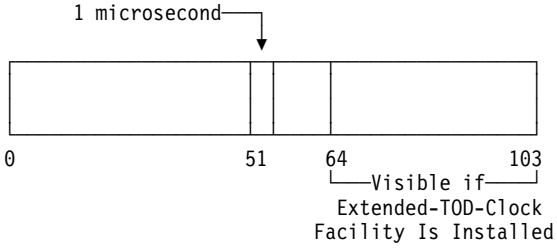
The time-of-day (TOD) clock provides a high-resolution measure of real time suitable for the indication of date and time of day. The cycle of the clock is approximately 143 years.

In a configuration with more than one CPU, each CPU may have a separate TOD clock, or more than one CPU may share a clock, depending on the model. In all cases, each CPU has access to a single clock.

### Format

The basic TOD clock is a 64-bit register. It is extended with an additional 40 rightmost bits if the extended-TOD-clock facility is installed. For ease of description, the TOD clock is treated as a 104-bit register of which the rightmost 40 bits are visible only if the extended-TOD-clock facility is installed.

The TOD clock is a binary counter with the format shown in the following illustration.




The TOD clock nominally is incremented by adding a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is incremented at such a frequency that the rate of advancing the clock is the same as if a one were added in bit position 51 every microsecond. The resolution of the TOD clock is such that the incrementing rate is comparable to the instruction-execution rate of the model.

A TOD clock is said to be in a particular multiprocessing configuration if at least one of the CPUs which shares that clock is in the configuration. Conversely, if all CPUs having access to a particular TOD clock have been removed from a particular configuration, then the TOD clock is no longer considered to be in that configuration.

When more than one TOD clock exists in the configuration, incrementing is synchronized such that all of the TOD clocks that are being incremented are incremented at exactly the same rate.

When incrementing of the clock causes a carry to be propagated out of bit position 0, the carry is ignored, and counting continues from zero. The program is not alerted, and no interruption condition is generated as a result of the overflow.

The operation of the clock is not affected by any normal activity or event in the system. Incrementing of the clock does not depend on whether the wait-state bit of the PSW is one or whether the CPU is in the operating, load, stopped, or check-stop state. Its operation is not affected by CPU, initial-CPU, or clear resets or by initial program loading. Operation of the clock is also not affected by the setting of the rate control or by an initial-machine-loading operation. Depending on the model and the configuration, a TOD clock may or may not be powered independent of a CPU that accesses it.


LzLabs GmbH. Ex. 1008-139

## States

The following states are distinguished for the TOD clock: set, not set, stopped, error, and not operational. The state determines the condition code set by execution of STORE CLOCK and STORE CLOCK EXTENDED. The clock is incremented, and is said to be running, when it is in either the set state or the not-set state.

***Not-Set State:*** When the power for the clock is turned on, the clock is set to zero, and the clock enters the not-set state. The clock is incremented when in the not-set state.

When the clock is in the not-set state, execution of STORE CLOCK or STORE CLOCK EXTENDED causes condition code 1 to be set and the current value of the running clock to be stored.

***Stopped State:*** The clock enters the stopped state when SET CLOCK is executed on a CPU accessing that clock and the execution results in the clock being set. This occurs when SET CLOCK is executed without encountering any exceptions and either any manual TOD-clock control in the configuration is set to the enable-set position or the TOD-clock-control-override control, bit 10 of control register 14, is one. The TOD-clock-control-override control is available if the TOD-clock-control-override facility is installed. The clock can be placed in the stopped state from the set, not-set, and error states. The clock is not incremented while in the stopped state.

When the clock is in the stopped state, execution of STORE CLOCK or STORE CLOCK EXTENDED by a CPU accessing that clock causes condition code 3 to be set and the value of the stopped clock to be stored.

***Set State:*** The clock enters the set state only from the stopped state. The change of state is under control of the TOD-clock-sync-control bit, bit 2 of control register 0, of the CPU which most recently caused that clock to enter the stopped state. If the bit is zero, the clock enters the set state at the completion of execution of SET CLOCK. If the bit is one, the clock remains in the stopped state until the bit is set to zero on that CPU, until another CPU executes a SET CLOCK instruction affecting the clock, or until any other clock in the configuration is incremented to a value of all zeros in bit positions 32 through the rightmost bit position that is incremented when the clock is running. If an external time reference (ETR) is installed, a signal from the ETR may be used to set the set state from the stopped state. If any clock is set to a value of all zeros in bit positions 32 through the rightmost incremented bit position and enters the set state as the result of a signal from another clock, the updating of bits 32 through the rightmost incremented bit position of the two clocks is in synchronism. Bits 0-31 of the clocks may be different.

Incrementing of the clock begins with the first stepping pulse after the clock enters the set state.

When the clock is in the set state, execution of STORE CLOCK or STORE CLOCK EXTENDED causes condition code 0 to be set and the current value of the running clock to be stored.

***Error State:*** The clock enters the error state when a malfunction is detected that is likely to have affected the validity of the clock value. It depends on the model whether the clock can be placed in this state. A timing-facility-damage machine-check-interruption condition is generated on each CPU which has access to that clock whenever it enters the error state.

When STORE CLOCK or STORE CLOCK EXTENDED is executed and the clock accessed is in the error state, condition code 2 is set, and the value stored is zero.

***Not-Operational State:*** The clock is in the not-operational state when its power is off or when it is disabled for maintenance. It depends on the model whether the clock can be placed in this state. Whenever the clock enters the not-operational state, a timing-facility-damage machine-check-interruption condition is generated on each CPU that has access to that clock.

When the clock is in the not-operational state, execution of STORE CLOCK or STORE CLOCK EXTENDED causes condition code 3 to be set, and zero is stored.



LzLabs GmbH. Ex. 1008-140

## Changes in Clock State

When the TOD clock accessed by a CPU changes value because of the execution of SET CLOCK or changes state, interruption conditions pending for the clock comparator, CPU timer, and TOD-clock-sync check may or may not be recognized for up to 1.048576 seconds ($2^{20}$ microseconds) after the change.

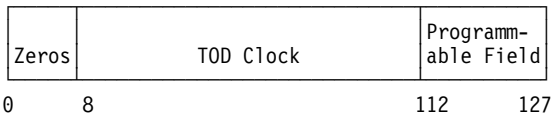
The results of channel-subsystem-monitoring-facility operations may be unpredictable as a result of changes to the TOD clock.

## Setting and Inspecting the Clock

The clock can be set to a specified value by execution of SET CLOCK if the manual TOD-clock control of any CPU in the configuration is in the enable-set position or the TOD-clock-control-override control, bit 10 of control register 14, is one. The TOD-clock-control-override control is available if the TOD-clock-control-override facility is installed. SET CLOCK sets bits of the clock with the contents of corresponding bit positions of a doubleword operand in storage.

Setting the clock replaces the values in all bit positions from bit position 0 through the rightmost position that is incremented when the clock is running. However, on some models, the rightmost bits starting at or to the right of bit 52 of the specified value are ignored, and zeros are placed in the corresponding positions of the clock. Zeros are also placed in positions to the right of bit position 63 of the clock.

The TOD clock can be inspected by executing STORE CLOCK, which causes bits 0-63 of the clock to be stored in an eight-byte operand in storage, or by executing STORE CLOCK EXTENDED, which causes bits 0-103 of the clock to be stored in bytes 1-13 of a 16-byte operand in storage. STORE CLOCK EXTENDED is available when the extended-TOD-clock facility is installed. STORE CLOCK EXTENDED stores zeros in the leftmost byte, byte 0, of its storage operand, and it obtains the TOD programmable field from bit positions 16-31 of the TOD programmable register and stores it in byte positions 14 and 15 of the storage operand. The operand stored by STORE CLOCK EXTENDED has the following format:




At some time in the future, STORE CLOCK EXTENDED on new models will store a leftmost extension of the TOD clock in byte position 0 of its storage operand; see programming note 14 on page 4-34.

Two executions of STORE CLOCK or STORE CLOCK EXTENDED, possibly on different CPUs in the same configuration, always store different values of the clock if the clock is running or, if separate clocks are accessed, both clocks are running and are synchronized. If the clock is stopped, zeros are stored in the clock value, bits 8-111 of the storage operand, in positions to the right of the rightmost bit position that is incremented when the clock is running. The programmable field continues to be stored even when the clock is stopped.

The values stored for a running clock by STORE CLOCK or STORE CLOCK EXTENDED always correctly imply the sequence of execution of these instructions by one or more CPUs for all cases where the sequence can be discovered by the program. To ensure that unique values are obtained when the value of a running clock is stored, nonzero values may be stored in positions to the right of the rightmost incremented bit position. When the value of a running clock is stored by STORE CLOCK EXTENDED, the value in bit positions 64-103 of the clock (bit positions 72-111 of the storage operand) is always nonzero; this ensures that values stored by STORE CLOCK EXTENDED are always unique when compared with values stored by STORE CLOCK and extended on the right with zeros.

For the purpose of establishing uniqueness and sequence of occurrence of the results of STORE CLOCK and STORE CLOCK EXTENDED, the 64-bit value provided by STORE CLOCK may be considered to be extended to 104 bits by appending 40 zeros on the right, with the STORE CLOCK value and STORE CLOCK EXTENDED bits 8-111 then both being treated as 104-bit unsigned binary integers.

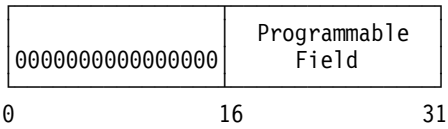
In a configuration where more than one CPU accesses the same clock, SET CLOCK is inter-


LzLabs GmbH. Ex. 1008-141

locked such that the entire contents appear to be updated concurrently; that is, if SET CLOCK instructions are executed simultaneously by two CPUs, the final result is either one or the other value. If SET CLOCK is executed by one CPU and STORE CLOCK or STORE CLOCK EXTENDED by the other, the result obtained by STORE CLOCK or STORE CLOCK EXTENDED is either the entire old value or the entire new value. When SET CLOCK is executed by one CPU, a STORE CLOCK or STORE CLOCK EXTENDED instruction executed by another CPU may find the clock in the stopped state even when the TOD-clock-sync-control bit, bit 2 of control register 0, of each CPU is zero. Since the clock enters the set state before incrementing, the first STORE CLOCK or STORE CLOCK EXTENDED instruction executed after the clock enters the set state may still find the original value introduced by SET CLOCK.

## TOD Programmable Register

When the extended-TOD-clock facility is installed, each CPU has a TOD programmable register, and the instruction SET CLOCK PROGRAMMABLE FIELD is provided. Bits 16-31 of the register contain the programmable field that is appended on the right to the TOD-clock value by STORE CLOCK EXTENDED. The register has the following format:




The register is loaded by SET CLOCK PROGRAMMABLE FIELD. Bits 16-31 of the register are stored in bit positions 112-127 of its storage operand by STORE CLOCK EXTENDED. The contents of the register are reset to a value of all zeros by initial CPU reset.

**Programming Notes:**

1. Bit position 31 of the clock is incremented every 1.048576 seconds; for some applications, reference to the leftmost 32 bits of the clock may provide sufficient resolution.
2. Communication between systems is facilitated by establishing a standard time origin that is the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 a.m. Coordinated Universal Time (UTC) is recommended as this origin, and it is said to begin the standard epoch for the clock. This is also the epoch used when the TOD clock is synchronized to the external time reference (ETR), and, for this reason, the epoch is sometimes referred to as ETR time. The former term, Greenwich Mean Time (GMT), is now obsolete and has been replaced with the more precise UTC.
3. Historically, one of the most important uses of standard time has been for navigation. Prior to 1972, standard time, then called GMT, was defined to have a variable-length second and was synchronized to within 100 milliseconds of the rotational position of the earth. Synchronization was accomplished by occasional changes in the length of the second, typically in parts per billion, and also by occasional insertion and deletion of small increments of time, typically 50 or 100 milliseconds. Beginning in 1972, a new standard time scale, called UTC, was defined to have a fixed-length second and be kept synchronized to within 900 milliseconds of the rotational position of the earth by means of occasional adjustments of exactly one second called a leap second. The change from GMT to UTC occurred between the last second of the day on December 31, 1971 and the first second of the day on January 1, 1972 and included insertion of 107.758 milliseconds in the standard time scale to make UTC exactly 10 seconds behind International Atomic Time (TAI). For reasons of simplicity in this document, the term UTC is sometimes extrapolated backward before 1972 by assuming no time adjustments in that time scale before 1972. For the same reasons, conversion between ETR time and UTC does not take into consideration the time adjustments prior to 1972, and, thus, ETR time differs from TAI by a fixed amount of 10 seconds. Because of the occurrence of 22 leap seconds, UTC now is behind TAI by 32 seconds.
4. A program using the clock value as a time-of-day and calendar indication must be consistent with the programming support under which the program is to be executed. If the programming support uses the standard epoch, bit 0 of the clock remains one through the years 1972-2041. (Bit 0 turned on at 11:56:53.685248 (UTC) May 11, 1971.) Ordinarily, testing bit 0 for a one is sufficient to



LzLabs GmbH. Ex. 1008-142

determine if the clock value is in the standard epoch.

5. In converting to or from the current date or time, the programming support must take into account that "leap seconds" have been inserted or deleted because of time-correction standards. When the TOD clock has been set correctly to a time within the standard epoch, the sum of the accumulated leap seconds must be subtracted from the clock time to determine UTC time.

6. Because of the limited accuracy of manually setting the clock value, the rightmost bit positions of the clock, expressing fractions of a second, are normally not valid as indications of the time of day. However, they permit elapsed-time measurements of high resolution.

7. The following chart shows the time interval between instants at which various bit positions of the TOD clock are stepped. This time value may also be considered as the weighted time value that the bit, when one, represents.

| TOD-Clock Bit | Stepping Interval | | | |
|---|---|---|---|---|
| | Days | Hours | Min. | Seconds |
| 51 | | | | 0.000 001 |
| 47 | | | | 0.000 016 |
| 43 | | | | 0.000 256 |
| 39 | | | | 0.004 096 |
| 35 | | | | 0.065 536 |
| 31 | | | | 1.048 576 |
| 27 | | | | 16.777 216 |
| 23 | | | 4 | 28.435 456 |
| 19 | | 1 | 11 | 34.967 296 |
| 15 | | 19 | 5 | 19.476 736 |
| 11 | 12 | 17 | 25 | 11.627 776 |
| 7 | 203 | 14 | 43 | 6.044 416 |
| 3 | 3257 | 19 | 29 | 36.710 656 |

8. The following chart shows the TOD clock setting for 00:00:00 (0 am), UTC time, for several dates: January 1, 1900, January 1, 1972, and for that instant in time just after each of the 22 leap seconds that have occurred through November, 2000. Each of these leap seconds was inserted in the UTC time scale beginning at 23:59:60 UTC of the day previous to the one listed and ending at 00:00:00 UTC of the day listed.

| Year | Mth | Day | Leap Sec. | Clock Setting (Hex) |
|---|---|---|---|---|
| 1900 | 1 | 1 | | 0000 0000 0000 0000 |
| 1972 | 1 | 1 | | 8126 D60E 4600 0000 |
| 1972 | 7 | 1 | 1 | 820B A981 1E24 0000 |
| 1973 | 1 | 1 | 2 | 82F3 00AE E248 0000 |
| 1974 | 1 | 1 | 3 | 84BD E971 146C 0000 |
| 1975 | 1 | 1 | 4 | 8688 D233 4690 0000 |
| 1976 | 1 | 1 | 5 | 8853 BAF5 78B4 0000 |
| 1977 | 1 | 1 | 6 | 8A1F E595 20D8 0000 |
| 1978 | 1 | 1 | 7 | 8BEA CE57 52FC 0000 |
| 1979 | 1 | 1 | 8 | 8DB5 B719 8520 0000 |
| 1980 | 1 | 1 | 9 | 8F80 9FDB B744 0000 |
| 1981 | 7 | 1 | 10 | 9230 5C0F CD68 0000 |
| 1982 | 7 | 1 | 11 | 93FB 44D1 FF8C 0000 |
| 1983 | 7 | 1 | 12 | 95C6 2D94 31B0 0000 |
| 1985 | 7 | 1 | 13 | 995D 40F5 17D4 0000 |
| 1988 | 1 | 1 | 14 | 9DDA 69A5 57F8 0000 |
| 1990 | 1 | 1 | 15 | A171 7D06 3E1C 0000 |
| 1991 | 1 | 1 | 16 | A33C 65C8 7040 0000 |
| 1992 | 7 | 1 | 17 | A5EC 21FC 8664 0000 |
| 1993 | 7 | 1 | 18 | A7B7 0ABE B888 0000 |
| 1994 | 7 | 1 | 19 | A981 F380 EAAC 0000 |
| 1996 | 1 | 1 | 20 | AC34 336F ECD0 0000 |
| 1997 | 7 | 1 | 21 | AEE3 EFA4 02F4 0000 |
| 1999 | 1 | 1 | 22 | B196 2F93 0518 0000 |

9. The stepping value of TOD-clock bit position 63, if implemented, is $2^{-12}$ microseconds, or approximately 244 picoseconds. This value is called a clock unit.

   The following chart shows various time intervals in clock units expressed in hexadecimal notation.

| Interval | Clock Units (Hex) |
|---|---|
| 1 microsecond | 1000 |
| 1 millisecond | 3E 8000 |
| 1 second | F424 0000 |
| 1 minute | 39 3870 0000 |
| 1 hour | D69 3A40 0000 |
| 1 day | 1 41DD 7600 0000 |
| 365 days | 1CA E8C1 3E00 0000 |
| 366 days | 1CC 2A9E B400 0000 |
| 1,461 days* | 72C E4E2 6E00 0000 |

* Number of days in four years, including a leap year. Note that the year 1900 was not a leap year. Thus, the four-year span starting in 1900 has only 1,460 days.

10. The charts in notes 6-8 are useful when examining the value stored by STORE CLOCK. Similar charts for use when exam-


LzLabs GmbH. Ex. 1008-143

ining the value stored by STORE CLOCK EXTENDED are in programming notes at the end of the definition of that instruction.

11. In a multiprocessing configuration, after the TOD clock is set and begins running, the program should delay activity for $2^{20}$ microseconds (1.048576 seconds) to ensure that the CPU-timer, clock-comparator, and TOD-clock-sync-check interruption conditions are recognized by the CPU.
12. Due to the sequencing rules for the results of STORE CLOCK and STORE CLOCK EXTENDED, the execution of STORE CLOCK may be considerably slower than that of STORE CLOCK EXTENDED on models that increment a bit position of the TOD clock to the right of position 63.
13. Uniqueness of TOD-clock values can be extended to apply to processors in separate configurations by including a configuration identification in the TOD programmable field.
14. At some time in the future, new models will use a carry from bit position 0 of the TOD clock to increment an additional eight-bit binary counter. STORE CLOCK EXTENDED will store the contents of this counter in byte position 0 of its storage operand. A variation of SET CLOCK will set the counter, as well as the TOD clock. Variations of SET CLOCK COMPARATOR and STORE CLOCK COMPARATOR will manipulate a comparable byte at the left of the clock comparator. These actions will allow the TOD clock to continue to measure time within the standard epoch after the current 143-year limit caused by a carry from bit position 0 has been exceeded, and they will allow continued use of the clock comparator. It may be desired to have programs that process 16-byte STORE CLOCK EXTENDED operands take these future developments into account.

## TOD-Clock Synchronization

In a configuration with more than one CPU, each CPU may have a separate TOD clock, or more than one CPU may share a TOD clock, depending on the model. In all cases, each CPU has access to a single clock.

The TOD-clock-synchronization facility, in conjunction with a clock-synchronization program, makes it possible to provide the effect of all CPUs in a multiprocessing configuration sharing a single TOD clock. The result is such that, to all programs storing the TOD-clock value, it appears that all CPUs in the configuration read the same TOD clock. The TOD-clock-synchronization facility provides these functions in such a way that even though the number of CPUs sharing a TOD clock is model-dependent, a single model-independent clock-synchronization routine can be written. The following functions are provided:

- Synchronizing the stepping rates for all TOD clocks in the configuration.
- Comparing bits 32 through the rightmost incremented bit of each clock in the configuration. An unequal condition is signaled by an external interruption with the interruption code 1003 hex, indicating the TOD-clock-sync-check condition. When there is only one clock in the configuration, this comparison may alternatively be done by comparing to bits of the ETR, in which case an unequal condition is indicated by an external-damage machine-check-interruption condition. The machine-check-interruption condition may not be recognized for up to 1.048576 seconds ($2^{20}$ microseconds) after the unequal condition occurs.
- Setting a TOD clock to the stopped state.
- Causing a stopped clock, with the TOD-clock-sync-control bit set to one, to start incrementing when bits 32 through the rightmost incremented bit of any running clock in the configuration are incremented to zero or, when the clock is the only clock in the configuration, when an ETR signal occurs. This permits the program to synchronize all clocks to any particular clock or to the ETR without requiring special operator action to select a “master clock” as the source of the clock-synchronization pulses.

**Programming Notes:**

1. TOD-clock synchronization provides for synchronizing and checking only bits 32 through the rightmost incremented bit of the TOD clock. When more than one clock exists in the configuration, the program must check for synchronization of the leftmost bits and must communicate the leftmost bit values from one CPU to another in order to correctly set the



LzLabs GmbH. Ex. 1008-144

TOD-clock contents. When a single clock exists in the configuration and it is synchronized with the ETR, bits 0-31 of the clock may be set different from those of the ETR.

2. The TOD-clock-sync-check external interruption can be used to determine the number of TOD clocks in the configuration.

## Clock Comparator

The clock comparator provides a means of causing an interruption when the TOD-clock value exceeds a value specified by the program.

In a configuration with more than one CPU, each CPU has a separate clock comparator.

The clock comparator has the same format as the basic TOD clock. The clock comparator nominally consists of bits 0-47, which are compared with the corresponding bits of the TOD clock. In some models, higher resolution is obtained by providing more than 48 bits. The bits in positions provided in the clock comparator are compared with the corresponding bits of the clock. When the resolution of the clock is less than that of the clock comparator, the contents of the clock comparator are compared with the clock value as this value would be stored by executing STORE CLOCK.

The clock comparator causes an external interruption with the interruption code 1004 hex. A request for a clock-comparator interruption exists whenever either of the following conditions exists:

1. The TOD clock is running and the value of the clock comparator is less than the value in the compared portion of the clock, both values being considered unsigned binary integers. Comparison follows the rules of unsigned binary arithmetic.
2. The TOD clock is in the error state or the not-operational state.

A request for a clock-comparator interruption does not remain pending when the value of the clock comparator is made equal to or greater than that of the TOD clock or when the value of the TOD clock is made less than the clock-comparator value. The latter may occur as a result of the TOD clock either being set or wrapping to zero.

The clock comparator can be inspected by executing the instruction STORE CLOCK COMPARATOR and can be set to a specified value by executing the SET CLOCK COMPARATOR instruction.

The contents of the clock comparator are initialized to zero by initial CPU reset.

**Programming Notes:**

1. An interruption request for the clock comparator persists as long as the clock-comparator value is less than that of the TOD clock or as long as the TOD clock is in the error state or the not-operational state. Therefore, one of the following actions must be taken after an external interruption for the clock comparator has occurred and before the CPU is again enabled for external interruptions: the value of the clock comparator must be replaced, the TOD clock must be set, the TOD clock must wrap to zero, or the clock-comparator-subclass mask must be set to zero. Otherwise, loops of external interruptions are formed.
2. The instruction STORE CLOCK or STORE CLOCK EXTENDED may store a value which is greater than that in the clock comparator, even though the CPU is enabled for the clock-comparator interruption. This is because the TOD clock may be incremented one or more times between when instruction execution is begun and when the clock value is accessed. In this situation, the interruption occurs when the execution of STORE CLOCK or STORE CLOCK EXTENDED is completed.

## CPU Timer

The CPU timer provides a means for measuring elapsed CPU time and for causing an interruption when a specified amount of time has elapsed.

In a configuration with more than one CPU, each CPU has a separate CPU timer.

The CPU timer is a binary counter with a format which is the same as that of the basic TOD clock, except that bit 0 is considered a sign. The CPU timer nominally is decremented by subtracting a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is decremented at such a frequency that the rate of decrementing the CPU timer is the same as if a one were subtracted in


LzLabs GmbH. Ex. 1008-145

bit position 51 every microsecond. The resolution of the CPU timer is such that the stepping rate is comparable to the instruction-execution rate of the model.

The CPU timer requests an external interruption with the interruption code 1005 hex whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). The request does not remain pending when the CPU-timer value is changed to a nonnegative value.

When both the CPU timer and the TOD clock are running, the stepping rates are synchronized such that both are stepped at the same rate. Normally, decrementing the CPU timer is not affected by concurrent I/O activity. However, in some models the CPU timer may stop during extreme I/O activity and other similar interference situations. In these cases, the time recorded by the CPU timer provides a more accurate measure of the CPU time used by the program than would have been recorded had the CPU timer continued to step.

The CPU timer is decremented when the CPU is in the operating state or the load state. When the manual rate control is set to instruction step, the CPU timer is decremented only during the time in which the CPU is actually performing a unit of operation. However, depending on the model, the CPU timer may or may not be decremented when the TOD clock is in the error, stopped, or not-operational state.

Depending on the model, the CPU timer may or may not be decremented when the CPU is in the check-stop state.

The CPU timer can be inspected by executing the instruction STORE CPU TIMER and can be set to a specified value by executing the SET CPU TIMER instruction.

The CPU timer is set to zero by initial CPU reset.

**Programming Notes:**

1. The CPU timer in association with a program may be used both to measure CPU-execution time and to signal the end of a time interval on the CPU.
2. The time measured for the execution of a sequence of instructions may depend on the effects of such things as I/O interference, the availability of pages, and instruction retry. Therefore, repeated measurements of the same sequence on the same installation may differ.
3. The fact that a CPU-timer interruption does not remain pending when the CPU timer is set to a positive value eliminates the problem of an undesired interruption. This would occur if, between the time when the old value is stored and a new value is set, the CPU is disabled for CPU-timer interruptions and the CPU timer value goes from positive to negative.
4. The fact that CPU-timer interruptions are requested whenever the CPU timer is negative (rather than just when the CPU timer goes from positive to negative) eliminates the requirement for testing a value to ensure that it is positive before setting the CPU timer to that value.

   As an example, assume that a program being timed by the CPU timer is interrupted for a cause other than the CPU timer, external interruptions are disallowed by the new PSW, and the CPU-timer value is then saved by STORE CPU TIMER. This value could be negative if the CPU timer went from positive to negative since the interruption. Subsequently, when the program being timed is to continue, the CPU timer may be set to the saved value by SET CPU TIMER. A CPU-timer interruption occurs immediately after external interruptions are again enabled if the saved value was negative.

   The persistence of the CPU-timer-interruption request means, however, that after an external interruption for the CPU timer has occurred, the value of the CPU timer must be replaced, the value in the CPU timer must wrap to a positive value, or the CPU-timer-subclass mask must be set to zero before the CPU is again enabled for external interruptions. Otherwise, loops of external interruptions are formed.
5. The instruction STORE CPU TIMER may store a negative value even though the CPU is enabled for the interruption. This is because the CPU-timer value may be decremented one or more times between when instruction execution is begun and when the CPU timer is accessed. In this situation, the



LzLabs GmbH. Ex. 1008-146

interruption occurs when the execution of STORE CPU TIMER is completed.

# Externally Initiated Functions

## Resets

Five reset functions are provided:

- CPU reset
- Initial CPU reset
- Subsystem reset
- Clear reset
- Power-on reset

CPU reset provides a means of clearing equipment-check indications and any resultant unpredictability in the CPU state with the least amount of information destroyed. In particular, it is used to clear check conditions when the CPU state is to be preserved for analysis or resumption of the operation.

Initial CPU reset provides the functions of CPU reset together with initialization of the current PSW, saved PSW (for use by the set-architecture order of SIGNAL PROCESSOR), CPU timer, clock comparator, TOD programmable register, prefix, and control registers.

Subsystem reset provides a means for clearing floating interruption conditions as well as for invoking I/O-system reset.

Clear reset causes initial CPU reset and subsystem reset to be performed and, additionally, clears or initializes all storage locations and registers in all CPUs in the configuration, with the exception of the TOD clock. Such clearing is useful in debugging programs and in ensuring user privacy. Clear reset also releases all locks used by the PERFORM LOCKED OPERATION instruction. Clearing does not affect external storage, such as direct-access storage devices used by the control program to hold the contents of unaddressable pages.

CPU power-on reset causes initial CPU reset to be performed and clears the contents of general registers, access registers, and floating-point registers to zeros with valid checking-block code. Locks used by PERFORM LOCKED OPERATION and associated with the CPU are released unless they are held by a CPU already powered on. The power-on-reset sequences for the TOD clock, main storage, and the channel subsystem may be included as part of the CPU power-on sequence, or the power-on sequence for these units may be initiated separately.

CPU reset, initial CPU reset, subsystem reset, and clear reset may be initiated manually by using the operator facilities (see Chapter 12, “Operator Facilities”). Initial CPU reset is part of the initial-program-loading function. Figure 4-7 on page 4-38 summarizes how these four resets are manually initiated. Power-on reset is performed as part of turning power on. The reset actions are tabulated in Figure 4-8 on page 4-39. For information concerning what resets can be performed by the SIGNAL PROCESSOR instruction, see “Set Prefix” on page 4-46.


LzLabs GmbH. Ex. 1008-147

| Key Activated | Function Performed on[1] | | |
|---|---|---|---|
| | CPU on Which Key Was Activated | Other CPUs in Config | Remainder of Configuration |
| System-reset-normal key | CPU reset | CPU reset | Subsystem reset |
| System-reset-clear key | Clear reset[2] | Clear reset[2] | Clear reset[3] |
| Load-normal key | Initial CPU reset, followed by IPL | CPU reset | Subsystem reset |
| Load-clear key | Clear reset[2], followed by IPL | Clear reset[2] | Clear reset[3] |

**Explanation:**

[1] Activation of a system-reset or load key may change the configuration, including the connection with I/O, storage units, and other CPUs.

[2] Only the CPU elements of this reset apply.

[3] Only the non-CPU elements of this reset apply.

*Figure 4-7. Manual Initiation of Resets*



LzLabs GmbH. Ex. 1008-148

| Area Affected | Reset Function | | | | |
|---|---|---|---|---|---|
| | Sub-system Reset | CPU Reset | Initial CPU Reset | Clear Reset | Power-On Reset |
| CPU | U | S | S[1] | S[1] | S |
| PSW | U | U/V | C*[1] | C*[1] | C* |
| Saved PSW for use by SIGNAL PROCESSOR set-architecture order | U | U | C | C | C |
| Prefix | U | U/V | C | C | C |
| CPU timer | U | U/V | C | C | C |
| Clock comparator | U | U/V | C | C | C |
| TOD programmable register | U | U/V | C | C | C |
| Control registers | U | U/V | I | I | I |
| Access registers | U | U/V | U/V | C | C |
| General registers | U | U/V | U/V | C | C |
| Floating-point registers | U | U/V | U/V | C | C |
| Vector-facility registers | U | U/V | U/V | C | C |
| Storage keys | U | U | U | C | C[2] |
| Volatile main storage | U | U | U | C | C[2] |
| Nonvolatile main storage | U | U | U | C | U |
| Expanded storage | U[3] | U[3] | U[3] | U[3] | C[2] |
| TOD clock | U[4] | U[4] | U[4] | U[4] | T[2] |
| Floating interruption conditions | C | U | U | C | C[2] |
| I/O system | R | U | U | R | R[5] |
| PERFORM LOCKED OPERATION locks | U | U | U | RC | RP |

**Explanation:**

* Clearing the contents of the PSW to zero causes the PSW to be invalid.

[1] When the IPL sequence follows the reset function on that CPU, the CPU does not necessarily enter the stopped state, and the PSW is not necessarily cleared to zeros.

[2] When these units are separately powered, the action is performed only when the power for the unit is turned on.

[3] Access to change expanded storage at the time a reset function is performed may cause the contents of the 4K-byte block in expanded storage to be unpredictable. Access to examine expanded storage does not affect the contents of the expanded storage.

[4] Access to the TOD clock by means of STORE CLOCK at the time a reset function is performed does not cause the value of the TOD clock to be affected.

*Figure 4-8 (Part 1 of 2). Summary of Reset Actions*


LzLabs GmbH. Ex. 1008-149

**Explanation** (Continued):

5 When the channel subsystem is separately powered or consists of multiple elements which are separately powered, the reset action is applied only to those subchannels, channel paths, and I/O control units and devices on those paths associated with the element which is being powered on.

C The condition or contents are cleared. If the area affected is a field, the contents are set to zero with valid checking-block code.

I The state or contents are initialized. If the area affected is a field, the contents are set to the initial value with valid checking-block code.

R I/O-system reset is performed in the channel subsystem. As part of this reset, system reset is signaled to all I/O control units and devices attached to the channel subsystem.

RC All locks in the configuration are released.

RP All locks in the configuration are released except for ones held by CPUs already powered on.

S The CPU is reset; current operations, if any, are terminated; the ALB and TLB are cleared of entries; interruption conditions in the CPU are cleared; and the CPU is placed in the stopped state. The effect of performing the start function is unpredictable when the stopped state has been entered by means of a reset.

T The TOD clock is initialized to zero and validated; it enters the not-set state.

U The state, condition, or contents of the field remain unchanged. However, the result is unpredictable if an operation is in progress that changes the state, condition, or contents of the field at the time of reset.

U/V The contents remain unchanged, provided the field is not being changed at the time the reset function is performed. However, on some models the checking-block code of the contents may be made valid. The result is unpredictable if an operation is in progress that changes the contents of the field at the time of reset.

*Figure 4-8 (Part 2 of 2). Summary of Reset Actions*

## CPU Reset

CPU reset causes the following actions:

1. The execution of the current instruction or other processing sequence, such as an interruption, is terminated, and all program-interruption and supervisor-call-interruption conditions are cleared.
2. Any pending external-interruption conditions which are local to the CPU are cleared. Floating external-interruption conditions are not cleared.
3. Any pending machine-check-interruption conditions and error indications which are local to the CPU and any check-stop states are cleared. Floating machine-check-interruption conditions are not cleared. Any machine-



LzLabs GmbH. Ex. 1008-150

check condition which is reported to all CPUs in the configuration and which has been made pending to a CPU is said to be local to the CPU.

4. All copies of prefetched instructions or operands are cleared. Additionally, any results to be stored because of the execution of instructions in the current checkpoint interval are cleared.
5. The ART-lookaside buffer and translation-lookaside buffer are cleared of entries.
6. The CPU is placed in the stopped state after actions 1-5 have been completed. When the IPL sequence follows the reset function on that CPU, the CPU enters the load state at the completion of the reset function and does not necessarily enter the stopped state during the execution of the reset operation.

Registers, storage contents, and the state of conditions external to the CPU remain unchanged by CPU reset. However, the subsequent contents of the register, location, or state are unpredictable if an operation is in progress that changes the contents at the time of the reset. The saved PSW, which is the PSW saved during a change from the z/Architecture architectural mode to the ESA/390 architectural mode, is not affected by a CPU reset occurring in the ESA/390 mode. (See the definition of the set-architecture order of SIGNAL PROCESSOR on page 4-49 for a description of the use of the saved PSW.) A lock held by the CPU when executing PERFORM LOCKED OPERATION is not released by CPU reset.

When the reset function in the CPU is initiated at the time the CPU is executing an I/O instruction or is performing an I/O interruption, the current operation between the CPU and the channel subsystem may or may not be completed, and the resultant state of the associated channel-subsystem facility may be unpredictable.

**Programming Note:** Most operations which would change a state, a condition, or the contents of a field cannot occur when the CPU is in the stopped state. However, some signal-processor functions and some operator functions may change these fields. To eliminate the possibility of losing a field when CPU reset is issued, the CPU should be stopped, and no operator functions should be in progress.

## Initial CPU Reset

Initial CPU reset combines the CPU reset functions with the following clearing and initializing functions:

1. The contents of the current PSW, saved PSW (for use by the set-architecture order of SIGNAL PROCESSOR), prefix, CPU timer, clock comparator, and TOD programmable register are set to zero. When the IPL sequence follows the reset function on that CPU, the contents of the PSW are not necessarily set to zero.
2. The contents of the control registers are set to their initial value.

These clearing and initializing functions include validation.

Setting the current PSW to zero causes the PSW to be invalid, since PSW bit 12 must be one. Thus, if the CPU is placed in the operating state after a reset without first introducing a new PSW, a specification exception is recognized.

## Subsystem Reset

Subsystem reset operates only on those elements in the configuration which are not CPUs. It performs the following actions:

1. I/O-system reset is performed by the channel subsystem (see “I/O-System Reset” on page 17-13).
2. All floating interruption conditions in the configuration are cleared.

As part of I/O-system reset, pending I/O-interruption conditions are cleared, and system reset is signaled to all control units and devices attached to the channel subsystem (see “I/O-System Reset” on page 17-13). The effect of system reset on I/O control units and devices and the resultant control-unit and device state are described in the appropriate System Library publication for the control unit or device. A system reset, in general, resets only those functions in a shared control unit or device that are associated with the particular channel path signaling the reset.


LzLabs GmbH. Ex. 1008-151

## Clear Reset

Clear reset combines the initial-CPU-reset function with an initializing function which causes the following actions:

1. The access, general, and floating-point registers of those CPUs which are in the configuration are set to zero.
2. The registers (vector-status register, vector-mask register, vector-activity count, and all vector registers) of those vector facilities, if any, which are in the configuration are cleared to zero with valid checking-block code.
3. The contents of the main storage in the configuration and the associated storage keys are set to zero with valid checking-block code.
4. The locks used by any CPU in the configuration when executing the PERFORM LOCKED OPERATION instruction are released.
5. A subsystem reset is performed.

Validation is included in setting registers and in clearing storage and storage keys.

**Programming Notes:**

1. For the CPU-reset operation not to affect the contents of fields that are to be left unchanged, the CPU must not be executing instructions and must be disabled for all interruptions at the time of the reset. Except for the operation of the CPU timer and for the possibility of a machine-check interruption occurring, all CPU activity can be stopped by placing the CPU in the wait state and by disabling it for I/O and external interruptions. To avoid the possibility of causing a reset at the time that the CPU timer is being updated or a machine-check interruption occurs, the CPU must be in the stopped state.
2. CPU reset, initial CPU reset, subsystem reset, and clear reset do not affect the value and state of the TOD clock.
3. The conditions under which the CPU enters the check-stop state are model-dependent and include malfunctions that preclude the completion of the current operation. Hence, if CPU reset or initial CPU reset is executed while the CPU is in the check-stop state, the contents of the PSW, registers, and storage locations, including the storage keys and the storage location accessed at the time of the error, may have unpredictable values, and, in some cases, the contents may still be in error after the check-stop state is cleared by these resets. In this situation, a clear reset is required to clear the error.

## Power-On Reset

The power-on-reset function for a component of the machine is performed as part of the power-on sequence for that component.

The power-on sequences for the TOD clock, vector facility, main storage, expanded storage, and channel subsystem may be included as part of the CPU power-on sequence, or the power-on sequence for these units may be initiated separately. The following sections describe the power-on resets for the CPU, TOD clock, vector facility, main storage, expanded storage, and channel subsystem. See also Chapter 17, "I/O Support Functions," and the appropriate System Library publication for the channel subsystem, control units, and I/O devices.

***CPU Power-On Reset:*** The power-on reset causes initial CPU reset to be performed and may or may not cause I/O-system reset to be performed in the channel subsystem. The contents of general registers, access registers, and floating-point registers are cleared to zeros with valid checking-block code. Locks used by PERFORM LOCKED OPERATION and associated with the CPU are released unless they are held by a CPU already powered on.

***TOD-Clock Power-On Reset:*** The power-on reset causes the value of the TOD clock to be set to zero with valid checking-block code and causes the clock to enter the not-set state.

***Vector-Facility Power-On Reset:*** The power-on reset causes the registers of the vector facility (vector-status register, vector-mask register, vector-activity count, and all vector registers) to be cleared to zeros with valid checking-block code.

***Main-Storage Power-On Reset:*** For volatile main storage (one that does not preserve its contents when power is off) and for storage keys, power-on reset causes zeros with valid checking-block code to be placed in these fields. The contents of nonvolatile main storage, including the checking-block code, remain unchanged.



LzLabs GmbH. Ex. 1008-152

***Expanded-Storage Power-On Reset:*** The contents of expanded storage are cleared to zeros with valid checking-block code.

***Channel-Subsystem Power-On Reset:*** The channel-subsystem power-on reset causes I/O-system reset to be performed in the channel subsystem. (See "I/O-System Reset" on page 17-13.)

## Initial Program Loading

Initial program loading (IPL) provides a manual means for causing a program to be read from a designated device and for initiating execution of that program.

Some models may provide additional controls and indications relating to IPL; this additional information is specified in the System Library publication for the model.

IPL is initiated manually by setting the load-unit-address controls to a four-digit number to designate an input device and by subsequently activating the load-clear or load-normal key for a particular CPU. In the description which follows, the term "this CPU" refers to the CPU in the configuration for which the load-clear or load-normal key was activated.

Activating the load-clear key causes a clear reset to be performed on the configuration.

Activating the load-normal key causes an initial CPU reset to be performed on this CPU, CPU reset to be propagated to all other CPUs in the configuration, and a subsystem reset to be performed on the remainder of the configuration.

In the loading part of the operation, after the resets have been performed, this CPU then enters the load state. This CPU does not necessarily enter the stopped state during the execution of the reset operations. The load indicator is on while the CPU is in the load state.

Subsequently, a channel-program read operation is initiated from the I/O device designated by the load-unit-address controls. The effect of executing the channel program is as if a format-0 CCW beginning at absolute storage location 0 specified a read command with the modifier bits zeros, a data address of zero, a byte count of 24, the chain-command and SLI flags ones, and all other flags zeros.

The details of the channel-subsystem portion of the IPL operation are defined in "Initial Program Loading" on page 17-17.

When the IPL I/O operation is completed successfully, the subsystem-identification word for the IPL device is stored in absolute storage locations 184-187, zeros are stored in absolute storage locations 188-191, and a new PSW is loaded from absolute storage locations 0-7. If the PSW loading is successful and no machine malfunctions are detected, this CPU leaves the load state, and the load indicator is turned off. If the rate control is set to the process position, the CPU enters the operating state, and the CPU operation proceeds under control of the new PSW. If the rate control is set to the instruction-step position, the CPU enters the stopped state, with the manual indicator on, after the new PSW is loaded.

If the IPL I/O operation or the PSW loading is not completed successfully, the CPU remains in the load state, and the load indicator remains on. The contents of absolute storage locations 0-7 are unpredictable.

## Store Status

The store-status operation places the contents of the CPU registers, except for the TOD clock, in assigned storage locations and in a store-status extended save area. If z/Architecture is installed, an architectural-mode identification of all zeros is also stored.

The store-status operation can be initiated manually by use of the store-status key (see Chapter 12, "Operator Facilities"). The operation can also be initiated at the addressed CPU by executing SIGNAL PROCESSOR, specifying the stop-and-store-status order.

Figure 4-9 on page 4-44 lists the fields that are stored in assigned storage locations, their lengths, and their locations in main storage.


LzLabs GmbH. Ex. 1008-153

| Field | Length in Bytes | Absolute Address |
|---|---|---|
| Architectural-mode id[1] | 1 | 163 |
| CPU timer | 8 | 216 |
| Clock comparator | 8 | 224 |
| Current PSW | 8 | 256 |
| Prefix | 4 | 264 |
| Access registers 0-15 | 64 | 288 |
| Fl-pt registers 0, 2, 4, 6 | 32 | 352 |
| General registers 0-15 | 64 | 384 |
| Control registers 0-15 | 64 | 448 |

**Explanation:**

[1] Stored only if z/Architecture is installed.

*Figure 4-9. Assigned Storage Locations for Store Status*

When the basic-floating-point-extensions facility is installed, the extended-save-area control, bit 2 of control register 14, is one, and bits 1-19 of the word at absolute locations 212-215 are not all zeros, then other fields are stored in a store-status extended save area. Figure 4-10 lists the fields that are stored, their lengths, and their offsets within the area. Bytes 144-4095 of the extended save area remain unchanged.

| Field | Length in Bytes | Byte Offset |
|---|---|---|
| Fl-pt registers 0-15 | 128 | 0 |
| Fl-pt-control register | 4 | 128 |
| Reserved (zeros stored) | 12 | 132 |
| Unchanged | 3,952 | 144 |

*Figure 4-10. Store-Status Extended Save Area*

The address of the store-status extended save area is formed by appending 12 zeros to the right of bits 1-19 of the word at absolute locations 212-215. This address is treated as a 31-bit absolute address. If the 4,096-byte block of storage at the address is not available in the configuration, or if bits 1-19 of locations 212-215 are all zeros, storing into the extended-save area is not performed.

During the storing into the assigned storage locations and the extended save area, the contents of the registers are not changed. If a machine error is encountered during the operation, the CPU enters the check-stop state.

Execution of SIGNAL PROCESSOR specifying the store-status-at-address order or the store-extended-status-at-address order causes some or all of the same status information to be stored by the addressed CPU at designated locations. See “Signal-Processor Orders” on page 4-45.

## Multiprocessing

The multiprocessing facility provides for the interconnection of CPUs, via a common main storage, in order to enhance system availability and to share data and resources. The multiprocessing facility includes the following facilities:

- Shared main storage
- CPU-to-CPU interconnection
- TOD-clock synchronization

Associated with these facilities are two external-interruption conditions (TOD-clock-sync check and malfunction alert), which are described in Chapter 6, “Interruptions”; and control-register positions for the TOD-clock-sync-control bit and for the masks for the external-interruption conditions, which are listed in “Control Registers” on page 4-6.

The channel subsystem, including all subchannels, in a multiprocessing configuration can be accessed by all CPUs in the configuration. I/O-interruption conditions are floating and can be accepted by any CPU in the configuration.

## Shared Main Storage

The shared-main-storage facility permits more than one CPU to have access to common main-storage locations. All CPUs having access to a common main-storage location have access to the entire 4K-byte block containing that location and to the associated storage key. The channel subsystem and all CPUs in the configuration refer to a shared main-storage location using the same absolute address.



LzLabs GmbH. Ex. 1008-154

## CPU-Address Identification

Each CPU has a 16-bit unsigned binary integer assigned, called its CPU address. A CPU address uniquely identifies one CPU within a configuration. The CPU is designated by specifying this address in the CPU-address field of SIGNAL PROCESSOR. The CPU signaling a malfunction alert, emergency signal, or external call is identified by storing this address in the CPU-address field with the interruption. The CPU address is assigned during system installation and is not changed as a result of reconfiguration changes. The program can determine the address of the CPU by using STORE CPU ADDRESS.

## CPU Signaling and Response

The CPU-signaling-and-response facility consists of SIGNAL PROCESSOR and a mechanism to interpret and act on several order codes. The facility provides for communications among CPUs, including transmitting, receiving, and decoding a set of assigned order codes; initiating the specified operation; and responding to the signaling CPU. A CPU can address SIGNAL PROCESSOR to itself. SIGNAL PROCESSOR is described in Chapter 10, "Control Instructions."

## Signal-Processor Orders

The signal-processor orders are specified in bit positions 24-31 of the second-operand address of SIGNAL PROCESSOR and are encoded as shown in Figure 4-11.

| Code (Hex) | Order |
|---|---|
| 00 | Unassigned |
| 01 | Sense |
| 02 | External call |
| 03 | Emergency signal |
| 04 | Start |
| 05 | Stop |
| 06 | Restart |
| 07 | Unassigned |
| 08 | Unassigned |
| 09 | Stop and store status |
| 0A | Unassigned |
| 0B | Initial CPU reset |
| 0C | CPU reset |
| 0D | Set prefix |
| 0E | Store status at address |
| 0F-10 | Unassigned |
| 11 | Store extended status at address |
| 12 | Set architecture |
| 13-FF | Unassigned |

*Figure 4-11. Encoding of Orders*

The orders are defined as follows:

***Sense:*** The addressed CPU presents its status to the issuing CPU (see "Status Bits" on page 4-51 for a definition of the bits). No other action is caused at the addressed CPU. The status, if not all zeros, is stored in the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction, and condition code 1 is set; if all status bits are zeros, condition code 0 is set.

***External Call:*** An external-call external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of SIGNAL PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. Only one external-call condition can be kept pending in a CPU at a time. The order is effective only when the addressed CPU is in the stopped or the operating state.

***Emergency Signal:*** An emergency-signal external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of SIGNAL


LzLabs GmbH. Ex. 1008-155

PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. At any one time the receiving CPU can keep pending one emergency-signal condition for each CPU in the configuration, including the receiving CPU itself. The order is effective only when the addressed CPU is in the stopped or the operating state.

***Start:*** The addressed CPU performs the start function (see "Stopped, Operating, Load, and Check-Stop States" on page 4-1). The CPU does not necessarily enter the operating state during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped state. The effect of performing the start function is unpredictable when the stopped state has been entered by reset.

***Stop:*** The addressed CPU performs the stop function (see "Stopped, Operating, Load, and Check-Stop States" on page 4-1). The CPU does not necessarily enter the stopped state during the execution of SIGNAL PROCESSOR. The order is effective only when the CPU is in the operating state.

***Restart:*** The addressed CPU performs the restart operation (see "Restart Interruption" on page 6-46). The CPU does not necessarily perform the operation during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped or the operating state.

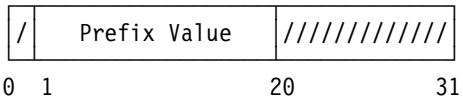
***Stop and Store Status:*** The addressed CPU performs the stop function, followed by the store-status operation (see "Store Status" on page 4-43). The CPU does not necessarily complete the operation, or even enter the stopped state, during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped or the operating state.

***Initial CPU Reset:*** The addressed CPU performs initial CPU reset (see "Resets" on page 4-37). The execution of the reset does not affect other CPUs and does not cause I/O to be reset. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

***CPU Reset:*** The addressed CPU performs CPU reset (see "Resets" on page 4-37). The execution of the reset does not affect other CPUs and does not cause I/O to be reset. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

***Set Prefix:*** The contents of bit positions 1-19 of the parameter register of the SIGNAL PROCESSOR instruction are treated as a prefix value, which replaces the contents of the prefix register of the addressed CPU. Bit 0 and bits 20-31 of the parameter register are ignored. The order is accepted only if the addressed CPU is in the stopped state, the value to be placed in the prefix register designates a location which is available in the configuration, and no other condition precludes accepting the order. Verification of the stopped state of the addressed CPU and of the availability of the designated storage is performed during execution of SIGNAL PROCESSOR. If accepted, the order is not necessarily completed during the execution of SIGNAL PROCESSOR.

The parameter register has the following format:




The set-prefix order is completed as follows:

- If the addressed CPU is not in the stopped state, the order is not accepted. Instead, bit 22 (incorrect state) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- The value to be placed in the prefix register of the addressed CPU is tested for the availability of the designated storage. The absolute address of a 4K-byte area of storage is formed by appending 12 zeros to the right of bits 1-19 of the parameter value. This address is treated as a 31-bit absolute address regardless of whether the sending and receiving CPUs are in the 24-bit or 31-bit addressing mode. The 4K-byte block of storage at this address is accessed. The access is not subject to protection, and the associated reference bit may or may not be set to one. If the block is not available in the configuration, the order is not accepted by the



LzLabs GmbH. Ex. 1008-156

addressed CPU, bit 23 (invalid parameter) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.

- The value is placed in the prefix register of the addressed CPU.
- The ALB and TLB of the addressed CPU are cleared of their contents.
- A serializing and checkpoint-synchronizing function is performed on the addressed CPU following insertion of the new prefix value.

***Store Status at Address:*** The contents of bit positions 1-22 of the parameter register of the SIGNAL PROCESSOR instruction are used as the origin of a 512-byte save area on a 512-byte boundary in absolute storage into which the status of the addressed CPU is stored. Bits 0 and 23-31 of the parameter register are ignored.

The order is accepted only if the addressed CPU is in the stopped state, the save-area origin designates a location that is available in the configuration, and no other condition precludes accepting the order. Verification of the stopped state of the addressed CPU and the availability of the designated storage is performed during the execution of SIGNAL PROCESSOR. If accepted, the order is not necessarily completed during the execution of SIGNAL PROCESSOR.

The parameter register has the following format:

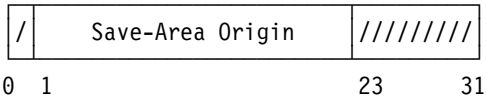



Figure 4-12 lists the fields that are stored in the save area, their lengths, and their offsets from the beginning of the area.

| Field | Length in Bytes | Byte Offset |
|---|---|---|
| CPU timer | 8 | 216 |
| Clock comparator | 8 | 224 |
| Current PSW | 8 | 256 |
| Prefix | 4 | 264 |
| Access registers 0-15 | 64 | 288 |
| Fl-pt registers 0, 2, 4, 6 | 32 | 352 |
| General registers 0-15 | 64 | 384 |
| Control registers 0-15 | 64 | 448 |

*Figure 4-12. Save-Area Locations for Store-Status-at-Address Order*

The store-status-at-address order is completed as follows.

- If the addressed CPU is not in the stopped state, the order is not accepted. Instead, bit 22 (incorrect state) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- The save area is tested for the availability of the designated storage. The absolute address of the save area is formed by appending nine zeros to the right of bits 1-22 of the parameter value. This address is treated as a 31-bit absolute address regardless of whether the sending and receiving CPUs are in the 24-bit or 31-bit addressing mode. The 512-byte block of storage at this address is accessed. The access is not subject to protection, and the associated reference bit may or may not be set to one. If the block is not available in the configuration, the order is not accepted by the addressed CPU, bit 23 (invalid parameter) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- Status of the addressed CPU is stored in the save area, as indicated in Figure 4-12. Bytes 0-215, 232-255, and 268-287 of the save area remain unchanged.
- A serialization and checkpoint-synchronization function is performed on the addressed CPU following storing of the status.

***Store Extended Status at Address:*** The contents of bit positions 1-22 of the parameter register of the SIGNAL PROCESSOR instruction are used as the origin of a 512-byte save area. Bits 0 and 23-31 of the parameter register are ignored. The contents of bit positions 1-19 of bytes 212-215 of the save area are used as the origin of a 4,096-byte extended save area. Bits 0 and 20-31 of bytes 212-215 are ignored.

Status of the addressed CPU is stored in the designated save area and extended save area.

The order is accepted only if the basic-floating-point-extensions facility is installed, the addressed CPU is in the stopped state, the save-area and extended-save-area origins designate locations that are available in the configuration, the


LzLabs GmbH. Ex. 1008-157

extended-save-area origin is not 0, and no other condition precludes accepting the order. Verification of the presence of the facility, the stopped state of the addressed CPU, and the availability of the designated storage is performed during the execution of SIGNAL PROCESSOR. If accepted, the order is not necessarily completed during the execution of SIGNAL PROCESSOR.

The parameter register has the following format:

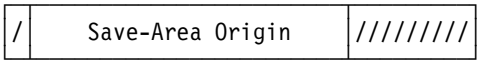



Figure 4-13 lists the fields in the save area, their lengths, and their offsets from the beginning of the area. The field in byte positions 212-215 is provided by the program. The other fields are stored during the execution of the operation specified by the order.

| Field | Length in Bytes | Byte Offset |
|---|---|---|
| Extended-save-area address | 4 | 212 |
| CPU timer | 8 | 216 |
| Clock comparator | 8 | 224 |
| Current PSW | 8 | 256 |
| Prefix | 4 | 264 |
| Access registers 0-15 | 64 | 288 |
| Fl-pt registers 0, 2, 4, 6 | 32 | 352 |
| General registers 0-15 | 64 | 384 |
| Control registers 0-15 | 64 | 448 |

*Figure 4-13. Save-Area Locations for Store-Extended-Status-at-Address Order*

The extended-save-area address in bytes 212-215 of the save area has the following format:

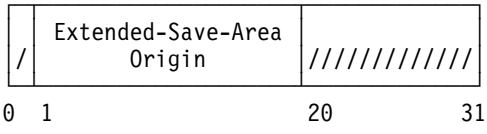



Figure 4-14 lists the fields that are stored in the extended-save area, their lengths, and their offsets from the origin of the area.

| Field | Length in Bytes | Byte Offset |
|---|---|---|
| Fl-pt registers 0-15 | 128 | 0 |
| Fl-pt-control register | 4 | 128 |
| Reserved (zeros stored) | 12 | 132 |
| Unchanged | 3,952 | 144 |

*Figure 4-14. Extended-Save Area Locations for Store-Extended-Status-at-Address Order.*

The store-extended-status-at-address order is completed as follows.

- If the basic-floating-point-extensions facility is not installed, the order is not accepted. Instead, bit 30 (invalid order) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- If the addressed CPU is not in the stopped state, the order is not accepted. Instead, bit 22 (incorrect state) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- The save area is tested for the availability of the designated storage. The absolute address of the save area is formed by appending nine zeros to the right of bits 1-22 of the parameter value. This address is treated as a 31-bit absolute address regardless of whether the sending and receiving CPUs are in the 24-bit or 31-bit addressing mode. The 512-byte block of storage at this address is accessed. The access is not subject to protection, and the associated reference bit may or may not be set to one. If the block is not available in the configuration, the order is not accepted by the addressed CPU, bit 23 (invalid parameter) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- The extended save area is tested for having a nonzero address and for the availability of the designated storage. The absolute address of the extended save area is formed by appending 12 zeros to the right of bits 1-19 of bytes 212-215 of the save area. This address is treated as a 31-bit absolute address regardless of whether the sending and receiving CPUs are in the 24-bit or 31-bit addressing mode. If the address is not 0, the 4,096-byte block of storage at the address is accessed.



LzLabs GmbH. Ex. 1008-158

The access is not subject to protection, and the associated reference bit may or may not be set to one. If the address is 0 or the block is not available in the configuration, the order is not accepted by the addressed CPU, bit 23 (invalid parameter) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.

- Status of the addressed CPU is stored in the save area, as indicated in Figure 4-13 on page 4-48, and in the extended save area as indicated in Figure 4-14 on page 4-48. Bytes 0-211, 232-255, and 268-287 of the save area and bytes 144-4095 of the extended save area remain unchanged.
- A serialization and checkpoint-synchronization function is performed on the addressed CPU following storing of the status.

***Set Architecture:*** The contents of bit positions 24-31 of the parameter register are used as a code specifying an architectural mode to which all CPUs in the configuration are to be set: code 0 specifies the ESA/390 mode, and codes 1 and 2* specify the z/Architecture mode. Code 1 specifies that, for each of all CPUs in the configuration, the current ESA/390 PSW is to be transformed to a z/Architecture PSW. Code 2 specifies that the PSW of the CPU executing SIGNAL PROCESSOR is to be transformed to a z/Architecture PSW and that, for each of all other CPUs in the configuration, the PSW is to be set with the value of the saved PSW for that CPU. The setting of the PSW with the value of the saved PSW will restore the PSW that existed when the CPU was last in the z/Architecture mode, provided that the saved PSW has not been set to all zeros by a reset.

Bits 0-23 of the parameter register are ignored. The contents of the CPU-address register of the SIGNAL PROCESSOR instruction are ignored; all other CPUs in the configuration are considered to be addressed.

The order is accepted only if the z/Architecture architectural mode is installed, the code is 0, 1, or 2, the CPU is not already in the mode specified by the code, each of all other CPUs is in either the stopped or the check-stop state, and no other condition precludes accepting the order. If accepted, the order is completed by all CPUs during the execution of SIGNAL PROCESSOR. In no case can different CPUs be in different architectural modes.

The set-architecture order is completed as follows:

- If the z/Architecture architectural mode is not installed, the order is not accepted. Instead, bit 30 (invalid order) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- If the code in the parameter register is not 0, 1, or 2, or if the CPU is already in the architectural mode specified by the code, the order is not accepted. Instead, bit 23 (invalid parameter) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- If it is not true that all other CPUs in the configuration are in the stopped or check-stop state, the order is not accepted. Instead, bit 22 (incorrect state) of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- The architectural mode of all CPUs in the configuration is set as specified by the code.
- If the order changes the architectural mode from ESA/390 to z/Architecture and the code is 1, then, for each CPU in the configuration, the eight-byte current PSW is changed to a 16-byte PSW, and the bits of the 16-byte PSW are set as follows: bits 0-11 and 13-32 are set equal to the same bits of the eight-byte PSW, bit 12 and bits 33-96 are set to zeros, and bits 97-127 are set equal to bits 33-63 of the eight-byte PSW. Also, bit 19 of the ESA/390 prefix, which becomes bit 51 of the z/Architecture prefix, is set to zero.

  If the code is 2, the PSW of the CPU executing SIGNAL PROCESSOR and the prefix values of all CPUs are set as in the code-1 case. For each of all other CPUs in the configuration, the PSW is set with the value of a PSW saved when the CPU last went from the z/Architecture mode to the ESA/390 mode because of a set-architecture order with code 0 or a CPU reset due to activation of the load-normal key. However, the saved PSW has been set to all zeros if the CPU performed a


LzLabs GmbH. Ex. 1008-159

reset, other than CPU reset, either at the time of the architectural-mode transition or subsequently.

- If the order changes the architectural mode from z/Architecture to ESA/390, then, for each CPU in the configuration, (1) the current PSW, which is the updated PSW in the case of the CPU executing SIGNAL PROCESSOR, is saved, and (2) the 16-byte current PSW is changed to an eight-byte PSW by setting the bits of the eight-byte PSW as follows: bits 0-11 and 13-32 are set equal to the same bits of the 16-byte PSW, bit 12 is set to one, and bits 33-63 are set equal to bits 97-127 of the 16-byte PSW. Bit 51 of the z/Architecture prefix, which becomes bit 19 of the ESA/390 prefix, remains unchanged.
- The ALBs and TLBs of all CPUs in the configuration are cleared of their contents.
- A serialization and checkpoint-synchronization function is performed on all CPUs in the configuration.

If the order changes the architectural mode from z/Architecture to ESA/390 and the SIGNAL PROCESSOR instruction causes occurrence of an instruction-fetching PER event, only the rightmost 31 bits of the address of the instruction are stored in the ESA/390 PER-address field.

**Programming Notes:**

1. If the set-architecture order changes the architectural mode from z/Architecture to ESA/390 and bit 31 of the PSW is one, the PSW is invalid.
2. For a discussion of the relative performance of the SIGNAL PROCESSOR orders, see the programming note following the instruction SIGNAL PROCESSOR in Chapter 10, "Control Instructions."

# Conditions Determining Response

## Conditions Precluding Interpretation of the Order Code

The following situations preclude the initiation of the order. The sequence in which the situations are listed is the order of priority for indicating concurrently existing situations:

1. The access path to the addressed CPU is busy because a concurrently executed SIGNAL PROCESSOR is using the CPU-signaling-and-response facility. The CPU which is concurrently executing the instruction can be any CPU in the configuration other than this CPU, and the CPU address can be any address, including that of this CPU or an invalid address. The order is rejected. Condition code 2 is set.
2. The addressed CPU is not operational; that is, it is not provided in the installation, it is not in the configuration, it is in any of certain customer-engineer test modes, or its power is off. The order is rejected. Condition code 3 is set. This condition cannot arise as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.
3. One of the following conditions exists at the addressed CPU:
   a. A previously issued start, stop, restart, stop-and-store-status, set-prefix, store-status-at-address, or store-extended-status-at-address order has been accepted by the addressed CPU, and execution of the function requested by the order has not yet been completed.
   b. A manual start, stop, restart, or store-status function has been initiated at the addressed CPU, and the function has not yet been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

   If the currently specified order is sense, external call, emergency signal, start, stop, restart, stop and store status, set prefix, store status at address, store extended status at address, or set architecture, then the order is rejected, and condition code 2 is set. If the currently specified order is one of the reset orders, or an unassigned or not-implemented order, the order code is interpreted as described in "Status Bits" on page 4-51.
4. One of the following conditions exists at the addressed CPU:
   a. A previously issued initial-CPU-reset or CPU-reset order has been accepted by the addressed CPU, and execution of the function requested by the order has not yet been completed.



LzLabs GmbH. Ex. 1008-160

b. A manual-reset function has been initiated at the addressed CPU, and the function has not yet been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

If the currently specified order is sense, external call, emergency signal, start, stop, restart, stop and store status, set prefix, store status at address, store extended status at address, or set architecture, then the order is rejected, and condition code 2 is set. If the currently specified order is one of the reset orders, or an unassigned or not-implemented order, either the order is rejected and condition code 2 is set or the order code is interpreted as described in "Status Bits."

When any of the conditions described in items 3 and 4 exists, the addressed CPU is referred to as "busy." Busy is not indicated if the addressed CPU is in the check-stop state or when the operator-intervening condition exists. A CPU-busy condition is normally of short duration; however, the conditions described in item 3 may last indefinitely because of a string of interruptions. In this situation, however, the CPU does not appear busy to any of the reset orders.

When the conditions described in items 1 and 2 above do not apply and operator-intervening and receiver-check status conditions do not exist at the addressed CPU, reset orders may be accepted regardless of whether the addressed CPU has completed a previously accepted order. This may cause the previous order to be lost when it is only partially completed, making unpredictable whether the results defined for the lost order are obtained.

## Status Bits

Various status conditions are defined whereby the issuing and addressed CPUs can indicate their responses to the specified order. The status conditions and their bit positions in the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction are shown in Figure 4-15.

| Bit Position | Status Condition |
|---|---|
| 0 | Equipment check |
| 1-21 | Unassigned; zeros stored |
| 22 | Incorrect state |
| 23 | Invalid parameter |
| 24 | External-call pending |
| 25 | Stopped |
| 26 | Operator intervening |
| 27 | Check stop |
| 28 | Unassigned; zero stored |
| 29 | Inoperative |
| 30 | Invalid order |
| 31 | Receiver check |

*Figure 4-15. Status Conditions*

The status condition assigned to bit position 0, and to bit position 23 when the order is set architecture, is generated by the CPU executing SIGNAL PROCESSOR. The remaining status conditions are generated by the addressed CPU.

When the invalid-parameter condition exists for the set-architecture order, bit 23 of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, all other bits in the register are set to zeros, and condition code 1 is set. No other action is taken.

When the equipment-check condition exists, except when the invalid-parameter condition exists for the set-architecture order, bit 0 of the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction is set to one, unassigned bits of the status register are set to zeros, and the other status bits are unpredictable. In this case, condition code 1 is set independent of whether the access path to the addressed CPU is busy and independent of whether the addressed CPU is not operational, is busy, or has presented zero status.

When the access path to the addressed CPU is not busy and the addressed CPU is operational and does not indicate busy to the currently specified order, the addressed CPU presents its status to the issuing CPU. These status bits are of two types:

1. Status bits 22, 23 when the order is not set architecture, 24-27, and 29 indicate the presence of the corresponding conditions in the addressed CPU at the time the order code is received. Except in response to the sense


LzLabs GmbH. Ex. 1008-161

order, each condition is indicated only when the condition precludes the successful execution of the specified order, although invalid parameter is not necessarily indicated when any other precluding condition exists. In the case of sense, all existing status conditions are indicated; the operator-intervening condition is indicated if it precludes the execution of any installed order.

2. Status bits 30 and 31 indicate that the corresponding conditions were detected by the addressed CPU during reception of the order.

If the presented status is all zeros, the addressed CPU has accepted the order, and condition code 0 is set at the issuing CPU; if the presented status is not all zeros, the order has been rejected, the status is stored at the issuing CPU in the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction, zeros are stored in the unassigned bit positions of the register, and condition code 1 is set.

When the order is set architecture, "the addressed CPU" refers to each of the other CPUs in the configuration. Those CPUs, in an unpredictable order, are tested for a condition that causes setting of condition code 1, 2, or 3. Conditions are prioritized for a single CPU as if it were the only CPU addressed, but there is no prioritization across CPUs. If a condition is recognized, no further CPUs are tested, the condition code corresponding to the condition is set, and the execution of SIGNAL PROCESSOR is completed.

The status conditions are defined as follows:

***Equipment Check:*** This condition exists when the CPU executing the instruction detects equipment malfunctioning that has affected only the execution of this instruction and the associated order. The order code may or may not have been transmitted and may or may not have been accepted, and the status bits provided by the addressed CPU may be in error. This condition is not detected if the invalid-parameter condition for the set-architecture order is detected.

***Incorrect State:*** A set-prefix, store-status-at-address, or store-extended-status-at-address order has been rejected because the addressed CPU is not stopped, or a set-architecture order has been rejected because not all other CPUs are stopped or in the check-stop state. When applicable, this status is generated during execution of SIGNAL PROCESSOR and is indicated concurrently with other indications of conditions which preclude execution of the order, except that this status is not generated if an invalid-parameter condition exists for a set-architecture order.

***Invalid Parameter:*** This condition exists in two cases:

1. The parameter value supplied with a set-prefix, store-status-at-address, or store-extended-status-at-address order designates a storage location which is not available in the configuration. When applicable, this status is generated during execution of SIGNAL PROCESSOR, except that it is not necessarily generated when another condition precluding execution of the order also exists.
2. The parameter value supplied with a set-architecture order either is not 0 or 1 or specifies the current architectural mode. When applicable, this status is generated during execution of SIGNAL PROCESSOR, and no other status is generated.

***External Call Pending:*** This condition exists when an external-call interruption condition is pending in the addressed CPU because of a previously issued SIGNAL PROCESSOR order. The condition exists from the time an external-call order is accepted until the resultant external interruption has been completed or a CPU reset occurs. The condition may be due to the issuing CPU or another CPU. The condition, when present, is indicated only in response to sense and to external call.

***Stopped:*** This condition exists when the addressed CPU is in the stopped state. The condition, when present, is indicated only in response to sense. This condition cannot be reported as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

***Operator Intervening:*** This condition exists when the addressed CPU is executing certain operations initiated from local or remote operator facilities. The particular manually initiated operations that cause this condition to be present depend on the model and on the order specified. The operator-intervening condition may exist when the addressed CPU uses reloadable control storage to perform an order and the required



LzLabs GmbH. Ex. 1008-162

licensed internal code has not been loaded by the IML function. The operator-intervening condition, when present, can be indicated in response to all orders. Operator intervening is indicated in response to sense if the condition is present and precludes the acceptance of any of the installed orders. The condition may also be indicated in response to unassigned or uninstalled orders. This condition cannot arise as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

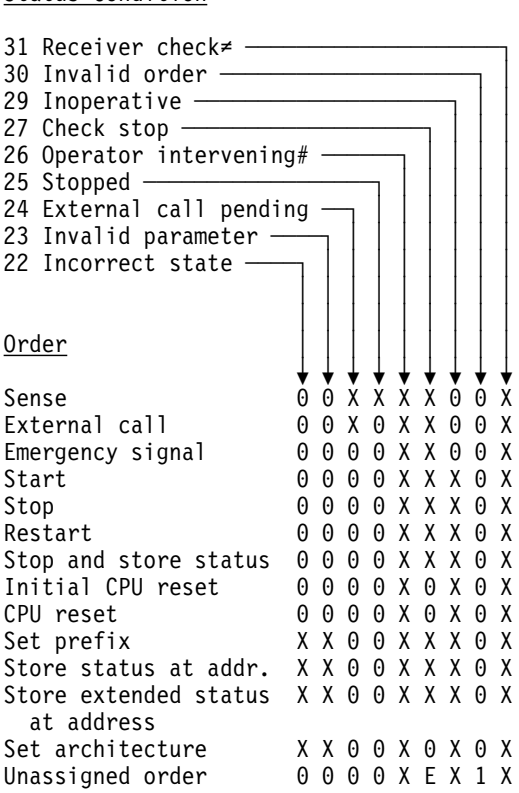
***Check Stop:*** This condition exists when the addressed CPU is in the check-stop state. The condition, when present, is indicated only in response to sense, external call, emergency signal, start, stop, restart, set prefix, store status at address, store extended status at address, and stop and store status. The condition may also be indicated in response to unassigned or uninstalled orders. This condition cannot be reported as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

***Inoperative:*** This condition indicates that the execution of the operation specified by the order code requires the use of a service processor which is inoperative. The failure of the service processor may have been previously reported by a service-processor-damage machine-check condition. The inoperative condition cannot occur for the sense, external-call, or emergency-signal order code.

***Invalid Order:*** This condition exists during the communications associated with the execution of SIGNAL PROCESSOR when an unassigned or uninstalled order code is decoded.

***Receiver Check:*** This condition exists when the addressed CPU detects malfunctioning of equipment during the communications associated with the execution of SIGNAL PROCESSOR. When this condition is indicated, the order has not been initiated, and, since the malfunction may have affected the generation of the remaining receiver status bits, these bits are not necessarily valid. A machine-check condition may or may not have been generated at the addressed CPU.

The following chart summarizes which status conditions are presented to the issuing CPU in response to each order code.




***Explanation:***

\# The current state of the operator-intervening condition may depend on the order code that is being interpreted.

≠ If a one is presented in the receiver-check bit position, the values presented in the other bit positions are not necessarily valid.

0 A zero is presented in this bit position regardless of the current state of this condition.

1 A one is presented in this bit position.

X A zero or a one is presented in this bit position, reflecting the current state of the corresponding condition.

E Either a zero or the current state of the corresponding condition is indicated.

If the presented status bits are all zeros, the order has been accepted, and the issuing CPU sets condition code 0. If one or more ones are presented, the order has been rejected, and the issuing CPU stores the status in the general register designated by the $R_1$ field of the SIGNAL PROCESSOR instruction and sets condition code 1.


LzLabs GmbH. Ex. 1008-163

**Programming Notes:**

1. All SIGNAL PROCESSOR orders except set architecture (which in effect is addressed to all other CPUs and affects all CPUs) can be addressed to this same CPU. The following are examples of functions obtained by a CPU addressing SIGNAL PROCESSOR to itself:

   a. *Sense* indicates whether an external-call condition is pending.

   b. *External call* and *emergency signal* cause the corresponding interruption conditions to be generated. *External call* can be rejected because of a previously generated external-call condition.

   c. *Start* sets condition code 0 and has no other effect.

   d. *Stop* causes the CPU to set condition code 0, take pending interruptions for which it is enabled, and enter the stopped state.

   e. *Restart* provides a means to store the current PSW.

   f. *Stop and store status* causes the machine to stop and store all current status.

2. Two CPUs can simultaneously execute SIGNAL PROCESSOR, with each CPU addressing the other. When this occurs, one CPU, but not both, can find the access path busy because of the transmission of the order code or status bits associated with SIGNAL PROCESSOR that is being executed by the other CPU. Alternatively, both CPUs can find the access path available and transmit the order codes to each other. In particular, two CPUs can simultaneously stop, restart, or reset each other.

3. To obtain status from another CPU which is in the check-stop state by means of the store-status-at-address or store-extended-status-at-address order, a CPU reset operation should first be used to bring the CPU to the stopped state. This reset order does not alter the status, and, depending on the nature of the malfunction, provides the best chance of establishing conditions in the addressed CPU which allow status to be obtained.



LzLabs GmbH. Ex. 1008-164

# Chapter 5. Program Execution


Instructions . . . . . 5-2
Operands . . . . . 5-3
Instruction Formats . . . . . 5-3
Register Operands . . . . . 5-6
Immediate Operands . . . . . 5-6
Storage Operands . . . . . 5-6
Address Generation . . . . . 5-7
Bimodal Addressing . . . . . 5-7
Sequential Instruction-Address Generation . 5-7
Operand-Address Generation . . . . . 5-8
Formation of the Intermediate Value . . 5-8
Formation of the Operand Address . . . 5-8
Branch-Address Generation . . . . . 5-9
Formation of the Intermediate Value . . 5-9
Formation of the Branch Address . . . . 5-9
Instruction Execution and Sequencing . . . . 5-9
Decision Making . . . . . 5-10
Loop Control . . . . . 5-10
Subroutine Linkage without the Linkage Stack . . . . . 5-10
Interruptions . . . . . 5-16
Types of Instruction Ending . . . . . 5-16
Completion . . . . . 5-16
Suppression . . . . . 5-16
Nullification . . . . . 5-17
Termination . . . . . 5-17
Interruptible Instructions . . . . . 5-17
Point of Interruption . . . . . 5-17
Unit of Operation . . . . . 5-17
Execution of Interruptible Instructions . 5-17
Condition-Code Alternative to Interruptibility . . . . . 5-19
Exceptions to Nullification and Suppression . . . . . 5-19
Storage Change and Restoration for DAT-Associated Access Exceptions 5-20
Modification of DAT-Table Entries . . . 5-20
Trial Execution for Editing Instructions and Translate Instruction . . . . . 5-21
Authorization Mechanisms . . . . . 5-21
Mode Requirements . . . . . 5-21
Extraction-Authority Control . . . . . 5-22
PSW-Key Mask . . . . . 5-22
Secondary-Space Control . . . . . 5-22
Subsystem-Linkage Control . . . . . 5-22
ASN-Translation Control . . . . . 5-22
Authorization Index . . . . . 5-23
Program-Call-Fast Control . . . . . 5-23
Access-Register and Linkage-Stack Mechanisms . . . . . 5-23
PC-Number Translation . . . . . 5-27
PC-Number Translation Control . . . . . 5-27
Control Register 0 . . . . . 5-27
Control Register 5 . . . . . 5-27
PC-Number Translation Tables . . . . . 5-28
Linkage-Table Entries . . . . . 5-28
Entry-Table Entries . . . . . 5-28
PC-Number-Translation Process . . . . . 5-30
Obtaining the Linkage-Table Designation . . . . . 5-31
Linkage-Table Lookup . . . . . 5-32
Entry-Table Lookup . . . . . 5-32
Recognition of Exceptions during PC-Number Translation . . . . . 5-32
Home Address Space . . . . . 5-33
Access-Register Introduction . . . . . 5-33
Summary . . . . . 5-34
Access-Register Functions . . . . . 5-34
Access-Register-Specified Address Spaces . . . . . 5-34
Access-Register Instructions . . . . . 5-41
Access-Register Translation . . . . . 5-42
Access-Register-Translation Control . . . . 5-42
Address-Space-Function Control . . . . 5-42
Control Register 2 . . . . . 5-43
Control Register 5 . . . . . 5-43
Control Register 8 . . . . . 5-43
Access Registers . . . . . 5-43
Access-Register-Translation Tables . . . . 5-44
Dispatchable-Unit-Control Table and Access-List Designations . . . . . 5-44
Access-List Entries . . . . . 5-46
Extended ASN-Second-Table Entries . 5-47
Access-Register-Translation Process . . . 5-48
Selecting the Access-List-Entry Token . 5-51
Obtaining the Primary or Secondary Segment-Table Designation . . . . . 5-51
Checking the First Byte of the ALET . . 5-51
Obtaining the Effective Access-List Designation . . . . . 5-51
Access-List Lookup . . . . . 5-51
Locating the ASN-Second-Table Entry . 5-52
Authorizing the Use of the Access-List Entry . . . . . 5-52
Checking for Access-List-Controlled Protection . . . . . 5-53
Obtaining the Segment-Table Designation from the ASN-Second-Table Entry . . . . . 5-53


© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-165

Recognition of Exceptions during Access-Register Translation . . . . 5-53
ART-Lookaside Buffer . . . . . 5-53
ALB Structure . . . . . 5-53
Formation of ALB Entries . . . . . 5-54
Use of ALB Entries . . . . . 5-54
Modification of ART Tables . . . . . 5-55
Subspace Groups . . . . . 5-55
Subspace-Group Tables . . . . . 5-56
Subspace-Group Dispatchable-Unit Control Table . . . . . 5-56
Subspace-Group ASN-Second-Table Entries . . . . . 5-57
Subspace-Replacement Operations . . . . 5-59
Linkage-Stack Introduction . . . . . 5-60
Summary . . . . . 5-60
Linkage-Stack Functions . . . . . 5-61
Transferring Program Control . . . . . 5-61
Branching Using the Linkage Stack . . 5-62
Adding and Retrieving Information . . . 5-63
Testing Authorization . . . . . 5-63
Program-Problem Analysis . . . . . 5-64
Extended Entry-Table Entries . . . . . 5-64
Linkage-Stack Operations . . . . . 5-66
Linkage-Stack-Operations Control . . . . . 5-68
Control Register 0 . . . . . 5-68
Control Register 15 . . . . . 5-68
Linkage Stack . . . . . 5-68
Entry Descriptors . . . . . 5-69
Header Entries . . . . . 5-70
Trailer Entries . . . . . 5-70
State Entries . . . . . 5-71
Stacking Process . . . . . 5-73
Locating Space for a New Entry . . . . 5-73
Forming the New Entry . . . . . 5-74
Updating the Current Entry . . . . . 5-75
Updating Control Register 15 . . . . . 5-75
Recognition of Exceptions during the Stacking Process . . . . . 5-75
Unstacking Process . . . . . 5-75
Locating the Current Entry and Processing a Header Entry . . . . . 5-76
Checking for a State Entry . . . . . 5-77
Restoring Information . . . . . 5-77
Updating the Preceding Entry . . . . . 5-77
Updating Control Register 15 . . . . . 5-77
Recognition of Exceptions during the Unstacking Process . . . . . 5-78
Sequence of Storage References . . . . . 5-78
Conceptual Sequence . . . . . 5-78
Overlapped Operation of Instruction Execution . . . . . 5-79
Divisible Instruction Execution . . . . . 5-79
Interlocks for Virtual-Storage References . 5-79
Interlocks between Instructions . . . . . 5-80
Interlocks within a Single Instruction . . 5-80
Instruction Fetching . . . . . 5-82
ART-Table and DAT-Table Fetches . . . . 5-83
Storage-Key Accesses . . . . . 5-84
Storage-Operand References . . . . . 5-85
Storage-Operand Fetch References . . 5-85
Storage-Operand Store References . . 5-85
Storage-Operand Update References . 5-86
Storage-Operand Consistency . . . . . 5-87
Single-Access References . . . . . 5-87
Multiple-Access References . . . . . 5-87
Block-Concurrent References . . . . . 5-88
Consistency Specification . . . . . 5-88
Relation between Operand Accesses . . . 5-90
Other Storage References . . . . . 5-90
Relation between Storage-Key Accesses . 5-90
Serialization . . . . . 5-91
CPU Serialization . . . . . 5-91
Channel-Program Serialization . . . . . 5-92

Normally, operation of the CPU is controlled by instructions in storage that are executed sequentially, one at a time, left to right in an ascending sequence of storage addresses. A change in the sequential operation may be caused by branching, LOAD PSW, interruptions, SIGNAL PROCESSOR orders, or manual intervention.

## Instructions

Each instruction consists of two major parts:

- An operation code (op code), which specifies the operation to be performed
- The designation of the operands that participate



LzLabs GmbH. Ex. 1008-166

## Operands

Operands can be grouped in three classes: operands located in registers, immediate operands, and operands in storage. Operands may be either explicitly or implicitly designated.

Register operands can be located in general, floating-point, access, or control registers, with the type of register identified by the op code. The register containing the operand is specified by identifying the register in a four-bit field, called the R field, in the instruction. For some instructions, an operand is located in an implicitly designated register, the register being implied by the op code.

Immediate operands are contained within the instruction, and the 8-bit, 16-bit, or 32-bit field containing the immediate operand is called the I field.

Operands in storage may have an implied length; be specified by a bit mask; be specified by a four-bit or eight-bit length specification, called the L field, in the instruction; or have a length specified by the contents of a general register. The addresses of operands in storage are specified by means of a format that uses the contents of a general register as part of the address. This makes it possible to:

1. Specify a complete address by using an abbreviated notation
2. Perform address manipulation using instructions which employ general registers for operands
3. Modify addresses by program means without alteration of the instruction stream
4. Operate independent of the location of data areas by directly using addresses received from other programs

The address used to refer to storage either is contained in a register designated by the R field in the instruction or is calculated from a base address, index, and displacement, specified by the B, X, and D fields, respectively, in the instruction.

When the CPU is in the access-register mode, a B or R field may designate an access register in addition to being used to specify an address.

To describe the execution of instructions, operands are designated as first and second operands and, in some cases, third operands.

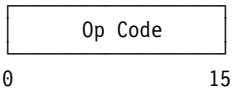
In general, two operands participate in an instruction execution, and the result replaces the first operand. However, CONVERT TO DECIMAL, TEST BLOCK, and instructions with “store” in the instruction name (other than STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK) use the second-operand address to designate a location in which to store. TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP may perform an update on the second operand. Except when otherwise stated, the contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

## Instruction Formats

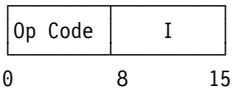
An instruction is one, two, or three halfwords in length and must be located in storage on a halfword boundary. Each instruction is in one of 18 basic formats: E, I, RR, RRE, RRF, RX, RXE, RXF, RS, RSE, RSL, RSI, RI, RIL, SI, S, SSE, and SS, with three variations of RRF, two of RS and RIL, and four of SS. See Figure 5-1 on page 5-4.


LzLabs GmbH. Ex. 1008-167

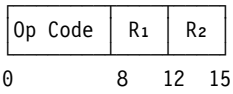
E Format



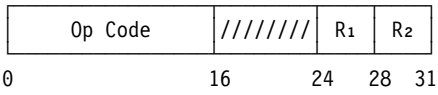

I Format



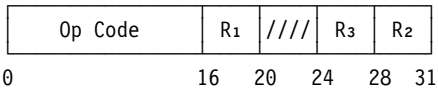

RR Format

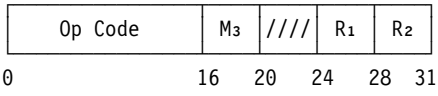


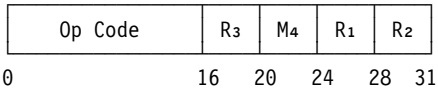

RRE Format



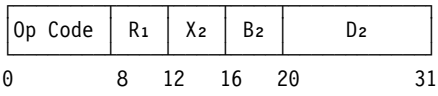

RRF Format




RX Format




RXE Format

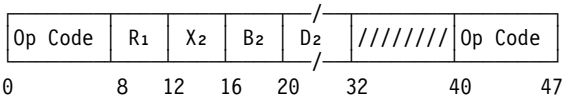
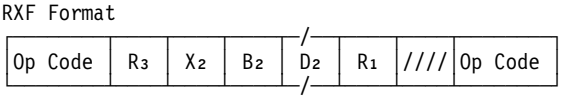
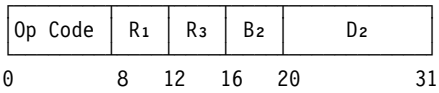


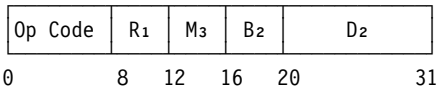

*Figure 5-1 (Part 1 of 3). Basic Instruction Formats*

RXF Format

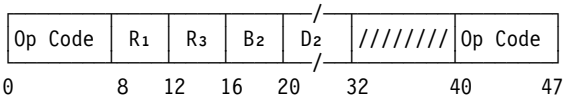



RS Format

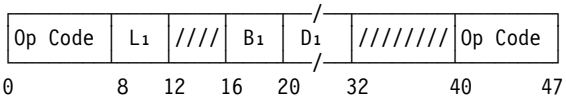


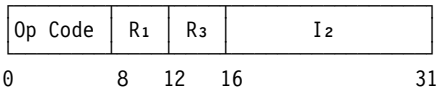

RSE Format



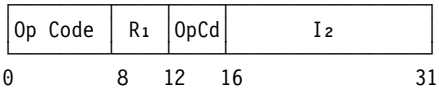

RSL Format



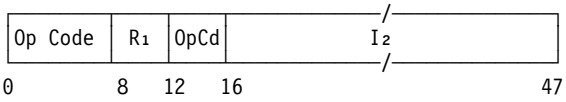

RSI Format




RI Format




RIL Format



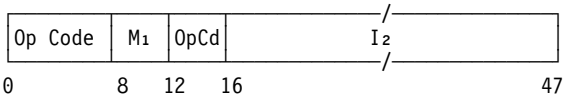




*Figure 5-1 (Part 2 of 3). Basic Instruction Formats*



LzLabs GmbH. Ex. 1008-168

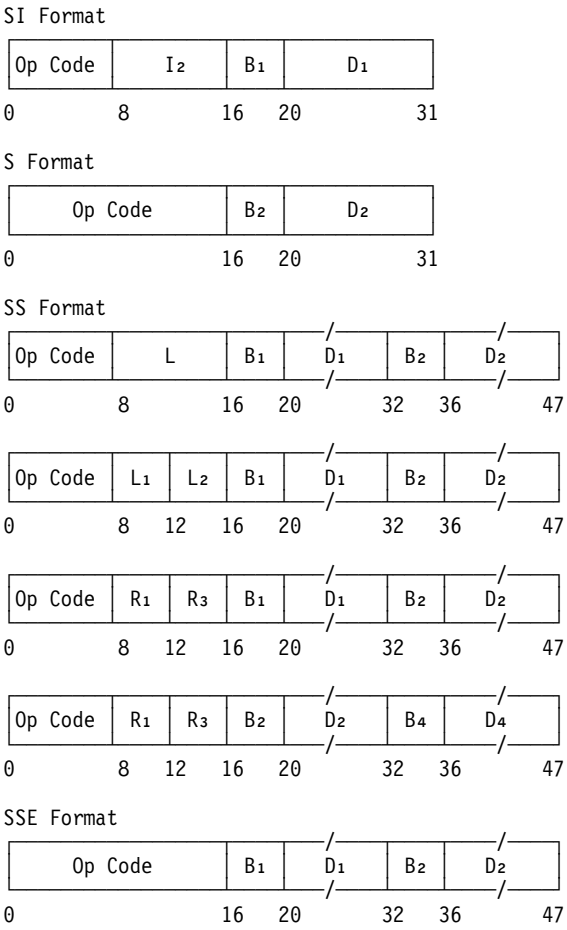



*Figure 5-1 (Part 3 of 3). Basic Instruction Formats*

Instruction fields shown in Figure 5-1 on page 5-4 as containing slashes (/) are currently unassigned. These fields in an instruction should contain zeros; otherwise, the program may not operate compatibly in the future.

Some instructions contain fields that vary slightly from the basic format, and in some instructions the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

Those instruction formats which are unique to instructions associated with the vector facility are described in the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207.

The format names indicate, in general terms, the classes of operands which participate in the operation and some details about fields:

- E denotes an operation using implied operands and an extended op-code field.
- I denotes an immediate operation
- RR denotes a register-and-register operation.
- RRE denotes a register-and-register operation and an extended op-code field.
- RRF denotes a register-and-register operation, an extended op-code field, and an additional R field, M field, or both.
- RX denotes a register-and-indexed-storage operation.
- RXE denotes a register-and-indexed-storage operation and an extended op-code field.
- RXF denotes a register-and-indexed-storage operation, an extended op-code field, and an additional R field.
- RS denotes a register-and-storage operation.
- RSE denotes a register-and-storage operation and an extended op-code field.
- RSL denotes a storage operation (with an instruction format derived from the RSE format).
- RSI denotes a register-and-immediate operation.
- RI denotes a register-and-immediate operation and an extended op-code field.
- RIL denotes a register-and-immediate operation, an extended op-code field, and a longer immediate field.
- SI denotes a storage-and-immediate operation.
- S denotes an operation using an implied operand and storage.
- SS denotes a storage-and-storage operation.
- SSE denotes a storage-and-storage operation and an extended op-code field.

In the RR, RX, RS, RSI, SI, and SS formats, the first byte of an instruction contains the op code. In the E, RRE, RRF, S, and SSE formats, the first two bytes of an instruction contain the op code, except that for some instructions in the S format, the op code is in only the first byte. In the RI and RIL formats, the op code is in the first byte and bit positions 12-15 of an instruction. In the RXE, RXF, RSE, and RSL formats, the op code is in the first byte and bits 40-47 of an instruction.

The first two bits of the first or only byte of the op code specify the length and format of the instruction, as follows:


LzLabs GmbH. Ex. 1008-169

| Bit Positions 0-1 | Instruction Length (in Halfwords) | Instruction Format |
|---|---|---|
| 00 | One | E/RR |
| 01 | Two | RX |
| 10 | Two | RRE/RRF/RX/RS/RSI/RI/SI/S |
| 11 | Three | RXE/RXF/RSE/RSL/RIL/SS/SSE |

In the format illustration for each individual instruction description, the op-code field or fields show the op code as hexadecimal digits within single quotes. The hexadecimal representation uses 0-9 for the binary codes 0000-1001 and A-F for the binary codes 1010-1111.

The remaining fields in the format illustration for each instruction are designated by code names, consisting of a letter and possibly a subscript number. The subscript number denotes the operand to which the field applies.

## Register Operands

In the RR, RRE, RRF, RX, RXE, RXF, RS, RSE, RSI, RI, and RIL formats, the contents of the register designated by the $R_1$ field are called the first operand. The register containing the first operand is sometimes referred to as the "first-operand location," and sometimes as "register $R_1$." In the RR, RRE, and RRF formats, the $R_2$ field designates the register containing the second operand, and the $R_2$ field may designate the same register as $R_1$. In the RRF, RXF, RS, RSE, and RSI formats, the use of the $R_3$ field depends on the instruction. In the RS format, the $R_3$ field may instead be an $M_3$ field specifying a mask.

The R field designates a general or access register in the general instructions, a general register in the control instructions, and a floating-point register in the floating-point instructions. However, in the instructions EXTRACT STACKED REGISTERS and LOAD ADDRESS EXTENDED, the R field designates both a general register and an access register, and, in the instructions LOAD CONTROL and STORE CONTROL, the R field designates a control register. (This paragraph refers only to register operands, not to the use of access registers in addressing storage operands.)

Unless otherwise indicated in the individual instruction description, the register operand is one register in length (32 bits for a general, access, or control register and 64 bits for a floating-point register), and the second operand is the same length as the first.

## Immediate Operands

In the I format, the contents of the eight-bit immediate-data field, the I field of the instruction, are directly used as the operand.

In the SI format, the contents of the eight-bit immediate-data field, the $I_2$ field of the instruction, are used directly as the second operand. The $B_1$ and $D_1$ fields specify the first operand, which is one byte in length.

In the RI format for the instructions ADD HALFWORD IMMEDIATE, COMPARE HALFWORD IMMEDIATE, LOAD HALFWORD IMMEDIATE, and MULTIPLY HALFWORD IMMEDIATE, the contents of the 16-bit $I_2$ field of the instruction are used directly as a signed binary integer; and for the instructions TEST UNDER MASK HIGH and TEST UNDER MASK LOW, the contents are used as a mask. The $R_1$ field specifies the first operand, which is one word in length.

For the relative-branch instructions in the RI and RSI formats, the contents of the 16-bit $I_2$ field are used as a signed binary integer designating a number of halfwords. This number, when added to the address of the branch instruction, specifies the branch address. In the RIL format, the $I_2$ field is 32 bits and is used in the same way.

## Storage Operands

The use of B and R fields to designate access registers to refer to storage operands is described in "Access-Register-Specified Address Spaces" on page 5-34.

In the RSL, SI, SS, and SSE formats, the contents of the general register designated by the $B_1$ field are added to the contents of the $D_1$ field to form the first-operand address. In the RS, RSE, S, SS, and SSE formats, the contents of the general register designated by the $B_2$ field are added to the contents of the $D_2$ field to form the second-operand address. In the RX, RXE, and RXF formats, the contents of the general registers designated by the $X_2$ and $B_2$ fields are added to the contents of the $D_2$ field to form the second-operand address.



LzLabs GmbH. Ex. 1008-170

When a general register contains a 24-bit or 32-bit length of a storage operand, the length is an unsigned binary integer, except that it is signed for COMPARE UNTIL SUBSTRING EQUAL, with a negative value treated as zero. Similarly, the contents of an L, $L_1$, or $L_2$ field of an instruction are an unsigned binary integer.

In the SS format with a single, eight-bit length field, for the instructions AND (NC), EXCLUSIVE OR (XC), MOVE (MVC), MOVE NUMERICS, MOVE ZONES, and OR (OC), L specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-256, corresponding to a length code in L of 0-255. Storage results replace the first operand and are never stored outside the field specified by the address and length. In this format, the second operand has the same length as the first operand. There are variations of the preceding definition that apply to EDIT, EDIT AND MARK, PACK ASCII, PACK UNICODE, TRANSLATE, TRANSLATE AND TEST, UNPACK ASCII, and UNPACK UNICODE.

In the SS format with two length fields, and in the RSL format, $L_1$ specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-16, corresponding to a length code in $L_1$ of 0-15. Similarly, $L_2$ specifies the number of additional operand bytes to the right of the location designated by the second-operand address. Results replace the first operand and are never stored outside the field specified by the address and length. If the first operand is longer than the second, the second operand is extended on the left with zeros up to the length of the first operand. This extension does not modify the second operand in storage.

In the SS format with two R fields, as used by the MOVE TO PRIMARY, MOVE TO SECONDARY, and MOVE WITH KEY instructions, the contents of the general register specified by the $R_1$ field are a 32-bit unsigned value called the true length. The operands are both of a length called the effective length. The effective length is equal to the true length or 256, whichever is less. The instructions set the condition code to facilitate programming a loop to move the total number of bytes specified by the true length. The SS format with two R fields is also used to specify one or two registers and one or two storage operands for the PERFORM LOCKED OPERATION instruction.

# Address Generation

## Bimodal Addressing

Bit 32 of the current PSW is the addressing-mode bit. This bit controls the size of the effective address produced by address generation. When bit 32 of the current PSW is zero, the CPU is in the 24-bit addressing mode, and 24-bit instruction and operand effective addresses are generated. When bit 32 of the current PSW is one, the CPU is in the 31-bit addressing mode, and 31-bit instruction and operand effective addresses are generated.

Execution of instructions by the CPU involves generation of the addresses of instructions and operands. This section describes address generation as it applies to most instructions. In some instructions, the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

## Sequential Instruction-Address Generation

When an instruction is fetched from the location designated by the current PSW, the instruction address is increased by the number of bytes in the instruction, and the instruction is executed. The same steps are then repeated by using the new value of the instruction address to fetch the next instruction in the sequence.

In the 24-bit addressing mode, instruction addresses wrap around, with the halfword at instruction address $2^{24} - 2$ being followed by the halfword at instruction address 0. Thus, in the 24-bit addressing mode, any carry out of PSW bit position 40, as a result of updating the instruction address, is lost.

In the 31-bit addressing mode, instruction addresses wrap around, with the halfword at instruction address $2^{31} - 2$ being followed by the halfword at instruction address 0. Thus, in the 31-bit addressing mode, any carry out of PSW bit


LzLabs GmbH. Ex. 1008-171

position 33, as a result of updating the instruction address, is lost.

## Operand-Address Generation

### Formation of the Intermediate Value

An operand address that refers to storage is derived from an intermediate value, which either is contained in a register designated by an R field in the instruction or is calculated from the sum of three binary numbers: base address, index, and displacement.

The base address (B) is a 32-bit number contained in a general register specified by the program in a four-bit field, called the B field, in the instruction. Base addresses can be used as a means of independently addressing each program and data area. In array-type calculations, it can designate the location of an array, and, in record-type processing, it can identify the record. The base address provides for addressing the entire storage. The base address may also be used for indexing.

The index (X) is a 32-bit number contained in a general register designated by the program in a four-bit field, called the X field, in the instruction. It is included only in the address specified by the RX-format instructions. The RX-format instructions permit double indexing; that is, the index can be used to provide the address of an element within an array.

The displacement (D) is a 12-bit number contained in a field, called the D field, in the instruction. The displacement provides for relative addressing of up to 4,095 bytes beyond the location designated by the base address. In array-type calculations, the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

In forming the intermediate sum, the base address and index are treated as 32-bit binary integers. The displacement is similarly treated as a 12-bit unsigned binary integer, and 20 zero bits are appended on the left. The three are added as 32-bit binary numbers, ignoring overflow. The sum is always 32 bits long and is used as an intermediate value to form the generated address. The bits of the intermediate value are numbered 0-31.

A zero in any of the $B_1$, $B_2$, or $X_2$ fields indicates the absence of the corresponding address component. For the absent component, a zero is used in forming the intermediate sum, regardless of the contents of general register 0. A displacement of zero has no special significance.

When an instruction description specifies that the contents of a general register designated by an R field are used to address an operand in storage, the register contents are used as the 32-bit intermediate value.

An instruction can designate the same general register both for address computation and as the location of an operand. Address computation is completed before registers, if any, are changed by the operation.

Unless otherwise indicated in an individual instruction definition, the generated operand address designates the leftmost byte of an operand in storage.

### Formation of the Operand Address

The generated operand address is always 31 bits long, and the bits are numbered 1-31. In some portions of this document, the generated address may be referred to as being 32 bits long, with the bits numbered 0-31. Bit 0 of the generated address is always forced to be zero. The manner in which the generated address is obtained from the intermediate value depends on the current addressing mode. In the 24-bit addressing mode, bits 0-7 of the intermediate value are ignored, bits 0-7 of the generated address are forced to be zeros, and bits 8-31 of the intermediate value become bits 8-31 of the generated address. In the 31-bit addressing mode, bit 0 of the intermediate value is ignored, bit 0 of the generated address is forced to be zero, and bits 1-31 of the intermediate value become bits 1-31 of the generated address.

**Programming Note:** Negative values may be used in index and base-address registers. Bit 0 of these values is always ignored, and, in the 24-bit addressing mode, bits 1-7 of these values are also ignored.



LzLabs GmbH. Ex. 1008-172

## Branch-Address Generation

### Formation of the Intermediate Value

For branch instructions, the address of the next instruction to be executed when the branch is taken is called the branch address. Depending on the branch instruction, the instruction format may be RR, RX, RS, RSI, RI or RIL.

In the RS and RX formats, the branch address is specified by a base address, a displacement, and, for RX, an index. In the RS and RX formats, the generation of the intermediate value follows the same rules as for the generation of the operand-address intermediate value.

In the RR format, the contents of the general register designated by the $R_2$ field are used as the intermediate value from which the branch address is formed. General register 0 cannot be designated as containing a branch address. A value of zero in the $R_2$ field causes the instruction to be executed without branching.

The relative-branch instructions are in the RSI, RI, and RIL formats. In the RSI and RI formats for the relative-branch instructions, the contents of the $I_2$ field are treated as a 16-bit signed binary integer designating a number of halfwords. In the RIL format, the contents of the $I_2$ field are treated as a 32-bit signed binary integer designating a number of halfwords. The branch address is the number of halfwords designated by the $I_2$ field added to the address of the relative-branch instruction.

The 32-bit intermediate value for a relative branch instruction in the RSI, RI, or RIL format is the sum of two addends, with overflow ignored. In the RSI or RI format, the first addend is the contents of the $I_2$ field with one zero bit appended on the right and 15 bits equal to the sign bit of the contents appended on the left. In the RIL format, the first addend is bits 1-31 of the $I_2$ field with one zero bit appended on the right. In all formats, the second addend is the 31-bit address of the branch instruction with one zero bit appended on the left. The address of the branch instruction is the instruction address in the PSW before that address is updated to address the next sequential instruction, or it is the address of the target of the EXECUTE instruction if EXECUTE is used. If EXECUTE is used in the 24-bit addressing mode, the address of the branch instruction is the target address with seven zeros appended on the left.

### Formation of the Branch Address

The branch address is always 31 bits long, with the bits numbered 1-31. The branch address replaces bits 33-63 of the current PSW. The manner in which the branch address is obtained from the intermediate value depends on the addressing mode. For those branch instructions which change the addressing mode, the new addressing mode is used. In the 24-bit addressing mode, bits 0-7 of the intermediate value are ignored, bits 1-7 of the branch address are made zeros, and bits 8-31 of the intermediate value become bits 8-31 of the branch address. In the 31-bit addressing mode, bit 0 of the intermediate value is ignored, and bits 1-31 of the intermediate value become bits 1-31 of the branch address.

For several branch instructions, branching depends on satisfying a specified condition. When the condition is not satisfied, the branch is not taken, normal sequential instruction execution continues, and the branch address is not used. When a branch is taken, bits 1-31 of the branch address replace bits 33-63 of the current PSW. The branch address is not used to access storage as part of the branch operation.

A specification exception due to an odd branch address and access exceptions due to fetching of the instruction at the branch location are not recognized as part of the branch operation but instead are recognized as exceptions associated with the execution of the instruction at the branch location.

A branch instruction, such as BRANCH AND LINK, can designate the same general register for branch-address computation and as the location of an operand. Branch-address computation is completed before the remainder of the operation is performed.

## Instruction Execution and Sequencing

The program-status word (PSW), described in Chapter 4, “Control” contains information required for proper program execution. The PSW is used to control instruction sequencing and to hold and


LzLabs GmbH. Ex. 1008-173

indicate the status of the CPU in relation to the program currently being executed. The active or controlling PSW is called the current PSW.

Branch instructions perform the functions of decision making, loop control, and subroutine linkage. A branch instruction affects instruction sequencing by introducing a new instruction address into the current PSW. The relative-branch instructions with a 16-bit $I_2$ field allow branching to a location at an offset of up to plus 64K - 2 bytes or minus 64K bytes relative to the location of the branch instruction, without the use of a base register. The relative-branch instructions with a 32-bit $I_2$ field allow branching to a location at an offset of up to plus 2G - 2 bytes or minus 2G bytes relative to the location of the branch instruction, without the use of a base register.

## Decision Making

Facilities for decision making are provided by the BRANCH ON CONDITION, BRANCH RELATIVE ON CONDITION, and BRANCH RELATIVE ON CONDITION LONG instructions. These instructions inspect a condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. The condition code, which consists of two bits, provides for four possible condition-code settings: 0, 1, 2, and 3.

The specific meaning of any setting depends on the operation that sets the condition code. For example, the condition code reflects such conditions as zero, nonzero, first operand high, equal, overflow, and subchannel busy. Once set, the condition code remains unchanged until modified by an instruction that causes a different condition code to be set. See Appendix C, “Condition-Code Settings” on page C-1 for a summary of the instructions which set the condition code.

## Loop Control

Loop control can be performed by the use of BRANCH ON CONDITION, BRANCH RELATIVE ON CONDITION, and BRANCH RELATIVE ON CONDITION LONG. to test the outcome of address arithmetic and counting operations. For some particularly frequent combinations of arithmetic and tests, BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL are provided, and relative-branch equivalents of these instructions are also provided. These branches, being specialized, provide increased performance for these tasks.

## Subroutine Linkage without the Linkage Stack

This section describes only the methods for subroutine linkage that do not use the linkage stack. For the linkage extensions provided by the linkage stack, see “Linkage-Stack Introduction” on page 5-60. (Those extensions include a different method of operation of the PROGRAM CALL instruction and also the BRANCH AND STACK, PROGRAM CALL FAST, and PROGRAM RETURN instructions.)

Subroutine linkage is provided by the BRANCH AND LINK, BRANCH AND SAVE, BRANCH RELATIVE AND SAVE, and BRANCH RELATIVE AND SAVE LONG instructions, which permit not only the introduction of a new instruction address but also the preservation of the return address and associated information. Instructions are also provided which set and save the addressing-mode bit, PSW bit 32. These instructions provide the facility for subroutine linkage between programs using the 24-bit and 31-bit addressing modes. Linkage between a problem-state program and the supervisor or monitoring program is provided by means of the SUPERVISOR CALL and MONITOR CALL instructions.

The instructions PROGRAM CALL and PROGRAM TRANSFER provide the facility for linkage between programs of different authority and in different address spaces. PROGRAM CALL permits linkage to a number of preassigned programs that may be in either the problem or the supervisor state and may be in either the same address space or an address space different from that of the caller. It permits a change of the addressing mode, and it permits an increase of PSW-key-mask authority, which authorizes the execution of the SET PSW KEY FROM ADDRESS instruction and also other functions. In general, PROGRAM CALL is used to transfer control to a program of higher authority. PROGRAM TRANSFER permits a change of the instruction address, addressing mode, and address space. PROGRAM TRANSFER also permits a reduction of PSW-key-mask authority and a change from the supervisor to the problem



LzLabs GmbH. Ex. 1008-174

state. In general, it is used to transfer control from one program to another of equal or lower authority.

When a calling linkage is to increase authority, the calling linkage can be performed by PROGRAM CALL and the return linkage by PROGRAM TRANSFER. Alternatively, when the calling linkage is to decrease authority, the calling linkage can be performed by PROGRAM TRANSFER and the return linkage by PROGRAM CALL.

The operation of PROGRAM CALL is controlled by means of an entry-table entry, which is located as part of a table-lookup process during the execution of the instruction. The entry-table entry specifies either a basic (nonstacking) operation or the stacking operation described in "Linkage-Stack Introduction" on page 5-60. The instruction causes the primary address space to be changed only when the ASN in the entry-table entry is nonzero. When the primary address space is changed, the operation is called PROGRAM CALL with space switching (PC-ss). When the primary address space is not changed, the operation is called PROGRAM CALL to current primary (PC-cp).

PROGRAM TRANSFER specifies the new addressing mode and the address space which is to become the new primary address space. When the primary address space is changed, the operation is called PROGRAM TRANSFER with space switching (PT-ss). When the primary address space is not changed, the operation is called PROGRAM TRANSFER to current primary (PT-cp).

The BRANCH AND SET AUTHORITY instruction is available when the branch-and-set-authority facility is installed. BRANCH AND SET AUTHORITY can improve performance by replacing a PT-cp instruction used to perform a calling linkage in which PSW-key-mask authority is reduced, and by replacing a PC-cp instruction used to perform the associated return linkage in which PSW-key-mask authority is restored. BRANCH AND SET AUTHORITY also permits changes between the supervisor and problem states, and it can replace SET PSW KEY FROM ADDRESS by changing the PSW key during the linkage. The calling-linkage operation is called BRANCH AND SET AUTHORITY in the base-authority state (BSA-ba), and the return-linkage operation is called BRANCH AND SET AUTHORITY in the reduced-authority state (BSA-ra).

The BRANCH IN SUBSPACE GROUP instruction is available when the subspace-group facility is installed. The instruction allows linkage within a group of address spaces called a subspace group, where one address space in the group is called the base space and the others are called subspaces. It is intended that each subspace contain a different subset of the storage in the base space, that the base space and each subspace contain a subsystem control program, such as CICS, and application programs, and that each subspace contain the data for a single transaction being processed under the subsystem control program. The placement of the data for each transaction in a different subspace prevents a program that is being executed to process one particular transaction from erroneously damaging the data of other transactions. It is intended that the primary address space be the base space when the control program is being executed, and that it be the subspace for a transaction when an application program is being executed to process that transaction. BRANCH IN SUBSPACE GROUP changes not only the instruction address in the PSW but also the primary segment-table designation in control register 1. BRANCH IN SUBSPACE GROUP does not change the primary ASN in control register 4 or the primary-ASN-second-table-entry origin in control register 5, and, therefore, the base space and the subspaces all are associated with the same ASN, and the programs in those address spaces all are of equal authority.

Although a subspace is intended to be a subset of the base space as described above, the subspace-group facility does not require this, and the facility may be useful in ways other than as described above.

BRANCH IN SUBSPACE GROUP uses an access-list-entry token (ALET) in an access register as an identifier of the address space that is to receive control. The instruction saves the updated instruction address to permit a return linkage, but it does not save an identifier of the address space from which control was transferred. However, an ALET equal to 00000000 hex, called ALET 0, can be used to return from a subspace to the base space, and an ALET equal to 00000001


LzLabs GmbH. Ex. 1008-175

hex, called ALET 1, can be used to return from the base space to the subspace that last had control.

The linkage instructions provided and the functions performed by each are summarized in Figure 5-2 on page 5-13.

The RESUME PROGRAM instruction is available when the resume-program facility is installed. RESUME PROGRAM is intended for use by a problem-state interruption-handling program to return to the interrupted program. The interruption-handling program can use LOAD ACCESS MULTIPLE and LOAD MULTIPLE instructions to restore the contents of the interrupted program's access and general registers from a save area, except for the contents of one access-and-general register pair. The interruption-handling program then can use RESUME PROGRAM to restore the contents of certain PSW fields, including the instruction address, and also the contents of the remaining access-and-general pair from the save area, with that pair first being used by RESUME PROGRAM to address the save area.

The TRAP instruction is available when the trap facility is installed. TRAP (TRAP2, TRAP4) can overlay instructions in an application program and give control to a trap program for performing fix-ups of data used by the application program. The RESUME PROGRAM instruction can be used to return control from the trap program to the application program.



LzLabs GmbH. Ex. 1008-176

| Instruction | Format | Instruction Address PSW Bits 33-63 | | Addressing Mode PSW Bit 32 | | Problem State PSW Bit 15 | | PASN CR4 Bits 16-31 | | PSW-Key Mask Changed in CR3 | Trace |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Save | Set | Save | Set | Save | Set | Save | Set | | |
| BALR* | RR | Yes | $R_2$[1] | AM | - | - | - | - | - | - | $R_2$[1] |
| BAL* | RX | Yes | Yes | AM | - | - | - | - | - | - | - |
| BASR | RR | Yes | $R_2$[1] | Yes | - | - | - | - | - | - | $R_2$[1] |
| BAS | RX | Yes | Yes | Yes | - | - | - | - | - | - | - |
| BASSM | RR | Yes | $R_2$[1] | Yes | $R_2$[1] | - | - | - | - | - | $R_2$[1] |
| BRAS | RI | Yes | Yes | Yes | - | - | - | - | - | - | - |
| BRASL | RI | Yes | Yes | Yes | - | - | - | - | - | - | - |
| BSA-ba | RRE | Yes | Yes | Yes | Yes | Yes | Yes[4] | - | - | "AND" $R_1$[5] | Yes |
| BSA-ra | RRE | $R_1$[1] | Yes | $R_1$[1] | Yes | - | Yes | - | - | Yes | Yes |
| BSG | RRE | Yes | Yes | Yes | Yes | - | - | - | -[3] | - | Yes |
| BSM | RR | - | $R_2$[1] | $R_1$[1] | $R_2$[1] | - | - | - | - | - | - |
| MC#[2] | SI | Yes | Yes | Yes | Yes | Yes | Yes | - | - | - | - |
| PC-cp | S | Yes | Yes | Yes | Yes | Yes | Yes | - | - | "OR" EKM | Yes |
| PC-ss | S | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | "OR" EKM | Yes |
| PT-cp | RRE | - | $R_2$ | - | $R_2$ | - | $R_2$** | - | - | "AND" $R_1$ | Yes |
| PT-ss | RRE | - | $R_2$ | - | $R_2$ | - | $R_2$** | - | Yes | "AND" $R_1$ | Yes |
| RP | S | - | Yes | - | Yes | - | - | - | - | - | Yes |
| SVC[2] | RR | Yes | Yes | Yes | Yes | Yes | Yes | - | - | - | - |
| TRAP2 | E | Yes | Yes | Yes | Yes | Yes | - | - | - | - | Yes |
| TRAP4 | S | Yes | Yes | Yes | Yes | Yes | - | - | - | - | Yes |

**Explanation:**

- \- No
- * In the 24-bit addressing mode, the instruction-length code, condition code, program mask, and 24-bit instruction address are saved, and the 24-bit instruction address is set; in the 31-bit addressing mode, the addressing mode and the 31-bit instruction address are saved, and the 31-bit instruction address is set.
- ** A change from the supervisor to the problem state is allowed; a privileged-operation exception is recognized when a change from the problem to the supervisor state is specified.
- # Monitor-mask bits provide a means of disallowing linkage, or enabling linkage, for selected classes of events.
- [1] The action takes place only if the associated R field in the instruction is nonzero.

*Figure 5-2 (Part 1 of 2). Summary of Linkage Instructions without the Linkage Stack*


LzLabs GmbH. Ex. 1008-177

**Explanation** (Continued):

2 MC and SVC, as part of the interruption, save the entire current PSW and load a new PSW.

3 The primary segment-table designation is set even though the PASN is not set.

4 The problem state is set.

5 The PSW key also is set from general register $R_1$.

AM Saved only if the 31-bit addressing mode is specified.

*Figure 5-2 (Part 2 of 2). Summary of Linkage Instructions without the Linkage Stack*

**Programming Note:** This section describes the linkage instructions that were included in 370-XA and carried forward to ESA/370 and ESA/390. To give the reader a better understanding of the utility and intended usage of these linkage instructions, the following paragraphs in this note describe various program linkages and conventions and the use of the linkage instructions in these situations.

BRANCH RELATIVE AND SAVE and BRANCH RELATIVE AND SAVE LONG, which are not mentioned in the remainder of this section, may be used in place of BRANCH AND SAVE.

The linkage instructions are provided to permit System/370 programs to operate with no modification or only slight modification on ESA/390 systems and also to provide additional function for those programs which are designed to take advantage of the 31-bit addressing of ESA/390. The instructions provide the capability for both old and new programs to coexist in storage and to communicate with each other. It is assumed that old, unmodified programs operate in the 24-bit addressing mode and call, or directly communicate with, other programs operating in the 24-bit addressing mode only. Modified programs normally operate in the 24-bit addressing mode but may call programs which operate in either the 24-bit or 31-bit addressing mode. New programs may be written to operate in either the 24-bit or 31-bit addressing mode, and, in some cases, a program may be written such that it can be invoked in either mode.

SUPERVISOR CALL is provided for compatibility purposes and also because it provides the simplest mechanism to call a program which operates in the supervisor state. It has the advantage over PROGRAM CALL that no general registers are disturbed, that only two bytes in storage are required in line, and that a complete change of PSW status is provided. The return from a routine called by SUPERVISOR CALL normally is accomplished by means of LOAD PSW, which is a privileged instruction.

PROGRAM CALL is provided for fast communication to a program operating in the supervisor state or higher-authority problem state, or even to a program with the same authority. PROGRAM CALL permits a program to call a program operating in a different address space. This would normally be used in the situation where the authorization index associated with the called address space had a higher level of authority than that of the calling address space. The advantage of PROGRAM CALL over SUPERVISOR CALL is in speed, since first-and second-level interruption-handler programs are avoided. It also provides a possible $2^{20}$ different entry points. The authorization key mask in the entry-table entry permits a particular entry point to be available to a limited subset of the programs in the system. Thus, some or all of the authority checking which would otherwise have to be placed in the called program can be eliminated. Return from a routine called by PROGRAM CALL is normally accomplished by means of the PROGRAM TRANSFER instruction; however, LOAD PSW may be used if the called routine is in the supervisor state.

PROGRAM TRANSFER is provided as the return instruction for PROGRAM CALL. It is also useful for calling or transferring to programs with the same authority in another address space. Although PROGRAM TRANSFER does not save the current PASN, the instruction EXTRACT PRIMARY ASN may be used to provide the PASN for return purposes.

BRANCH AND SAVE AND SET MODE (BASSM) is intended to be the principal calling instruction to subroutines outside of an assembler/linkage-editor



LzLabs GmbH. Ex. 1008-178

control section (CSECT), for use by all new programs. BRANCH AND SET MODE (BSM) is intended to be the return instruction used after a BASSM. The calling sequence would normally be:

```
         L      15,ACON
         BASSM  14,15
         ...
         EXTRN  SUB
ACON     DC     A(X'80000000'+SUB)
```

where ACON is an A-type address constant, and the X'80000000' should be present to give control in the 31-bit addressing mode or should be omitted to give control in the 24-bit addressing mode.

The return from such a routine would normally be:

```
         BSM    0,14
```

The BRANCH AND LINK (BAL, BALR) instruction is provided primarily for compatibility reasons. It is defined to operate in the 31-bit addressing mode to increase the probability that an old, straightforward program can be modified to operate in the 31-bit addressing mode with minimal or no change. It is recommended, however, that BRANCH AND SAVE (BAS and BASR) be used instead and that BRANCH AND LINK be avoided since it places nonzero information in the left part of the general register in the 24-bit addressing mode, which may lead to problems. Additionally, BRANCH AND LINK is likely to be slower than BRANCH AND SAVE because BRANCH AND SAVE always saves the right half of the PSW, whereas BRANCH AND LINK must take additional time to check the addressing mode, and then even more time, if in the 24-bit addressing mode, to construct the ILC, condition code, and program mask to be placed in the leftmost byte of the link register.

It is assumed that the normal return from a subroutine called by BRANCH AND LINK (BAL or BALR) will be:

```
     BCR    15,14
```

However, the standard “return instruction”:

```
     BSM    0,14
```

operates correctly for all cases except for a calling BAL executed in the 24-bit addressing mode. In the 24-bit addressing mode, BAL causes an ILC of 10 to be placed in the leftmost two bits of the link register. Thus, a BSM would return in the 31-bit addressing mode. Note that an EXECUTE of BALR in the 24-bit addressing mode also causes the same ILC effect.

The BRANCH AND SAVE (BAS, BASR) instruction is provided to be used for subroutine linkage to any program either within the same CSECT or known to be in the same addressing mode. BASR with the $R_2$ field 0 is also useful for obtaining addressability to the instruction stream by getting a 31-bit address, uncluttered by leftmost fields, in the 24-bit addressing mode. BRANCH AND SAVE (BAS, BASR) is the fastest linkage instruction since the linkage information is not addressing-mode sensitive and since the instruction does not change the addressing mode.

The instruction for returning from a routine called by BRANCH AND SAVE (BAS or BASR) may be either

```
     BCR    15,14
```

or

```
     BSM    0,14
```

In some cases, it may be desirable to rewrite a program that is called by an old program which has not been rewritten. In such a case, the old program, which operates in the 24-bit addressing mode, will be given the address of an intermediate program that will set up the correct entry and return modes and then call the rewritten program. Such an intermediate program is sometimes referred to as a *glue module*. The instruction BRANCH AND SET MODE (BSM) with a nonzero $R_1$ field provides the function necessary to perform this operation efficiently. This is shown in Figure 5-3 on page 5-16.

Note that the “BSM 14,15” in the glue module causes the addressing mode to be saved in bit position 0 of general register 14 and that bits 1-31 of general register 14 are unchanged. Thus, when “BSM 0,14” is executed in the new program, control passes directly back to the old program without passing through the glue module again.


LzLabs GmbH. Ex. 1008-179

```
Old Program              Glue Module                          New Program

        L     15,OLDACON
        BALR  14,15
         •
         •
         •
        EXTRN GLUE
OLDACON DC    A(GLUE)
                         GLUE    CSECT
                                 USING  *,15
                                 L      15,NEWACON
                                 BSM    14,15
                                 EXTRN  NEW
                         NEWACON DC     A(X'80000000'+NEW)
                                                        NEW CSECT
                                                            USING  *,15
                                                             •
                                                             •
                                                             •
                                                            BSM    0,14
```

*Figure 5-3. Glue Module*

## Interruptions

Interruptions permit the CPU to change state as a result of conditions external to the system, in subchannels or input/output (I/O) devices, in other CPUs, or in the CPU itself. Details are to be found in Chapter 6, “Interruptions.”

Six classes of interruption conditions are provided: external, I/O, machine check, program, restart, and supervisor call. Each class has two related PSWs, called old and new, in permanently assigned real storage locations. In all classes, an interruption involves storing information identifying the cause of the interruption, storing the current PSW at the old-PSW location, and fetching the PSW at the new-PSW location, which becomes the current PSW.

The old PSW contains CPU-status information necessary for resumption of the interrupted program. At the conclusion of the program invoked by the interruption, the instruction LOAD PSW may be used to restore the current PSW to the value of the old PSW.

## Types of Instruction Ending

Instruction execution ends in one of five ways: completion, nullification, suppression, termination, and partial completion.

Partial completion of instruction execution occurs only for interruptible instructions; it is described in “Interruptible Instructions” on page 5-17.

### Completion

Completion of instruction execution provides results as called for in the definition of the instruction. When an interruption occurs after the completion of the execution of an instruction, the instruction address in the old PSW designates the next sequential instruction.

### Suppression

Suppression of instruction execution causes the instruction to be executed as if it specified “no operation.” The contents of any result fields, including the condition code, are not changed. The instruction address in the old PSW on an interruption after suppression designates the next sequential instruction.



LzLabs GmbH. Ex. 1008-180

### Nullification

Nullification of instruction execution has the same effect as suppression, except that when an interruption occurs after the execution of an instruction has been nullified, the instruction address in the old PSW designates the instruction whose execution was nullified (or an EXECUTE instruction, as appropriate) instead of the next sequential instruction.

### Termination

Termination of instruction execution causes the contents of any fields due to be changed by the instruction to be unpredictable. The operation may replace all, part, or none of the contents of the designated result fields and may change the condition code if such change is called for by the instruction. Unless the interruption is caused by a machine-check condition, the validity of the instruction address in the PSW, the interruption code, and the ILC are not affected, and the state or the operation of the machine is not affected in any other way. The instruction address in the old PSW on an interruption after termination designates the next sequential instruction.

**Programming Note:** Although the execution of an instruction is treated as a no-operation when suppression or nullification occurs, stores may be performed as the result of the implicit tracing action associated with some instructions. See "Tracing" on page 4-10.

## Interruptible Instructions

### Point of Interruption

For most instructions, the entire execution of an instruction is one operation. An interruption is permitted between operations; that is, an interruption can occur after the performance of one operation and before the start of a subsequent operation.

For the following instructions, referred to as interruptible instructions, an interruption is permitted also after partial completion of the instruction:

- COMPARE AND FORM CODEWORD
- COMPARE LOGICAL LONG
- COMPARE UNTIL SUBSTRING EQUAL
- MOVE LONG
- TEST BLOCK
- UPDATE TREE
- Interruptible instructions of the vector facility (see the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207)

### Unit of Operation

Whenever points of interruption that include those occurring within the execution of an interruptible instruction are discussed, the term "unit of operation" is used. For a noninterruptible instruction, the entire execution consists, in effect, in the execution of one unit of operation.

The execution of an interruptible instruction is considered to consist in the execution of a number of units of operation, and an interruption is permitted between units of operation. The amount of data processed in a unit of operation depends on the particular instruction and may depend on the model and on the particular condition that causes the execution of the instruction to be interrupted.

When an instruction execution consists of a number of units of operation and an interruption occurs after some, but not all, units of operation have been completed, the instruction is said to be partially completed. In this case, the type of ending (completion, inhibition, nullification, or suppression) is associated with the unit of operation. In the case of termination, the entire instruction is terminated, not just the unit of operation.

An exception may exist that causes the first unit of operation of an interruptible instruction not to be completed. In this case when the ending is nullification or suppression, all operand parameters and result locations remain unchanged, except that the condition code is unpredictable if the instruction is defined to set the condition code.

When a storage-alteration PER event is recognized on a model with z/Architecture installed, fewer than 4K additional bytes are stored before the event is indicated by an interruption.

### Execution of Interruptible Instructions

The execution of an interruptible instruction is completed when all units of operation associated with that instruction are completed. When an interruption occurs after completion, inhibition, nullification, or suppression of a unit of operation, all preceding units of operation have been completed, and subsequent units of operation and instructions have not been started. The main difference between these types of ending is the handling of the current unit of operation and whether


LzLabs GmbH. Ex. 1008-181

the instruction address stored in the old PSW identifies the current instruction or the next sequential instruction.

At the time of an interruption, changes to register contents, which are due to be made by an interruptible vector instruction beyond the point of interruption, have not yet been made. Changes to storage locations, however, which are due to be made by an interruptible vector instruction beyond the point of interruption, may have occurred for one or more storage locations beyond the location containing the element identified by the interruption parameters, but not for any location beyond the last element specified by the instruction and not for any locations for which access exceptions exist. Changes to storage locations or register contents which are due to be made by instructions following the interrupted instruction have not yet been made at the time of interruption.

***Completion:*** On completion of the last unit of operation of an interruptible instruction, the instruction address in the old PSW designates the next sequential instruction. The result location for the current unit of operation has been updated. It depends on the particular instruction how the operand parameters are adjusted. On completion of a unit of operation other than the last one, the instruction address in the old PSW designates the interrupted instruction or an EXECUTE instruction, as appropriate. The result location for the current unit of operation has been updated. The operand parameters are adjusted such that the execution of the interrupted instruction is resumed from the point of interruption when the old PSW stored during the interruption is made the current PSW.

***Inhibition:*** When a unit of operation is inhibited, the instruction address in the old PSW designates the interrupted instruction or an EXECUTE instruction, as appropriate. The result location for the current unit of operation is not changed. The operand parameters are adjusted such that, if the instruction is reexecuted, execution of the interrupted instruction is resumed with the next unit of operation. Inhibition occurs only during interruptible vector instructions and is described in more detail in the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207.

***Nullification:*** When a unit of operation is nullified, the instruction address in the old PSW designates the interrupted instruction or an EXECUTE instruction, as appropriate. The result location for the current unit of operation remains unchanged. The operand parameters are adjusted such that, if the instruction is reexecuted, execution of the interrupted instruction is resumed with the current unit of operation.

***Suppression:*** When a unit of operation is suppressed, the instruction address in the old PSW designates the next sequential instruction. The operand parameters, however, are adjusted so as to indicate the extent to which instruction execution has been completed. If the instruction is reexecuted after the conditions causing the suppression have been removed, the execution is resumed with the current unit of operation.

***Termination:*** When an exception which causes termination occurs as part of a unit of operation of an interruptible instruction, the entire operation is terminated, and the contents, in general, of any fields due to be changed by the instruction are unpredictable. On such an interruption, the instruction address in the old PSW designates the next sequential instruction.

The differences among the five types of ending for a unit of operation are summarized in Figure 5-4.

| Unit of Operation Is | Instruction Address | Operand Parameters | Current Result Location |
|---|---|---|---|
| Completed | | | |
| Last unit of operation | Next instruction | Depends on the instruction | Changed |
| Any other unit of operation | Current instruction | Next unit of operation | Changed |
| Inhibited | Current instruction | Next unit of operation | Unchanged |
| Nullified | Current instruction | Current unit of operation | Unchanged |
| Suppressed | Next instruction | Current unit of operation | Unchanged |
| Terminated | Next instruction | Unpredictable | Unpredictable |

*Figure 5-4. Types of Ending for a Unit of Operation*

If an instruction is defined to set the condition code, the execution of the instruction makes the condition code unpredictable except when the last unit of operation has been completed.



LzLabs GmbH. Ex. 1008-182

### Condition-Code Alternative to Interruptibility

The following instructions are not interruptible instructions but instead may be completed after performing a CPU-determined subportion of the processing specified by the parameters of the instructions:

- CHECKSUM
- COMPARE LOGICAL LONG EXTENDED
- COMPARE LOGICAL LONG UNICODE
- COMPARE LOGICAL STRING
- CONVERT UNICODE TO UTF-8
- CONVERT UTF-8 TO UNICODE
- MOVE LONG EXTENDED
- MOVE LONG UNICODE
- MOVE STRING
- SEARCH STRING
- TRANSLATE EXTENDED
- TRANSLATE ONE TO ONE
- TRANSLATE ONE TO TWO
- TRANSLATE TWO TO ONE
- TRANSLATE TWO TO TWO

When any of the above instructions is completed after performing only a CPU-determined amount of processing instead of all specified processing, the instruction sets condition code 3. On such completion, the instruction address in the PSW designates the next sequential instruction, and the operand parameters of the instruction have been adjusted so that the processing of the instruction can be resumed simply by branching back to the instruction to execute it again. When the instruction has performed all specified processing, it sets a condition code other than 3.

The points at which any of the above instructions may set condition code 3 are comparable to the points of interruption of an interruptible instruction, and the amount of processing between adjacent points is comparable to a unit of operation of an interruptible instruction. However, since the instruction is not interruptible, each execution is considered the execution of one unit of operation.

Completion with the setting of condition code 3 permits interruptions to occur. Depending on the model and the instruction, condition code 3 may or may not be set when there is not a need for an interruption.

When a storage-alteration PER event is recognized on a model with z/Architecture installed, fewer than 4K additional bytes are stored before the event is indicated by an interruption.

The COMPARE UNTIL SUBSTRING EQUAL instruction is both an interruptible instruction and one that may set condition code 3 after performing a CPU-determined amount of processing.

**Programming Notes:**

1. Any interruption, other than supervisor call and some program interruptions, can occur after a partial execution of an interruptible instruction. In particular, interruptions for external, I/O, machine-check, restart, and program interruptions for access exceptions and PER events can occur between units of operation.
2. The amount of data processed in a unit of operation of an interruptible instruction depends on the model and may depend on the type of condition which causes the execution of the instruction to be interrupted or stopped. Thus, when an interruption occurs at the end of the current unit of operation, the length of the unit of operation may be different for different types of interruptions. Also, when the stop function is requested during the execution of an interruptible instruction, the CPU enters the stopped state at the completion of the execution of the current unit of operation. Similarly, in the instruction-step mode, only a single unit of operation is performed, but the unit of operation for the various cases of stopping may be different.

## Exceptions to Nullification and Suppression

In certain unusual situations, the result fields of an instruction having a store-type operand are changed in spite of the occurrence of an exception which would normally result in nullification or suppression. These situations are exceptions to the general rule that the operation is treated as a no-operation when an exception requiring nullification or suppression is recognized. Each of these situations may result in the turning on of the change bit associated with the store-type operand, even though the final result in storage may appear unchanged. Depending on the particular situation, additional effects may be observable. The extent


LzLabs GmbH. Ex. 1008-183

of these effects is described along with each of the situations.

All of these situations are limited to the extent that a store access does not occur and the change bit is not set when the store access is prohibited. For the CPU, a store access is prohibited whenever an access exception exists for that access, or whenever an exception exists which is of higher priority than the priority of an access exception for that access.

When, in these situations, an interruption for an exception requiring suppression occurs, the instruction address in the old PSW designates the next sequential instruction. When an interruption for an exception requiring nullification occurs, the instruction address in the old PSW designates the instruction causing the exception even though partial results may have been stored.

## Storage Change and Restoration for DAT-Associated Access Exceptions

In this section, the term "DAT-associated access exceptions" is used to refer to those exceptions which may occur as part of the dynamic-address-translation process. These exceptions are page translation, segment translation, translation specification, and addressing due to a DAT-table entry being designated at a location that is not available in the configuration. The first two of these exceptions normally cause nullification, and the last two normally cause suppression. Protection exceptions, including those due to page protection, are not considered to be DAT-associated access exceptions.

For DAT-associated access exceptions, on some models, channel programs may observe the effects on storage as described in the following case.

When, for an instruction having a store-type operand, a DAT-associated access exception is recognized for any operand of the instruction, that portion, if any, of the store-type operand which would not cause an exception may be changed to an intermediate value but is then restored to the original value.

The accesses associated with storage change and restoration for DAT-associated access exceptions are only observable by channel programs and are not observable by other CPUs in a multiprocessing configuration. Except for instructions which are defined to have multiple-access operands, the intermediate value, if any, is always equal to what would have been the final value if the DAT-associated access exception had not occurred.

**Programming Notes:**

1. Storage change and restoration for DAT-associated access exceptions occur in two main situations:
   a. The exception is recognized for a portion of a store-type operand which crosses a page boundary, and the other portion has no access exception.
   b. The exception is recognized for one operand of an instruction having two storage operands (for example, an SS-format instruction or MOVE LONG), and the other operand, which is a store-type operand, has no access exception.
2. To avoid letting a channel program observe intermediate operand values due to storage change and restoration for DAT-associated access exceptions (especially when a CCW chain is modified), the CPU program should do one of the following:
   a. Operate on one storage page at a time
   b. Perform preliminary testing to ensure that no exceptions occur for any of the required pages
   c. Operate with DAT off

## Modification of DAT-Table Entries

When a valid and attached DAT-table entry is changed to a value which would cause an exception, and when, before the TLB is cleared of entries which qualify for substitution for that entry, an attempt is made to refer to storage by using a virtual address requiring that entry for translation, the contents of any fields due to be changed by the instruction are unpredictable. Results, if any, associated with the virtual address whose DAT-table entry was changed may be placed in those real locations originally associated with the address. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable. On some machines, this situation may be reported by means of an instruction-processing-damage



LzLabs GmbH. Ex. 1008-184

machine check with the delayed-access-exception bit also indicated.

### Trial Execution for Editing Instructions and Translate Instruction

For the instructions EDIT, EDIT AND MARK, and TRANSLATE, the portions of the operands that are actually used in the operation may be established in a trial execution for operand accessibility that is performed before the execution of the instruction is started. This trial execution consists in an execution of the instruction in which results are not stored. If the first operand of TRANSLATE or either operand of EDIT or EDIT AND MARK is changed by another CPU or by a channel program, after the initial trial execution but before completion of execution, the contents of any fields due to be changed by the instruction are unpredictable. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable.

## Authorization Mechanisms

The authorization mechanisms that are described in this section permit the control program to establish the degree of function provided to a particular semiprivileged program. The authorization mechanisms are intended for use by programs considered to be semiprivileged, that is, programs that are executed in the problem state but which may be authorized to use additional capabilities. With these authorization controls, a hierarchy of programs may be established, with programs at a higher level having a greater degree of privilege or authority than programs at a lower level. The range of functions available at each level, and the ability to transfer control from a lower to a higher level, are specified in tables which are managed by the control program. When the linkage stack is used, a nonhierarchical transfer of control also can be specified.

A semiprivileged instruction is one which can be executed in the problem state, but which is subject to the control of one or more of the authorization mechanisms described in this section. There are 26 semiprivileged instructions and also the privileged LOAD ADDRESS SPACE PARAMETERS instruction that are controlled by the authorization mechanisms. All of these semiprivileged and privileged instructions are described in Chapter 10, "Control Instructions."

The instructions controlled by the authorization mechanisms are listed in Figure 5-5 on page 5-25. The figure also shows additional authorization mechanisms that do not control specifically semiprivileged instructions; they control implicit access-register translation (access-register translation as part of an instruction making a storage reference) and also access-register translation in the LOAD REAL ADDRESS, TEST ACCESS, and TEST PROTECTION instructions and a special form of access-register translation in the BRANCH IN SUBSPACE GROUP instruction. These additional mechanisms (the extended authorization index, ALE sequence number, and ASTE sequence number) are described in "Access-Register-Specified Address Spaces" on page 5-34.

### Mode Requirements

Most of the semiprivileged instructions can be executed only with DAT on. Basic PROGRAM CALL, and PROGRAM TRANSFER, are valid only in the primary-space mode. (Basic PROGRAM CALL is the PROGRAM CALL operation when the linkage stack is not used. When the linkage stack is used, the PROGRAM CALL operation is called stacking PROGRAM CALL). MOVE TO PRIMARY and MOVE TO SECONDARY are valid only in the primary-space and secondary-space modes. BRANCH AND STACK, stacking PROGRAM CALL, PROGRAM CALL FAST, and PROGRAM RETURN are valid only in the primary-space and access-register modes. EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE are valid only in the primary-space, access-register, and home-space modes. When a semiprivileged instruction is executed in an invalid translation mode, a special-operation exception is recognized.

PROGRAM TRANSFER specifies a new value for the problem-state bit in the PSW. If a program in the problem state attempts to execute PROGRAM TRANSFER and set the supervisor state, a privileged-operation exception is recognized. A privileged-operation exception is also recognized on an attempt to use RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST to set the home-space mode in the problem state.


LzLabs GmbH. Ex. 1008-185

### Extraction-Authority Control

The extraction-authority-control bit is located in bit position 4 of control register 0. In the problem state, bit 4 must be one to allow completion of these instructions:

- EXTRACT PRIMARY ASN
- EXTRACT SECONDARY ASN
- INSERT ADDRESS SPACE CONTROL
- INSERT PSW KEY
- INSERT VIRTUAL STORAGE KEY

Otherwise, a privileged-operation exception is recognized. The extraction-authority control is not examined in the supervisor state.

### PSW-Key Mask

The PSW-key mask consists of bits 0-15 in control register 3, with the bits corresponding to the values 0-15, respectively, of the PSW key. These bits are used in the problem state to control which keys and entry points are authorized for the program. The PSW-key mask is modified by PROGRAM TRANSFER, is modified or loaded by BRANCH AND SET AUTHORITY and PROGRAM CALL, and is loaded by LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL FAST, and PROGRAM RETURN. The PSW-key mask is used in the problem state to control the following:

- The PSW-key values that can be set by means of the instruction SET PSW KEY FROM ADDRESS.
- The PSW-key values that are valid for the six move instructions that specify a second access key: MOVE PAGE, MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH KEY, MOVE WITH SOURCE KEY, and MOVE WITH DESTINATION KEY.
- The entry points which can be called by means of PROGRAM CALL. In this case, the PSW-key mask is ANDed with the authorization key mask in the entry-table entry, and, if the result is zero, the program is not authorized.

When an instruction in the problem state attempts to use a key not authorized by the PSW-key mask, a privileged-operation exception is recognized. The same action is taken when an instruction in the problem state attempts to call an entry not authorized by the PSW-key mask. The PSW-key mask is not examined in the supervisor state, all keys and entry points being valid.

### Secondary-Space Control

Bit 5 of control register 0 is the secondary-space-control bit. This bit provides a mechanism whereby the control program can indicate whether or not the secondary segment table has been established. Bit 5 may be required to be one to allow completion of SET ADDRESS SPACE CONTROL FAST and must be one to allow completion of these instructions:

- MOVE TO PRIMARY
- MOVE TO SECONDARY
- SET ADDRESS SPACE CONTROL

Otherwise, a special-operation exception is recognized. The secondary-space control is examined in both the problem and supervisor states.

### Subsystem-Linkage Control

When the address-space-function (ASF) control, bit 15 of control register 0, is zero, bit 0 of control register 5 is the subsystem-linkage-control bit. When the ASF control is one, bit 96 of the primary ASN-second-table entry is the subsystem-linkage-control bit. The subsystem-linkage control must be one to allow completion of these instructions:

- PROGRAM CALL
- PROGRAM TRANSFER

Otherwise, a special-operation exception is recognized. The subsystem-linkage control is examined in both the problem and supervisor states and controls both the space-switching and current-primary versions of the instructions.

### ASN-Translation Control

Bit 12 of control register 14 is the ASN-translation-control bit. This bit provides a mechanism whereby the control program can indicate whether ASN translation may occur while a particular program is being executed. Bit 12 must be one to allow completion of these instructions:

- LOAD ADDRESS SPACE PARAMETERS
- SET SECONDARY ASN
- PROGRAM CALL with space switching
- PROGRAM RETURN with space switching and also when the restored secondary ASN is not equal to the restored primary ASN
- PROGRAM TRANSFER with space switching

Otherwise, a special-operation exception is recognized. The ASN-translation control is examined in both the problem and supervisor states. The ASN-translation control is examined by



LzLabs GmbH. Ex. 1008-186

PROGRAM CALL even when PROGRAM CALL obtains the address of the ASN-second-table entry directly from the entry-table entry instead of by performing ASN translation.

### Authorization Index

The authorization index is contained in bit positions 0-15 of control register 4. The authorization index is associated with the primary address space and is loaded along with the PASN when PROGRAM CALL with space switching, PROGRAM CALL FAST with space switching, PROGRAM RETURN with space switching, PROGRAM TRANSFER with space switching, or LOAD ADDRESS SPACE PARAMETERS is executed. The authorization index is used to determine whether a program is authorized to establish a particular address space. A program may be authorized to establish the address space as a secondary-address space, as a primary-address space, or both. The authorization index is examined in both the problem and supervisor states.

Associated with each address space is an authority table. The authorization index is used to select an entry in the authority table. Each entry contains two bits, which indicate whether the program with that authorization index is permitted to establish the address space as a primary address space, as a secondary address space, or both.

The instruction SET SECONDARY ASN with space switching, and the instruction PROGRAM RETURN when the restored secondary ASN is not equal to the restored primary ASN, use the authorization index to test the secondary-authority bit in the authority-table entry to determine if the address space can be established as a secondary address space. The tested bit must be one; otherwise, a secondary-authority exception is recognized.

The instruction PROGRAM TRANSFER with space switching uses the authorization index to test the primary-authority bit in the authority-table entry to determine if the address space can be established as a primary address space. The tested bit must be one; otherwise, a primary-authority exception is recognized.

The instruction PROGRAM CALL with space switching causes a new authorization index to be loaded from the ASN-second-table entry, and PROGRAM CALL FAST causes one to be loaded directly from the PCF-entry-table entry. This permits the program which is called to be given an authorization index which authorizes it to access more or different address spaces than those authorized for the calling program. The instructions PROGRAM RETURN with space switching and PROGRAM TRANSFER with space switching restore the authorization index that is associated with the returned-to address space.

The secondary-authority bit in the authority-table entry may also be used, along with the extended authorization index, to determine if the program is authorized to use an access-list entry in access-register translation. This is described in “Access-Register-Specified Address Spaces” on page 5-34.

### Program-Call-Fast Control

The program-call-fast-control bit is located in bit position 28 of control register 0. Bit 28 must be one to allow execution of PROGRAM CALL FAST. When bit 28 is zero, PROGRAM CALL FAST is treated as a PROGRAM CALL instruction.

### Access-Register and Linkage-Stack Mechanisms

Bit 15 of control register 0 is the address-space-function (ASF) control bit. Bit 15 must be one to allow completion of these instructions:

- BRANCH AND SET AUTHORITY
- BRANCH AND STACK
- BRANCH IN SUBSPACE GROUP
- EXTRACT STACKED REGISTERS
- EXTRACT STACKED STATE
- MODIFY STACKED STATE
- PROGRAM CALL FAST
- PROGRAM RETURN
- TEST ACCESS

Otherwise, a special-operation exception is recognized. The ASF control is examined in both the problem and supervisor states and controls both the space-switching and current-primary forms of PROGRAM RETURN.

Under certain circumstances when the ASF control is or has been zero, erroneous entries may exist in the ART-lookaside buffer (ALB), and this can cause erroneous access-register translation. A description of the circumstances and of how to remove the erroneous entries from the ALB


LzLabs GmbH. Ex. 1008-187

appears in “Formation of ALB Entries” on page 5-54.

The ASF control also controls the setting of the access-register mode by RESUME PROGRAM, SET ADDRESS SPACE CONTROL, and SET ADDRESS SPACE CONTROL FAST, the availability of the stacking PROGRAM CALL operation, control-register contents, the sizes of the entry-table entry and ASN-second-table entry, and other functions. A complete description of the effects of the ASF control is in “Address-Space-Function Control” on page 5-42.

The use of access registers also involves the extended authorization index, ALE sequence number, and ASTE sequence number as authorization mechanisms. These are described in “Access-Register-Specified Address Spaces” on page 5-34.



LzLabs GmbH. Ex. 1008-188

| Function or Instruction | Mode Requirement | | Authorization Mechanism | | | | | | | | | | Space Sw.-Event Ctl. (1.0, 13.0) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pr. Op. | Trans. Mode | Subs. Link. Ctl.[7] | Sec.-Space Ctl. (0.5) | ASN-Trans. Ctl. (14.12) | Extr. Auth. Ctl. (0.4) | PSW-Key Mask (3.0-3.15) | Auth. Index (4.0-4.15) | Ext.-Auth. Index (8.0-8.15) | ALE Seq. No.[8] | ASTE Seq. No.[9] | ASF Ctl. (0.15) | |
| Implic. AR trans. | | A | | | | | | | EA | ALQ | ASQ | EALB | |
| BAKR | | SO-PA | | | | | | | | | | SO | |
| BSA-ba | | | | | | | Q | | | | | SO | |
| BSA-ra | | | | | | | | | | | | SO | |
| BSG | | SO-PSAH | | | | | | | | | ASQ | SO | |
| EPAR | | SO-PSAH | | | | Q | | | | | | | |
| EREG | | SO-PAH | | | | | | | | | | SO | |
| ESAR | | SO-PSAH | | | | Q | | | | | | | |
| ESTA | | SO-PAH | | | | | | | | | | SO | |
| IAC | | SO-PSAH | | | | Q | | | | | | | |
| IPK | | | | | | Q | | | | | | | |
| IVSK | | SO-PSAH | | | | Q | | | | | | | |
| LASP | P | | | | SO | | | CC | | | | Y | CC |
| LRA | P | | | | | | | | CCA | CCA | CCA | | |
| MSTA | | SO-PAH | | | | | | | | | | SO | |
| MVCDK | | | | | | | Q | | | | | | |
| MVCK | | | | | | | Q | | | | | | |
| MVCP | | SO-PS | | SO | | | Q | | | | | | |
| MVCS | | SO-PS | | SO | | | Q | | | | | | |
| MVCSK | | | | | | | Q | | | | | | |
| bPC-cp | | SO-P | SO | | | | Q[1] | | | | | Y | |
| sPC-cp | | SO-PA | SO | | | | Q[1] | | | | | Z | |
| bPC-ss | | SO-P | SO | | SO | | Q[1] | | | | | Y | X1 |
| sPC-ss | | SO-PA | SO | | SO | | Q[1] | | | | | Z | X1 |
| PCF-cp[11] | | SO-PA | | | | | | | | | | SO | |
| PCF-ss[11] | | SO-PA | | | | | | | | | | SO | X1 |
| PR-cp | | SO-PA | | | SO[4] | | | SA[6] | | | | SO | |
| PR-ss | | SO-PA | | | SO | | | PASA[6] | | | | SO | X1 |
| PT-cp | Q[2] | SO-P | SO | | | | | | | | | | |
| PT-ss | Q[2] | SO-P | SO | | SO | | | PA | | | | Y | X1 |
| RP | | | | | | | | | | | | SO[5] | X2 |
| SAC | Q[3] | SO-PSAH | | SO | | | | | | | | SO[5] | X2 |
| SACF | Q[3] | SO-PSAH | | SO[10] | | | | | | | | SO[5] | X2 |
| SPKA | | | | | | | Q | | | | | | |
| SSAR-cp | | SO-PSAH | | | SO | | | | | | | | |
| SSAR-ss | | SO-PSAH | | | SO | | | SA | | | | Y | |

*Figure 5-5 (Part 1 of 2). Summary of Authorization Mechanisms*


LzLabs GmbH. Ex. 1008-189

| Function or Instruction | Mode Requirement | | Authorization Mechanism | | | | | | | | | | Space Sw.-Event Ctl. (1.0, 13.0) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pr. Op. | Trans. Mode | Subs. Link. Ctl.[7] | Sec.-Space Ctl. (0.5) | ASN-Trans. Ctl. (14.12) | Extr. Auth. Ctl. (0.4) | PSW-Key Mask (3.0-3.15) | Auth. Index (4.0-4.15) | Ext.-Auth. Index (8.0-8.15) | ALE Seq. No.[8] | ASTE Seq. No.[9] | ASF Ctl. (0.15) | |
| TAR | | | | | | | | | CC | CC | CC | SO | |
| TPROT | P | | | | | | | | CC | CC | CC | | |

*Figure 5-5 (Part 2 of 2). Summary of Authorization Mechanisms*

***Explanation for Summary of Authorization Mechanisms:***

1 The PSW-key mask is ANDed with the authorization key mask in the entry-table entry.

2 The exception is recognized on an attempt to set the supervisor state when in the problem state.

3 The exception is recognized on an attempt to set the home-space mode when in the problem state.

4 ASN translation is performed for the new SASN, and the exception may be recognized, only when the new SASN is not equal to the new PASN.

5 The exception is recognized on an attempt to set the access-register mode.

6 Secondary authority is checked for the new SASN, and the exception may be recognized, only when the new SASN is not equal to the new PASN.

7 Subsystem-linkage control is bit 0 of control register 5 if the address-space-function (ASF) control, bit 15 of control register 0, is zero; or it is bit 96 of the primary ASN-second-table entry if the ASF control is one.

8 ALE sequence number is bits 8-15 of the access-list-entry token and bits 8-15 of the access-list entry.

9 ASTE sequence number is bits 96-127 of the access-list entry and bits 160-191 of the ASN-second-table entry.

10 Whether the exception is recognized is unpredictable.

11 PROGRAM CALL FAST is treated as PROGRAM CALL if the program-call-fast control, bit 28 of control register 0, is zero.

A Access-register translation occurs only in the access-register mode.

ALQ ALE-sequence exception.

ASQ ASTE-sequence exception.

bPC Basic (nonstacking) PROGRAM CALL.

CC Test results in setting a condition code.

CCA Test results in setting a condition code. The test occurs only in the access-register mode.

CRx.y Control register x, bit position y.

EA Extended-authority exception.

EALB When bit 15 of control register 0 is or has been zero, erroneous ALB entries may exist under certain circumstances. See “Formation of ALB Entries” on page 5-54.

P Privileged-operation exception for privileged instruction.

PA Primary-authority exception.

PASA Primary-authority exception or secondary-authority exception.

Q Privileged-operation exception for semi-privileged instruction. Authority checked only in the problem state.

SA Secondary-authority exception.

SO Special-operation exception.



LzLabs GmbH. Ex. 1008-190

SO-P CPU must be in the primary-space mode; special-operation exception if the CPU is in the secondary-space, access-register, home-space, or real mode.

SO-PA CPU must be in the primary-space or access-register mode; special-operation exception if the CPU is in the secondary-space, home-space, or real mode.

SO-PAH CPU must be in the primary-space, access-register, or home-space mode; special-operation exception if the CPU is in the secondary-space or real mode.

SO-PS CPU must be in the primary-space or secondary-space mode; special-operation exception if the CPU is in the home-space, access-register, or real mode.

SO-PSAH CPU must be in the primary-space, secondary-space, access-register, or home-space mode; special-operation exception if the CPU is in the real mode.

sPC Stacking PROGRAM CALL.

X1 When bit 0 of control register 1 is one, a space-switch event is recognized. The operation is completed.

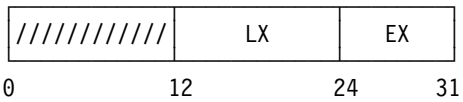
X2 When bit 0 of control register 1 or 13 is one and the instruction space is changed to or from the home address space, a space-switch event is recognized. The operation is completed.

Y The bit is tested to determine the size of the ASTE and/or the ETE.

Z Stacking PROGRAM CALL can occur only when the ASF control is one.

## PC-Number Translation

PC-number translation is the process of translating the 20-bit PC number to locate an entry-table entry as part of the execution of the PROGRAM CALL instruction. To perform this translation, the 20-bit PC number is divided into two fields. Bits 12-23 are the linkage index (LX), and bits 24-31 are the entry index (EX). The effective address, from which the PC number is taken, has the following format:




The translation is performed by means of two tables: a linkage table and an entry table. Both of these tables reside in real storage. The linkage-table designation may reside in control register 5, or it may reside instead in a third area in storage, called the primary ASN-second-table entry (primary ASTE), in which case the origin of the primary ASTE is in control register 5. The entry table is designated by means of a linkage-table entry.

## PC-Number Translation Control

PC-number translation may be controlled by means of a linkage-table designation in control register 5, or it may be controlled by means of controls in control registers 0 and 5 and a linkage-table designation in storage.

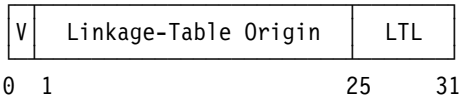
### Control Register 0

Bit 15 of control register 0 is the address-space-function (ASF) control bit. When the ASF control is zero, the linkage-table designation is in control register 5, and the entry-table entry has a length of 16 bytes. When the ASF control is one, control register 5 contains the origin of the primary ASN-second-table entry, the linkage-table designation is in the primary ASTE, and the entry-table entry has a length of 32 bytes.

The ASF control has other effects also. A complete description of the effects of the ASF control is in “Address-Space-Function Control” on page 5-42.

### Control Register 5

When the ASF control in control register 0 is zero, control register 5 contains the linkage-table designation. The register has the following format:




***Subsystem-Linkage Control (V):*** Bit 0 of control register 5 is the subsystem-linkage-control bit. Bit 0 must be one to allow completion of these instructions:

- PROGRAM CALL


LzLabs GmbH. Ex. 1008-191

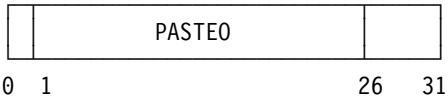
- PROGRAM TRANSFER

Otherwise, a special-operation exception is recognized. The subsystem-linkage control is examined in both the problem and the supervisor states and controls both the space-switching and current-primary versions of the instructions.

***Linkage-Table Origin:*** Bits 1-24 of control register 5, with seven zeros appended on the right, form a 31-bit real address that designates the beginning of the linkage table.

***Linkage-Table Length (LTL):*** Bits 25-31 of control register 5 specify the length of the linkage table in units of 128 bytes, thus making the length of the linkage table variable in multiples of 32 four-byte entries. The length of the linkage table, in units of 128 bytes, is one more than the value in bit positions 25-31. The linkage-table length is compared against the leftmost seven bits of the linkage-index portion of the PC number to determine whether the linkage index designates an entry within the linkage table.

When the ASF control is one, control register 5 specifies the location of the primary ASN-second-table entry. The register has the following format:




***Primary-ASTE Origin (PASTEO):*** Bits 1-25 of control register 5, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the primary ASTE.

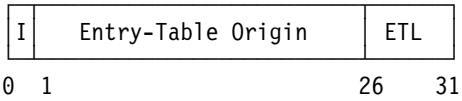
When the ASF control is one, the linkage-table designation is in bytes 12-15 of the primary ASTE. Thus, the subsystem-linkage control (V) is bit 0 of bytes 12-15 of the primary ASTE, the linkage-table origin (LTO) is bits 1-24 of bytes 12-15, and the linkage-table length (LTL) is bits 25-31 of bytes 12-15.

## PC-Number Translation Tables

The PC-number translation process consists in a two-level lookup using two tables: a linkage table and an entry table. These tables reside in real storage.

### Linkage-Table Entries

The entry fetched from the linkage table has the following format:




The fields in the linkage-table entry are allocated as follows:

***LX-Invalid Bit (I):*** Bit 0 controls whether the entry table associated with the linkage-table entry is available.

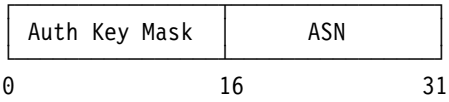
When the bit is zero, PC-number translation proceeds by using the linkage-table entry. When the bit is one, an LX-translation exception is recognized.

***Entry-Table Origin:*** Bits 1-25, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the entry table.

***Entry-Table Length (ETL):*** When the address-space-function (ASF) control, bit 15 of control register 0, is zero, bits 26-31 specify the length of the entry table in units of 64 bytes, thus making the entry table variable in multiples of four 16-byte entries. When the ASF control is one, bits 26-31 specify the entry-table length in units of 128 bytes, thus making the table variable in multiples of four 32-byte entries. The length of the entry table, in units of 64 or 128 bytes, is one more than the value in bit positions 26-31. The entry-table length is compared against the leftmost six bits of the entry index to determine whether the entry index designates an entry within the entry table.

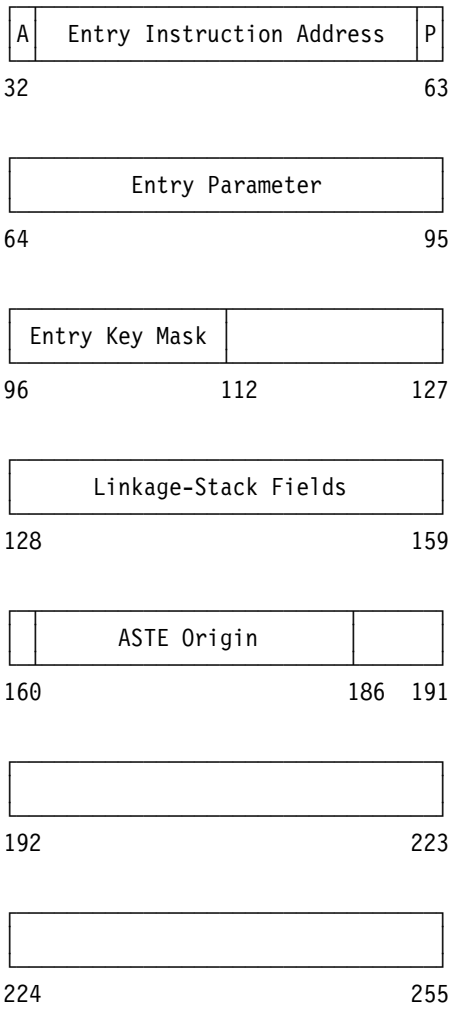
### Entry-Table Entries

When the ASF control in control register 0 is zero, the entry-table entry has a length of 16 bytes. When the ASF control is one, the entry has a length of 32 bytes. The format of the 16-byte entry-table entry is identical to that of the first 16 bytes of the 32-byte entry. The 32-byte entry-table entry has the following format:






LzLabs GmbH. Ex. 1008-192




The fields in the entry-table entry are allocated as follows:

***Authorization Key Mask:*** Bits 0-15 are used to verify whether the program issuing the PROGRAM CALL instruction, when in the problem state, is authorized to call this entry point. The authorization key mask and the current PSW-key mask in control register 3 are ANDed, and the result is checked for all zeros. If the result is all zeros, a privileged-operation exception is recognized. The test is not performed in the supervisor state.

***ASN:*** Bits 16-31 specify whether a space-switching (PC-ss) operation or a to-current-primary (PC-cp) operation is to occur. When bits 16-31 are zero, PC-cp is specified. When bits 16-31 are not all zeros, PC-ss is specified, and the bits are the ASN that replaces the primary ASN.

***Entry Addressing Mode (A):*** Bit 32 replaces the addressing-mode bit, bit 32 of the current PSW, as part of the PROGRAM CALL operation. When bit 32 is zero, bits 33-39 must also be zero; otherwise, a PC-translation-specification exception is recognized.

***Entry Instruction Address:*** Bits 33-62, with a zero appended on the right, form the instruction address which replaces the instruction address in the PSW as part of the PROGRAM CALL operation.

***Entry Problem State (P):*** Bit 63 replaces the problem-state bit, bit 15 of the current PSW, as part of the PROGRAM CALL operation.

***Entry Parameter:*** Bits 64-95 are placed in general register 4 as part of the PROGRAM CALL operation.

***Entry Key Mask:*** Bits 96-111 may be ORed into or may replace the PSW-key mask in control register 3 as part of the PROGRAM CALL operation, as determined by a bit in bit positions 128-159.

***ASTE Origin:*** When the address-space-function (ASF) control is one and bits 16-31 are not all zeros, bits 161-185, with six zeros appended on the right, form the real ASN-second-table-entry address that should result from applying the ASN-translation process to bits 16-31. When the ASF control is one, it is unpredictable whether PC-ss uses bits 161-185 or uses ASN translation to obtain the ASTE address.

Bits 128-159 are used in connection with the linkage stack and are described in “Extended Entry-Table Entries” on page 5-64.

Bits 112-127, 160, and 186-255 are reserved for possible future extensions and should be zeros.

**Programming Note:** The entry parameter is intended to provide the called program with an address which can be depended upon and used as the basis of addressability in locating necessary information which may be environment dependent. The parameter may be appropriately changed for each environment by setting up different entry tables. The alternative — obtaining this information from the calling program — may require extensive validity checking or may present an integrity exposure.


LzLabs GmbH. Ex. 1008-193

## PC-Number-Translation Process

The translation of the PC number is performed by means of a linkage table and entry table both of which reside in real storage. The translation may also require the use of the primary ASN-second-table entry, which also resides in real storage.

For the purposes of PC-number translation, the 20-bit PC number is divided into two parts: the leftmost 12 bits are called the linkage index (LX), and the rightmost eight bits are called the entry index (EX). The LX is used to select an entry from the linkage table, the starting address and length of which are specified by the linkage-table designation in either control register 5 or the primary ASTE. This entry designates the entry table to be used. The EX field of the PC number is then used to select an entry from the entry table.

When, for the purposes of PC-number translation, accesses are made to main storage to fetch entries from the primary ASTE, linkage table, and entry table, key-controlled protection does not apply.

The PC-number-translation process is shown in Figure 5-6 on page 5-31.



LzLabs GmbH. Ex. 1008-194

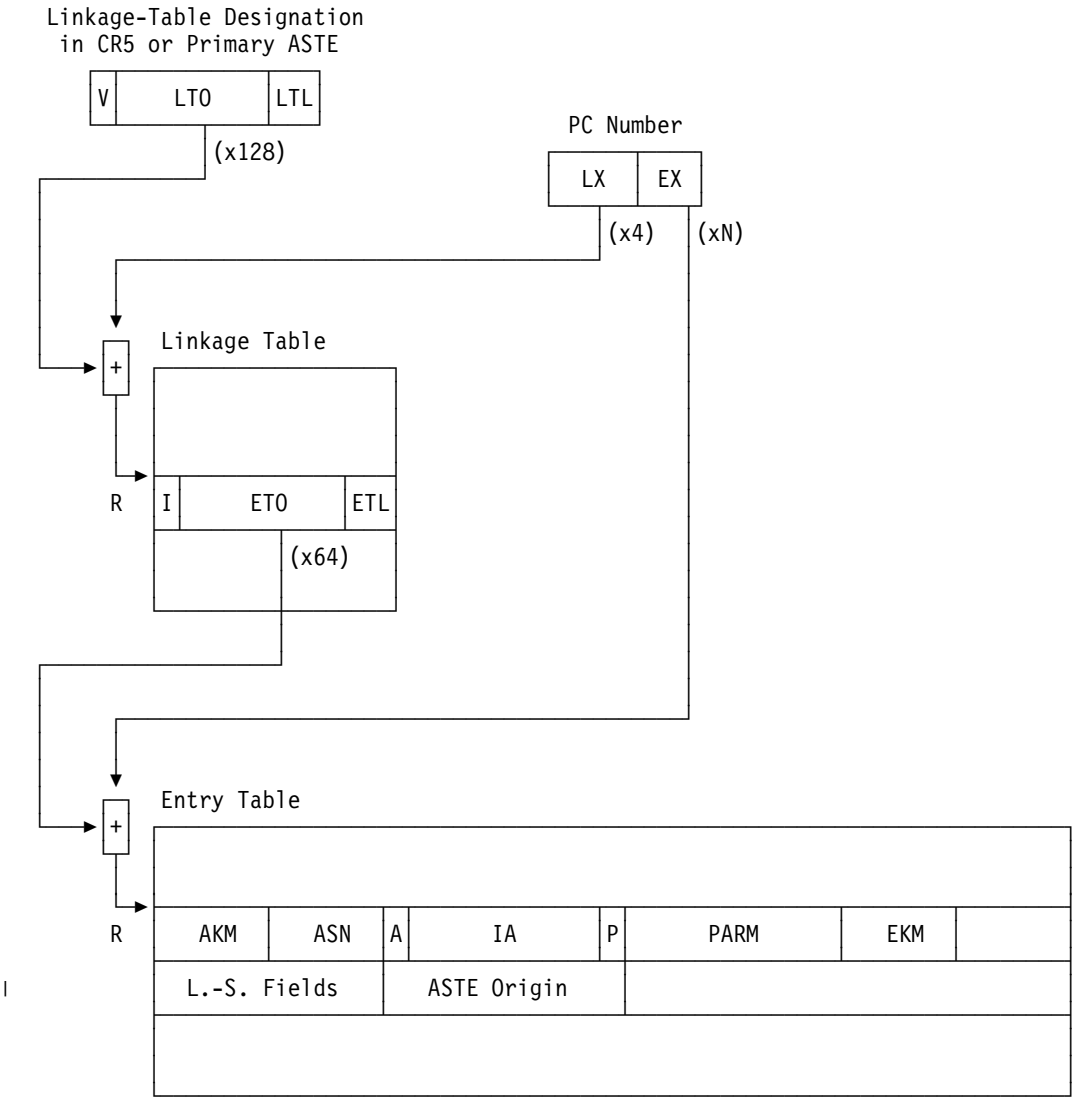



*Figure 5-6. PC-Number Translation*

## Obtaining the Linkage-Table Designation

When the address-space-function (ASF) control, bit 15 of control register 0, is zero, the linkage-table designation is the contents of control register 5. When the ASF control is one, the linkage-table designation is obtained from bytes 12-15 of the primary ASN-second-table entry, the starting address of which is specified by the contents of control register 5.

When the ASF control is one, the 31-bit real address of the linkage-table designation is obtained by appending six zeros on the right to the primary-ASTE origin, bits 1-25 of control register 5, and adding 12. The addition cannot cause a carry into bit position 0. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

When the ASF control is one, all four bytes of the linkage-table designation appear to be fetched concurrently from the primary ASTE as observed by other CPUs. The fetch access is not subject to protection. When the storage address which is generated for fetching the linkage-table designation designates a location which is not available


LzLabs GmbH. Ex. 1008-195

in the configuration, an addressing exception is recognized, and the operation is suppressed. Besides the linkage-table designation, no other field in the primary ASTE is examined.

## Linkage-Table Lookup

The linkage-index (LX) portion of the PC number, in conjunction with the linkage-table origin, is used to select an entry from the linkage table.

The 31-bit real address of the linkage-table entry is obtained by appending seven zeros on the right to the contents of bit positions 1-24 of the linkage-table designation and adding the linkage index, with two rightmost and 17 leftmost zeros appended. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31}$ - 1 to 0. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

As part of the linkage-table-lookup process, the leftmost seven bits of the linkage index are compared against the linkage-table length, bits 25-31 of the linkage-table designation, to establish whether the addressed entry is within the linkage table. If the value in the linkage-table-length field is less than the value of the seven leftmost bits of the linkage index, an LX-translation exception is recognized.

All four bytes of the linkage-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address which is generated for fetching the linkage-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the linkage-table entry specifies whether the entry table corresponding to the linkage index is available. This bit is inspected, and, if it is one, an LX-translation exception is recognized.

When no exceptions are recognized in the process of linkage-table lookup, the entry fetched from the linkage table designates the origin and length of the corresponding entry table.

## Entry-Table Lookup

The entry-index (EX) portion of the PC number, in conjunction with the entry-table origin contained in the linkage-table entry, is used to select an entry from the entry table.

The 31-bit real address of the entry-table entry is obtained by appending six zeros on the right to the entry-table origin and adding: (1) if the ASF control is zero, the entry index, with four rightmost and 19 leftmost zeros appended; or (2) if the ASF control is one, the entry index, with five rightmost and 18 leftmost zeros appended. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31}$ - 1 to 0. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

As part of the entry-table-lookup process, the six leftmost bits of the entry index are compared against the entry-table length, bits 26-31 of the linkage-table entry, to establish whether the addressed entry is within the table. If the value in the entry-table length field is less than the value of the six leftmost bits of the entry index, an EX-translation exception is recognized.

The 16-byte or 32-byte entry-table entry is fetched by using the real address. The fetch of the entry appears to be word concurrent as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining three or seven words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address which is generated for fetching the entry-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

The use that is made of the information fetched from the entry-table entry is described in the definition of the PROGRAM CALL instruction.

## Recognition of Exceptions during PC-Number Translation

The exceptions which can be encountered during the PC-number-translation process and their priority are described in the definition of the PROGRAM CALL instruction.

**Programming Note:** The linkage-table desig-



LzLabs GmbH. Ex. 1008-196

nation is fetched successfully from the primary ASN-second-table entry regardless of the values of bit 0, the ASX-invalid bit, and bits 30, 31, and 60-63 in the primary ASTE. A one value of any of these bits may cause an exception to be recognized in other circumstances.

## Home Address Space

Facilities are provided which a privileged program, such as the control program, can use to obtain control in and access the home address space of a dispatchable unit (for example, a task).

Each dispatchable unit normally has an address space associated with it in which the control program keeps the principal control blocks that represent the dispatchable unit. This address space is called the home address space of the dispatchable unit. Different dispatchable units may have the same or different home address spaces. When the control program initiates a dispatchable unit, it may set the primary and secondary address spaces equal to the home address space of the dispatchable unit. Thereafter, because of the dispatchable unit's possible use of the PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, PROGRAM TRANSFER, or SET SECONDARY ASN instruction, the control program normally cannot depend on either the primary address space or the secondary address space being the home address space when the home address space must be accessed, for example, during the processing by the control program of an interruption. Therefore, the control program normally must take some special action to ensure that the home address space is addressed when it must be accessed. The home-address-space facilities provide an efficient means to take this action.

The home-address-space facilities include:

- The home segment-table designation (HSTD) in control register 13. The HSTD is used by DAT in the same way as the primary segment-table designation (PSTD) in control register 1 and the secondary segment-table designation (SSTD) in control register 7.
- Home-space mode, which results when DAT is on and the address-space control, PSW bits 16 and 17, has the value 11 binary. When the CPU is in the home-space mode, instruction and logical addresses are home virtual addresses and are translated by DAT by means of the HSTD.
- The ability of the RESUME PROGRAM, SET ADDRESS SPACE CONTROL, and SET ADDRESS SPACE CONTROL FAST instructions to set the home-space mode in the supervisor state, and the ability of the INSERT ADDRESS SPACE CONTROL instruction to return an indication of the home-space mode.
- The home space-switch-event control, bit 0 of control register 13.
- Recognition of a space-switch event upon completion of a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction if the CPU was in the home-space mode before or after the operation but not both before and after the operation, if any of the following is true: (1) the primary space-switch-event control, bit 0 of control register 1, is one, (2) the home space-switch-event control is one, or (3) a PER event is to be indicated.

The space-switch event can be used to enable or disable PER or tracing when fetching of instructions begins or ends in particular address spaces.

## Access-Register Introduction

Many of the functions related to access registers are described in this section and in “Subroutine Linkage without the Linkage Stack” on page 5-10, “Access-Register Translation” on page 5-42, and “Sequence of Storage References” on page 5-78. Additionally, translation modes and access-list-controlled protection are described in Chapter 3, “Storage”; the PER-2 means of restricting storage-alteration events to designated address spaces and the handling of access registers during resets and during the store-status operation are described in Chapter 4, “Control”; interruptions are described in Chapter 6, “Interruptions”; instructions are described in Chapter 7, “General Instructions,” and Chapter 10, “Control Instructions”; the handling of access registers during a machine-check interruption and the programmed validation of the access registers are described in Chapter 11, “Machine-Check Handling”; and the alter-and-display controls for


LzLabs GmbH. Ex. 1008-197

access registers are described in Chapter 12, "Operator Facilities."

## Summary

These major functions are provided:

- A maximum of 16 address spaces, including the instruction space, for immediate and simultaneous use by a semiprivileged program; the address spaces are specified by 16 new registers called access registers.
- Instructions for examining and changing the contents of the access registers.

In addition, control and authority mechanisms are incorporated to control these functions.

Access registers allow a sequence of instructions, or even a single instruction such as MOVE (MVC) or MOVE LONG (MVCL), to operate on storage operands in multiple address spaces, without the requirement of changing either the translation mode or other control information. Thus, a program residing in one address space can use the complete instruction set to operate on data in that address space and in up to 15 other address spaces, and it can move data between any and all pairs of these address spaces. Furthermore, the program can change the contents of the access registers in order to access still other address spaces.

The instructions for examining and changing access-register contents are unprivileged and are described in Chapter 7, "General Instructions." They are:

- COPY ACCESS
- EXTRACT ACCESS
- LOAD ACCESS MULTIPLE
- LOAD ADDRESS EXTENDED
- SET ACCESS
- STORE ACCESS MULTIPLE

The privileged PURGE ALB instruction and COMPARE AND SWAP AND PURGE instruction are used in connection with access registers and are described in Chapter 10, "Control Instructions."

Access registers specify address spaces when the CPU is in the access-register mode. The SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST instructions allow setting of the access-register mode, and the INSERT ADDRESS SPACE CONTROL instruction provides an indication of the access-register mode. The stacking PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, and RESUME PROGRAM instructions also allow setting of the access-register mode. All of these instructions are described in Chapter 10, "Control Instructions."

Access registers are used in a special way by the BRANCH IN SUBSPACE GROUP instruction. The use of access registers by that instruction is described in detail only in the definition of the instruction in Chapter 10, "Control Instructions." However, "Subspace-Group Tables" on page 5-56 describes the use of the dispatchable-unit control table and the extended ASN-second-table entry by BRANCH IN SUBSPACE GROUP.

## Access-Register Functions

### Access-Register-Specified Address Spaces

The CPU includes sixteen 32-bit access registers numbered 0-15. In the access-register mode, which results when DAT is on and PSW bits 16 and 17 are 01 binary, an instruction B or R field that is used to specify the logical address of a storage operand designates not only a general register but also an access register. The designated general register is used in the ordinary way to form the logical address of the storage operand. The designated access register is used to specify the address space to which the logical address is relative. The access register specifies the address space by specifying a segment-table designation for the address space, and this segment-table designation is used by DAT to translate the logical address. An access register specifies a segment-table designation in an indirect way, not by containing the segment-table designation.

An access register may specify the primary or secondary segment-table designation in control register 1 or 7, respectively, or it may specify a segment-table designation contained in an ASN-second-table entry. In the latter case, the access register designates an entry in a table called an access list, and the designated access-list entry in turn designates the ASN-second-table entry.



LzLabs GmbH. Ex. 1008-198

The process of using the contents of an access register to obtain a segment-table designation for use by DAT is called access-register translation (ART). This is depicted in Figure 5-7 on page 5-35.

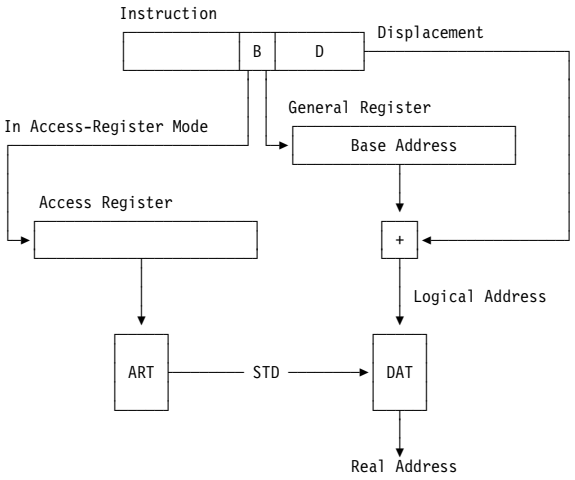



*Figure 5-7. Use of Access Registers*

An access register is said to specify an AR-specified address space by means of an AR-specified segment-table designation. The virtual addresses in an AR-specified address space are called AR-specified virtual addresses.

In the access-register mode, whereas all storage-operand addresses are AR-specified virtual, instruction addresses are primary virtual.

***Designating Access Registers:*** In the access-register mode, an instruction B or R field designates an access register, for use in access-register translation, under the following conditions:

- The field is a B field which designates a general register containing a base address. The base address is used, along with a displacement (D) and possibly an index (X), to form the logical address of a storage operand.
- The field is an R field which designates a general register containing the logical address of a storage operand.

For example, consider the following instruction:

```
MVC 0(L,1),0(2)
```

The second operand, of length L, is to be moved to the first-operand location. The logical address of the second operand is in general register 2, and that of the first-operand location in general register 1. The address space containing the second operand is specified by access register 2, and that containing the first-operand location by access register 1. These two address spaces may be different address spaces, and each may be different from the current instruction address space (the primary address space).

When PSW bits 16 and 17 are 01, the B field of the LOAD REAL ADDRESS instruction designates an access register, for use in access-register translation, regardless of whether DAT is on or off.

The COMPARE AND FORM CODEWORD and UPDATE TREE instructions specify storage operands by means of implicitly designated general registers and access registers.

The MOVE TO PRIMARY and MOVE TO SECONDARY instructions specify storage operands by means of primary virtual and secondary virtual addresses, and access registers do not apply to these instructions. An exception is recognized when either of these instructions is executed in the access-register mode. The MOVE WITH KEY instruction can be used in place of MOVE TO PRIMARY and MOVE TO SECONDARY in the access-register mode. The MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY instructions also can be used.

An instruction R field may designate an access register for other than the purpose of access-register translation.

The fields which may designate access registers, whether or not for access-register translation, are indicated in the summary figure at the beginning of each instruction chapter.

***Obtaining the Segment Table Designation:*** This section and the following ones introduce the access-register-translation process and present the concepts related to access lists.

The segment-table designation specified by an access register is obtained by access-register translation as follows:

- If the access register contains 00000000 hex, the specified segment-table designation is the primary segment-table designation (PSTD), obtained from control register 1.
- If the access register contains 00000001 hex, the specified segment-table designation is the


LzLabs GmbH. Ex. 1008-199

secondary segment-table designation (SSTD), obtained from control register 7.

- If the access register contains any other value, the specified segment-table designation is obtained from an ASN-second-table entry. The contents of the access register designate an access-list entry that contains the real origin of the ASN-second-table entry.

Access register 0 is treated in a special way by access-register translation; it is treated as containing 00000000 hex, and its actual contents are not examined. Thus, a logical address specified by means of a zero B or R field in the access-register mode is always relative to the primary address space, regardless of the contents of access register 0. However, there is one exception to how access register 0 is treated: the TEST ACCESS instruction uses the actual contents of access register 0, instead of treating access register 0 as containing 00000000 hex.

The treatment of an access register containing the value 00000000 hex as designating the current primary address space allows that address space to be addressed, in the access-register mode, without requiring the use of an access-list entry. This is useful when the primary address space is changed by a space-switching PROGRAM CALL (PC-ss), PROGRAM CALL FAST (PCF-ss), PROGRAM RETURN (PR-ss), or PROGRAM TRANSFER (PT-ss) instruction. Similarly, the treatment of an access register containing the value 00000001 hex as designating the secondary address space allows that space to be addressed after a space-switching operation, again without requiring the use of an access-list entry.

The contents of the access registers are not changed by the PROGRAM CALL, PROGRAM CALL FAST, and PROGRAM TRANSFER instructions. Therefore, an access register containing 00000000 or 00000001 hex may specify a different address space after the execution of PROGRAM CALL, PROGRAM CALL FAST, or PROGRAM TRANSFER than before the execution. For example, if a space-switching PROGRAM CALL instruction is executed, an access register containing 00000000 hex specifies the old primary address space before the execution and the new primary address space after the execution.

When access-register translation obtains a segment-table designation from an ASN-second-table entry, bit 0 of the entry, the ASX-invalid bit, must be zero; otherwise, an exception is recognized.

***Access Lists:*** The access-list entry that is designated by the contents of an access register can be located in either one of two access lists, the dispatchable-unit access list or the primary-space access list. A bit in the access register specifies which of the two access lists contains the designated entry. Both of the access lists reside in real or absolute storage. The locations of the access lists are specified by means of control registers 2 and 5.

Control register 2 contains the origin of a real-storage area called the dispatchable-unit control table. The dispatchable-unit control table contains the designation — the real or absolute origin, and length — of the dispatchable-unit access list.

When the address-space-function (ASF) control, bit 15 of control register 0, is one, control register 5 contains the origin of a real-storage area called the primary ASN-second-table entry. The primary ASN-second-table entry contains the designation of the primary-space access list, and it also contains the linkage-table designation. When the ASF control is zero, the linkage-table designation is in control register 5.

The ASF control determines the contents of control register 5 for the instructions LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, PROGRAM RETURN, and PROGRAM TRANSFER. The access-register-translation process always treats control register 5 as containing the primary-ASN-second-table-entry origin and does not examine the ASF control.

An access list, either the dispatchable-unit access list or the primary-space access list, contains one of the following, depending on the model: (1) some multiple of eight 16-byte entries, up to a maximum of 1,024 entries, or (2) some multiple of sixteen 16-byte entries, up to a maximum of 4,096 entries.

***Programs and Dispatchable Units:*** When discussing access lists, it is necessary to distinguish between the terms “program” and “dispatchable unit.” A program is a sequence of instructions and



LzLabs GmbH. Ex. 1008-200

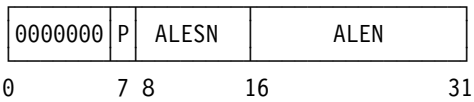
may be referred to as a program module. A program may be a sequence of calling and called programs. A dispatchable unit, which is sometimes called a process or a task, is a unit of work that is performed through the execution of a program by one CPU at a time.

The dispatchable-unit access list is intended to be associated with a dispatchable unit; that is, it is intended that a dispatchable unit have the same dispatchable-unit access list regardless of which program is currently being executed to perform the dispatchable unit. There is no mechanism, except for the LOAD CONTROL instruction, that changes the dispatchable-unit-control-table origin in control register 2.

The primary-space access list is associated with the primary address space that is specified by the primary ASN in control register 4 and the primary segment-table designation in control register 1. The primary-space access list that is available for use by a dispatchable unit changes as the primary address space of the dispatchable unit changes, that is, whenever a program in a different primary address space begins to be executed to perform the dispatchable unit. Whenever a LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, or PROGRAM TRANSFER instruction replaces the primary ASN in control register 4 and the primary segment-table designation in control register 1, it also replaces the primary-ASN-second-table-entry origin in control register 5, if the address-space-function control is one.

Thus, for a dispatchable unit, the dispatchable-unit access list is intended to be constant (although its entries may be changed, as will be described), and the primary-space access list is a function of which program is being executed, through being a function of the primary address space of the program. Also, all dispatchable units and programs in the same primary address space have the same primary-space access list.

***Access-List-Entry Token:*** The contents of an access register are called an access-list-entry token (ALET) since, in the general case, they designate an entry in an access list. An ALET has the following format:




The ALET contains a primary-list bit (P) that specifies which access list contains the designated access-list entry: the dispatchable-unit access list if the bit is zero, or the primary-space access list if the bit is one. The specified access list is called the effective access list.

The ALET also contains an access-list-entry number (ALEN) which, when multiplied by 16, is the number of bytes from the beginning of the effective access list to the designated access-list entry. During access-register translation, an exception is recognized if the ALEN designates an entry that is outside the effective access list or if the leftmost seven bits in the ALET are not all zeros.

The access-list-entry sequence number (ALESN) in the ALET is described in the next section.

The above format of the ALET does not apply when the ALET is 00000000 or 00000001 hex.

An ALET can exist in an access register, in a general register, or in storage, and it has no special protection from manipulation by the problem program. Any program can transfer ALETs back and forth among access registers, general registers, and storage. A called program can save the contents of the access registers in any storage area available to it, load and use the access registers for its own purposes, and then restore the original contents of the access registers before returning to its caller.

***Allocating and Invalidating Access-List Entries:*** It is intended that access lists be provided by the control program and that they be protected from direct manipulation by any problem program. This protection may be obtained by means of key-controlled protection or by placing the access lists in real storage not accessible by any problem program by means of DAT.

As determined by a bit in the entry, an access-list entry is either valid or invalid. A valid access-list entry specifies an address space and can be used by a suitably authorized program to access that space. An invalid access-list entry is available for allocation as a valid entry. It is intended that the


LzLabs GmbH. Ex. 1008-201

control program provide services that allocate valid access-list entries and that invalidate previously allocated entries.

Allocation of an access-list entry may consist in the following steps. A problem program passes some kind of identification of an address space to the control program, and it passes a specification of either the dispatchable-unit access list or the primary-space access list. The control program checks, by some means, the authority of the problem program to access the address space. If the problem program is authorized, the control program selects an invalid entry in the specified access list, changes it to a valid entry specifying the subject address space, and returns to the problem program an access-list-entry token (ALET) that designates the allocated entry. The problem program can subsequently place the ALET in an access register in order to access the address space. Later, through the use of the invalidation service of the control program, the access-list entry that was allocated may be made invalid. An exception is recognized during access-register translation if an ALET is used that designates an invalid access-list entry.

It may be that a particular access-list entry is allocated, then invalidated, and then allocated again, this time specifying a different address space than the first time. To guard against erroneous use of an ALET that designates a conceptually wrong address space, an access-list-entry sequence number (ALESN) is provided in both the ALET and the access-list entry. When the control program allocates an access-list entry, it should place the same ALESN in the entry and in the designating ALET that it returns to the problem program. When the control program reallocates an access-list entry, it should change the value of the ALESN. An exception is recognized during access-register translation if the ALESN in the ALET used is not equal to the ALESN in the designated access-list entry.

The ALESN check is a reliability mechanism, not an authority mechanism, because the ALET is not protected from the problem program, and the problem program can change the ALESN in the ALET to any value. Also, this is not a fail-proof reliability mechanism because the ALESN is one byte and its value wraps around after 256 reallocations, assuming that the value is incremented by one for each reallocation.

***Authorizing the Use of Access-List Entries:*** Although an access list is intended to be associated with either a dispatchable unit or a primary address space, the valid entries in the list are intended to be associated with the different programs that are executed, in some order, to perform the work of the dispatchable unit. It is intended that each program be able to have a particular authority that permits the use of only those access-list entries that are associated with the program. The authority being referred to here is represented by a 16-bit extended authorization index (EAX) in control register 8. Other elements used in the related authorization mechanism are: (1) a private bit in the access-list entry, (2) an access-list-entry authorization index (ALEAX) in the access-list entry, and (3) the authority table.

A program is authorized to use an access-list entry, in access-register translation, if any of the following conditions is met:

1. The private bit in the access-list entry is zero. This condition provides a high-performance means to authorize any and all programs that are executed to perform the dispatchable unit.
2. The ALEAX in the access-list entry is equal to the EAX in control register 8. This condition provides a high-performance means to authorize only particular programs.
3. The EAX selects a secondary bit that is one in the authority table associated with the address space that is specified by the access-list entry. The authority table is locatable in that the access-list entry contains the real origin of the ASN-second-table entry (ASTE) for the address space, and the ASTE contains the real origin of the authority table. This condition provides another means, less well-performing than condition 2, for authorizing only particular programs. However, providing for condition 3 to be met instead of condition 2 can be advantageous because it permits several programs, each executed with a different EAX, all to use a single access-list entry to access a particular address space.

Access-register translation tests for the three conditions in the order indicated by their numbers, and a higher-numbered condition is not tested for if a lower-numbered condition is met. An exception is recognized if none of the conditions is met.



LzLabs GmbH. Ex. 1008-202

Figure 5-8 on page 5-39 shows an example of how the authorization mechanism can be used. In the figure, “PBZ” means that the private bit is zero, and “PBO” means that the private bit is one.

The figure shows an access list — assume it is a dispatchable-unit access list — in which the entries of interest are entries 4, 7, 9, and 12. Each access-list entry contains a private bit, an ALEAX, and the real origin of the ASTE for an address space. The private bit in entry 4 is zero, and, therefore, the value of the ALEAX in entry 4 is immaterial and is not shown. The private bits in entries 7, 9, and 12 are ones, and the ALEAX values in these entries are as shown. The numbers used to identify the address spaces (36, 25, 62, and 17) are arbitrary. They may be the ASNs of the address spaces; however, ASNs are in no way used in access-register translation. Only the authority table for address space 17 is shown. In it, the secondary bit selected by EAX 10 is one. Assume that no secondary bits are ones in the authority tables for the other spaces.

The figure also shows a sequence of three programs, named A, B, and C, that is executed to perform the work of the dispatchable unit associated with the access list. These programs may be in the same or different address spaces. The EAX in control register 8 when each of these programs is executed is 0, 5, and 10, respectively.

Each of programs A, B, and C can use access-list entry (ALE) 4 to access address space 36 since the private bit in ALE 4 is zero. Program B can use ALE 7 to access space 25 because the ALEAX in the ALE equals the EAX for the program, and no other program can use this ALE. Similarly, only program C can use ALE 9. Program B can use ALE 12 because the ALEAX and EAX are equal, and program C can use it because C's EAX selects a secondary bit that is one in the authority table for space 17.

The example would be the same if programs A, B, and C were all in the same address space and the access list were the primary-space access list for that space.

An ALE in which the private bit is zero may be called public because the ALE can be used by any program, regardless of the value of the current EAX. An ALE in which the private bit is one may be called private because the ability of a program to use the ALE depends on the current EAX.

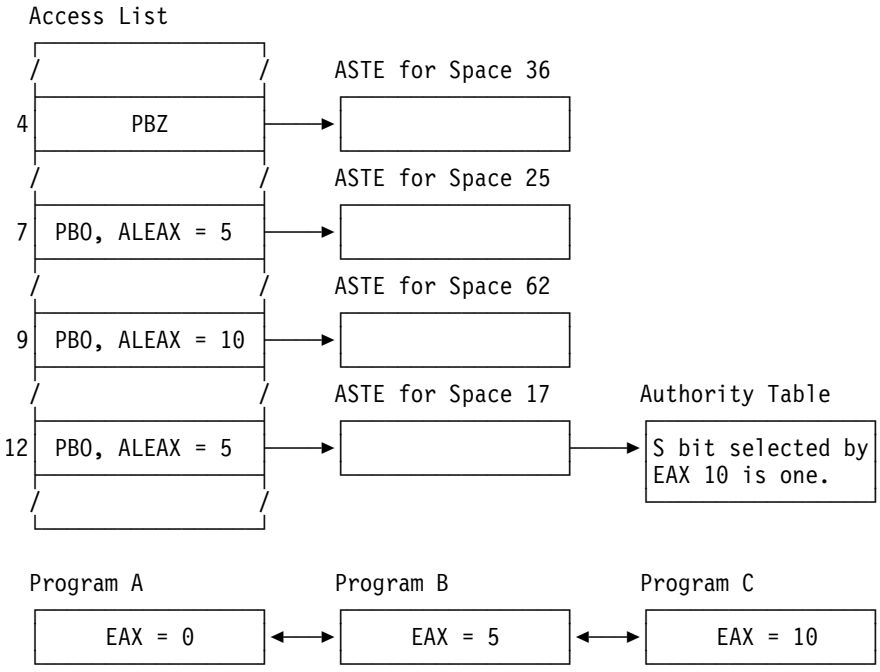



*Figure 5-8. Example of Authorizing the Use of Access-List Entries*


LzLabs GmbH. Ex. 1008-203

***Notes on the Authorization Mechanism:*** An access list is a kind of capability list, in the sense in which the word "capability" is used in computer science. It is up to the control program to formulate the policies that are used to allocate entries in an access list, and the programmed authorization checking required during allocation may be very complex and lengthy. After a valid entry has been made in an access list, the access-register-translation process enforces the control-program policies in a well-performing way by means of the authorization mechanism described above.

Using access lists has an advantage over using only ASNs and authority tables. For example, assume that an access register could contain an ASN and that access-register translation would do ASN translation of the ASN and then use the EAX to test the authority table. This would make the EAX relevant to all existing address spaces, and, therefore, it would make the management of EAXs and their assignment to programs more difficult. With the actual definitions of the ALET and access-register translation, an EAX is relevant to only the address spaces that are represented in the current dispatchable-unit and primary-space access lists. Also, since ASN translation is not done as a part of access-register translation, the number of concurrently existing address spaces, as represented by ASN-second-table entries, can be greater than the number of available ASNs (64K).

The extended entry-table entry and linkage stack can be used to assign EAXs to programs and to change the EAX in control register 8 during program linkages. These components are introduced in "Linkage-Stack Introduction" on page 5-60.

The SET SECONDARY ASN instruction and the authorization index (AX), bits 0-15 of control register 4, can play a role in the use of access registers. The space-switching form of SET SECONDARY ASN (SSAR-ss) establishes a new secondary address space if the secondary bit selected by the AX is one in the authority table associated with the new secondary space. The secondary space can be addressed by means of an ALET having the value 00000001 hex.

***Revoking Accessing Capability:*** Another mechanism, which is a combined authority and integrity mechanism, is part of access-register translation, and it is described in this section.

An access-list entry (ALE) contains an ASN-second-table-entry sequence number (ASTESN), and so does the ASTE designated by the ALE when the ASTE is extended to 64 bytes, as it is when the address-space-function control is one. During access-register translation, the ASTESN in the ALE must equal the ASTESN in the designated ASTE; otherwise, an exception is recognized.

When the control program allocates an ALE, it should copy the ASTESN from the designated ASTE into the ALE. Subsequently, the control program can, in effect, revoke the addressing capability represented by the ALE by changing the ASTESN in the ASTE. Changing the ASTESN in the ASTE makes all previously usable ALEs that designate the ASTE unusable.

Making an ALE unusable may be required in either of two cases:

1. Some element of the control-program policy for determining the authority of a program to have access to the address space specified by the ASTE has changed. This may mean that some or all of the programs that were authorized to the address space, and for which ALEs have been allocated, are no longer authorized.

   Changing the ASTESN in the ASTE ends the usability of all ALEs that designate the ASTE. If this revocation of capability is to be selective, then, when an exception is recognized because of unequal ASTESNs, the control program can reapply its programmed procedures for determining authorization, and an ALE which should have remained usable can be made usable again by copying the new ASTESN into it. When the usability of an ALE is restored, the control program normally should cause reexecution of the instruction that encountered the exception.

2. The ASTE has been reassigned to specify a conceptually different address space, and ALEs which specified the old address space must not be allowed to specify the new one. (Bit 0 of the ASTE, the ASX-invalid bit, can be



LzLabs GmbH. Ex. 1008-204

set to one to delete the assignment of the ASTE to an address space, and this prevents the use of the ASTE in access-register translation. But after reassignment, bit 0 normally again is zero.)

The ASTESN mechanism may be regarded as an authority mechanism in the first case above and as an integrity mechanism in the second.

The ASTESN mechanism is especially valuable because it avoids the need of the control program to keep track of the access lists that contain the ALEs that designate each ASTE. Furthermore, it avoids the need of searching through these access lists in order to find the ALEs and set them invalid, to prevent the use of the ALEs in access-register translation. The latter activity could be particularly time-consuming, or could present a particularly difficult management problem, because the access lists could be in auxiliary storage, such as a direct-access storage device, when the need arises to invalidate the ALEs.

The ASTESN is a four-byte field. Assuming a reasonable frequency of authorization-policy changes or address-space reassignments, the approximately four billion possible values of the ASTESN provide a fail-proof authority or integrity mechanism over the lifetime of the system.

***Preventing Store References:*** The access-list entry contains a fetch-only bit which, when one, specifies that the access-list entry cannot be used to perform storage-operand store references. The principal description of the effect of the fetch-only bit is in "Access-List-Controlled Protection" on page 3-11.

***Improving Translation Performance:*** Access-register translation (ART) conceptually occurs each time a logical address is used to reference a storage operand in the access-register mode. To improve performance, ART normally is implemented such that some or all of the information contained in the ART tables (access-list-designation sources, access lists, ASN second tables, and authority tables) is maintained in a special buffer referred to as the ART-lookaside buffer (ALB). The CPU necessarily refers to an ART-table entry in real storage only for the initial access to that entry. The information in the entry may be placed in the ALB, and subsequent translations may be performed using the information in the ALB.

The PURGE ALB instruction and the COMPARE AND SWAP AND PURGE instruction can be used to clear all information from the ALB after a change has been made to an ART-table entry in real storage.

## Access-Register Instructions

The following instructions are provided for examining and changing the contents of access registers:

- COPY ACCESS
- EXTRACT ACCESS
- LOAD ACCESS MULTIPLE
- LOAD ADDRESS EXTENDED
- SET ACCESS
- STORE ACCESS MULTIPLE

The SET ACCESS instruction replaces the contents of a specified access register with the contents of a specified general register. Conversely, the EXTRACT ACCESS instruction moves the contents of an access register to a general register. The COPY ACCESS instruction moves the contents of one access register to another.

The LOAD ACCESS MULTIPLE instruction loads a specified set of consecutively numbered access registers from a specified storage location whose length in words equals the number of access registers loaded. Conversely, the STORE ACCESS MULTIPLE instruction function stores the contents of a set of access registers at a storage location.

The LOAD ADDRESS EXTENDED instruction is similar to the LOAD ADDRESS instruction in that it loads a specified general register with an effective address specified by means of the B, X, and D fields of the instruction. In addition, LOAD ADDRESS EXTENDED operates on the access register having the same number as the general register loaded. When the address-space control, PSW bits 16 and 17, is 00, 10, or 11 binary, LOAD ADDRESS EXTENDED loads the access register with 00000000, 00000001, or 00000002 hex, respectively. When the address space control is 01 binary, LOAD ADDRESS EXTENDED loads the target access register with a value that depends on the B field of the instruction. If the B field is zero, LOAD ADDRESS EXTENDED loads the target access register with 00000000 hex. If the B field is nonzero, LOAD ADDRESS EXTENDED loads the target access register with the contents of the access register


LzLabs GmbH. Ex. 1008-205

designated by the B field. However, in the last case when bits 0-6 of the access register designated by the B field are not all zeros, the results in the target general register and access register are unpredictable.

The address-space-control values 00, 01, 10, and 11 binary specify primary-space, access-register, secondary-space, and home-space mode, respectively, when DAT is on. LOAD ADDRESS EXTENDED functions the same regardless of whether DAT is on or off.

When used in access-register translation, the access-register values 00000000 and 00000001 hex specify the primary and secondary address spaces, respectively, and the value 00000002 hex designates entry 2 in the dispatchable-unit access list. Loading the target access register with 00000002 hex when the address-space control is 11 binary is intended to support assignment, by the control program, of entry 2 in the dispatchable-unit access list as specifying the home address space.

## Access-Register Translation

Access-register translation is introduced in "Access-Register-Specified Address Spaces" on page 5-34.

## Access-Register-Translation Control

Access-register translation is controlled by an address-space control, by the address-space-function (ASF) control in control register 0, and by controls in control registers 2, 5, and 8. The address-space control, PSW bits 16 and 17, is described in "Translation Modes" on page 3-28. The other controls are described below.

Additional controls are located in the access-register-translation tables.

### Address-Space-Function Control

Bit 15 of control register 0 is the address-space-function (ASF) control. This bit must be one when a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that is to set the access-register mode is executed, and when a BRANCH AND SET AUTHORITY, BRANCH AND STACK, BRANCH IN SUBSPACE GROUP, EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, PROGRAM CALL FAST, PROGRAM RETURN, or TEST ACCESS instruction is executed; otherwise, a special-operation exception is recognized.

When the ASF control is one:

- PC-number translation obtains the linkage-table designation from the primary ASN-second-table entry by first obtaining the primary-ASTE origin from control register 5, instead of obtaining the linkage-table designation from control register 5.
- PC-number translation treats the length of the entry-table entry as changed from 16 bytes to 32 bytes.
- ASN translation treats the boundary alignment and length of the ASN-second-table entry as changed from 16 bytes to 64 bytes.

Access-register translation always treats control register 5 as containing the primary-ASTE origin and always treats the ASN-second-table entry designated by an access-list entry as being 64 bytes, and, for these purposes, it does not examine the ASF control. However, when the ASF control is or has been zero, erroneous entries may exist in the ART-lookaside buffer (ALB), and, therefore, access-register translation may be performed erroneously; see "Formation of ALB Entries" on page 5-54.

Also when the ASF control is one:

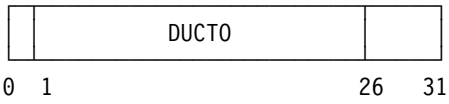
- PROGRAM CALL with space switching may obtain the address of an ASN-second-table entry from the entry-table entry used, instead of obtaining it by means of ASN translation.
- LOAD ADDRESS SPACE PARAMETERS, when it performs PASN translation, and also the space-switching forms of PROGRAM CALL and PROGRAM TRANSFER place the origin of the new primary ASTE in control register 5 instead of placing a linkage-table designation in that register. (PROGRAM RETURN requires that the ASF control be one. A space-switching PROGRAM RETURN also places the new primary-ASTE origin in control register 5.)



LzLabs GmbH. Ex. 1008-206

### Control Register 2

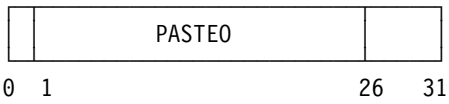
The location of the dispatchable-unit control table is specified in control register 2. The register has the following format:




***Dispatchable-Unit-Control-Table Origin (DUCTO):*** Bits 1-25 of control register 2, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the dispatchable-unit control table. Access-register translation may obtain the dispatchable-unit access-list designation from the dispatchable-unit control table.

### Control Register 5

The location of the primary ASN-second-table entry is specified in control register 5. The register has the following format:



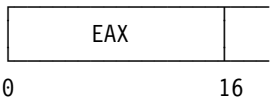

***Primary-ASTE Origin (PASTEO):*** Bits 1-25 of control register 5, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the primary ASN-second-table entry. Access-register translation may obtain the primary-space access-list designation from the primary ASTE. The primary-ASTE origin is set by LOAD ADDRESS SPACE PARAMETERS when it performs PASN translation and by the space-switching forms of PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, and PROGRAM TRANSFER. When any of these instructions places the primary-ASTE origin in control register 5, it also places zeros in bit positions 0 and 26-31 of control register 5.

When the ASF control is zero, LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, and PROGRAM TRANSFER treat control register 5 as containing the linkage-table designation. Access-register translation treats control register 5 as containing the primary-ASTE origin regardless of the value of the ASF control.

When control register 5 contains the primary-ASTE origin, bits 0 and 26-31 of the register are subject to possible future assignment, and they should not be depended upon to be zeros.

### Control Register 8

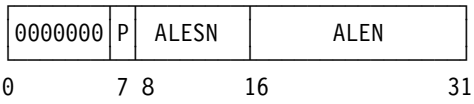
The extended authorization index is in control register 8. The register has the following format:




***Extended Authorization Index (EAX):*** Bits 0-15 of control register 8 are the extended authorization index. During access-register translation, the EAX may be compared against the access-list-entry authorization index (ALEAX) in an access-list entry, and it may be used as an index to locate a secondary bit in an authority table. The EAX may be set by a stacking PROGRAM CALL operation, and it is restored by PROGRAM RETURN.

## Access Registers

There are sixteen 32-bit access registers numbered 0-15. The contents of an access register are called an access-list-entry token (ALET). An ALET has the following format:




The fields in the ALET are allocated as follows:

***Primary-List Bit (P):*** When the ALET is not 00000000 or 00000001 hex, bit 7 specifies the access list to be used by access-register translation. When bit 7 is zero, the dispatchable-unit access list is used; this is specified by the dispatchable-unit access-list designation in the dispatchable-unit control table designated by the contents of control register 2. When bit 7 is one, the primary-space access list is used; this is specified by the primary-space access-list designation in the primary ASTE designated by the contents of control register 5.

***Access-List-Entry Sequence Number (ALESN):*** Bits 8-15 may be used as a check on whether the access-list entry designated by the ALET has been invalidated and reallocated since the ALET was obtained. During access-register translation when the ALET is not 00000000 or 00000001 hex, bits 8-15 of the ALET are com-


LzLabs GmbH. Ex. 1008-207

pared against the access-list-entry sequence number (ALESN) in the designated access-list entry.

***Access-List-Entry Number (ALEN):*** When the ALET is not 00000000 or 00000001 hex, bits 16-31 of the ALET designate an entry in either the dispatchable-unit access list or the primary-space access list, as determined by bit 7. The access-list designation that is used is called the effective access-list designation; it consists of the effective access-list origin and the effective access-list length.

During access-register translation, the ALEN, with four zeros appended on the right, is added to the 31-bit real or absolute address specified by the effective access-list origin, and the result is the real or absolute address of the designated access-list entry. The ALEN is compared against the effective access-list length to determine whether the designated access-list entry is within the list, and an ALEN-translation exception is recognized if the entry is outside the list. Although the largest possible value of the ALEN is 65,535, an access list can contain at most 1,024 or 4,096 entries, depending on the model.

Bits 0-6 must be zeros during access-register translation; otherwise, an ALET-specification exception is recognized.

When the ALET is 00000000 or 00000001 hex, it specifies the primary or secondary address space, respectively, and the above format does not apply.

Access register 0 usually is treated in access-register translation as containing 00000000 hex, and its actual contents are not examined; the access-register translation done as part of TEST ACCESS is the only exception. Access register 0 is also treated as containing 00000000 hex when it is designated by the B field of LOAD ADDRESS EXTENDED when PSW bits 16 and 17 are 01 binary. When access register 0 is specified for TEST ACCESS or as a source for COPY ACCESS, EXTRACT ACCESS, or STORE ACCESS MULTIPLE, the actual contents of the access register are used. Access register 0, like any other access register, can be loaded by COPY ACCESS, LOAD ACCESS MULTIPLE, LOAD ADDRESS EXTENDED, and SET ACCESS.

Another definition of ALETs 00000000 and 00000001 hex is given in “BRANCH IN SUB-SPACE GROUP” on page 10-13.

## Access-Register-Translation Tables

When the ALET being translated is not 00000000 or 00000001 hex, access-register translation performs a two-level lookup to locate first the effective access-list designation and then an entry in the effective access list. The effective access-list designation resides in real storage. The effective access list resides in real or absolute storage.

Access-register translation uses an origin in the access-list entry to locate an ASN-second-table entry, and it may perform a one-level lookup to locate an entry in an authority table. The ASN-second-table entry resides in real storage. The authority table resides in real or absolute storage.

Authority-table entries are described in “Authority-Table Entries” on page 3-24. Access-list designations, access-list entries, and ASN-second-table entries are described in the following sections.

### Dispatchable-Unit-Control Table and Access-List Designations

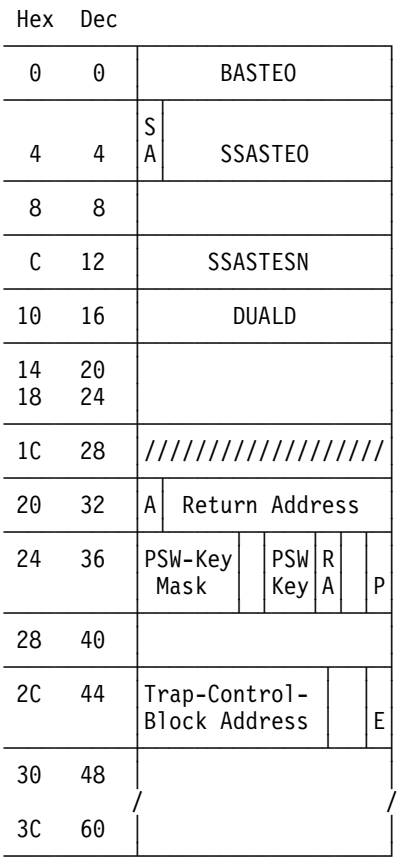
When the ALET being translated is not 00000000 or 00000001 hex, access-register translation obtains the dispatchable-unit access-list designation if bit 7 of the ALET is zero, or it obtains the primary-space access-list designation if bit 7 is one. The obtained access-list designation is called the effective access-list designation.

The dispatchable-unit access-list designation (DUALD) is located in bytes 16-19 of a 64-byte area called the dispatchable-unit control table (DUCT). The DUCT resides in real storage, and its location is specified by the DUCT origin in control register 2.

The dispatchable-unit control table has the following format:



LzLabs GmbH. Ex. 1008-208




Bytes 0-7 (BASTEO, SA, and SSASTEO) and 12-15 (SSASTESN) of the DUCT are described in "Subspace-Group Dispatchable-Unit Control Table" on page 5-56. Bytes 32-39 (A, return address, PSW key mask, PSW key, RA, and P) are described in "BRANCH AND SET AUTHORITY" on page 10-7. Bytes 44-47 (trap-control-block address and E) are described in "TRAP" on page 10-112. Bytes 8-11, 20-27, 40-43, and 48-63 are reserved for possible future extensions and should contain all zeros. Bytes 28-31 are available for use by programming.

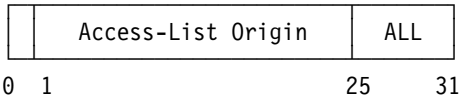
The primary-space access-list designation (PSALD) is located in bytes 16-19 of a 64-byte area called the primary ASN-second-table entry. The primary ASTE resides in real storage, and its location is specified by the primary-ASTE origin in control register 5. The format of the primary ASTE is described in "Extended ASN-Second-Table Entries" on page 5-47.

The dispatchable-unit and primary-space access-list designations both have the same format.

There are two possible formats of the access-list designation, called format 0 and format 1. A model implements one or the other of these two formats but not both; that is, the access-list-designation format that is available is model-dependent, and no control is provided by which the program can specify the format. A model provides no special indication of the format that it implements.

The two possible formats of the access-list designation are as follows.

Format-0 Access-List Designation



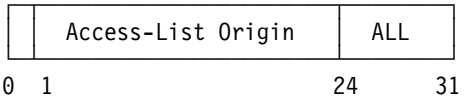

The fields in the format-0 access-list designation are allocated as follows:

***Access-List Origin:*** Bits 1-24 of the format-0 access-list designation, with seven zeros appended on the right, form a 31-bit address that designates the beginning of the access list. This address is treated unpredictably as either a real address or an absolute address.

***Access-List Length (ALL):*** Bits 25-31 of the format-0 access-list designation specify the length of the access list in units of 128 bytes, thus making the length of the access list variable in multiples of eight 16-byte entries. The length of the access list, in units of 128 bytes, is one more than the value in bit positions 25-31. The access-list length, with six zeros appended on the left, is compared against bits 0-12 of an access-list-entry number (bits 16-28 of an access-list-entry token) to determine whether the access-list-entry number designates an entry in the access list.

Bit 0 is reserved for a possible future extension and should be zero.

Format-1 Access-List Designation




The fields in the format-1 access-list designation are allocated as follows:


LzLabs GmbH. Ex. 1008-209

***Access-List Origin:*** Bits 1-23 of the format-1 access-list designation, with eight zeros appended on the right, form a 31-bit address that designates the beginning of the access list. This address is treated unpredictably as either a real address or an absolute address.

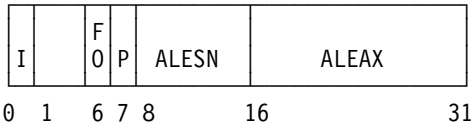
***Access-List Length (ALL):*** Bits 24-31 of the format-1 access-list designation specify the length of the access list in units of 256 bytes, thus making the length of the access list variable in multiples of sixteen 16-byte entries. The length of the access list, in units of 256 bytes, is one more than the value in bit positions 24-31. The access-list length, with four zeros appended on the left, is compared against bits 0-11 of an access-list-entry number (bits 16-27 of an access-list-entry token) to determine whether the access-list-entry number designates an entry in the access list.

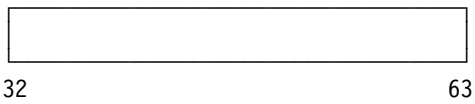
Bit 0 is reserved for a possible future extension and should be zero.

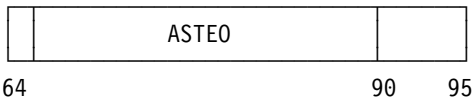
**Programming Note:** The maximum number of access-list entries allowed by a format-0 or format-1 access-list designation is 1,024 or 4,096, respectively. There are two access lists available for use at any time. Therefore, if a model implements the format-0 access-list designation, a maximum of 2,048 2G-byte address spaces can be addressable without control-program intervention, which is a total of 4T bytes; and if a model implements the format-1 access-list designation, a maximum of 8,192 2G-byte address spaces can be addressable without control-program intervention, which is a total of 16T bytes.

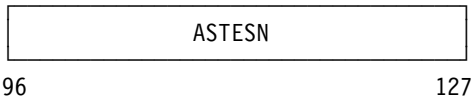
## Access-List Entries

The effective access list is the dispatchable-unit access list if bit 7 of the ALET being translated is zero, or it is the primary-space access list if bit 7 is one. The entry fetched from the effective access list is 16 bytes in length and has the following format:




The fields in the access-list entry are allocated as follows:

***ALEN-Invalid Bit (I):*** Bit 0, when zero, indicates that the access-list entry specifies an address space. When bit 0 is one during access-register translation, an ALEN-translation exception is recognized.

***Fetch-Only Bit (FO):*** Bit 6 controls which types of operand references are permitted to the address space specified by the access-list entry. When bit 6 is zero, both fetch-type and store-type references are permitted. When bit 6 is one, only fetch-type references are permitted, and an attempt to store causes a protection exception for access-list-controlled protection to be recognized and the operation to be suppressed.

***Private Bit (P):*** Bit 7, when zero, specifies that any program is authorized to use the access-list entry in access-register translation. When bit 7 is one, authorization is determined as described for bits 16-31.

***Access-List-Entry Sequence Number (ALESN):*** Bits 8-15 are compared against the ALESN in the ALET during access-register translation. Inequality causes an ALE-sequence exception to be recognized. It is intended that the control program change bits 8-15 each time it reallocates the access-list entry.

***Access-List-Entry Authorization Index (ALEAX):*** Bits 16-31 may be used to determine whether the program for which access-register translation is being performed is authorized to use the access-list entry. The program is authorized if any of the following conditions is met:

1. Bit 7 is zero.
2. Bits 16-31 are equal to the extended authorization index (EAX) in control register 8.



LzLabs GmbH. Ex. 1008-210

3. The EAX selects a secondary bit that is one in the authority table for the specified address space.

An extended-authority exception is recognized if none of the conditions is met.

***ASN-Second-Table-Entry Origin (ASTEO):*** Bits 65-89, with six zeros appended on the right, form the 31-bit real address of the ASTE for the specified address space. Access-register translation obtains the segment-table designation for the address space from the ASTE.

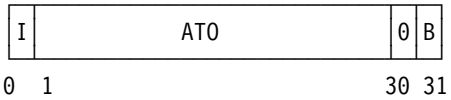
***ASTE Sequence Number (ASTESN):*** Bits 96-127 may be used to revoke the addressing capability represented by the access-list entry. Bits 96-127 are compared against an ASTE sequence number (ASTESN) in the designated ASTE during access-register translation.

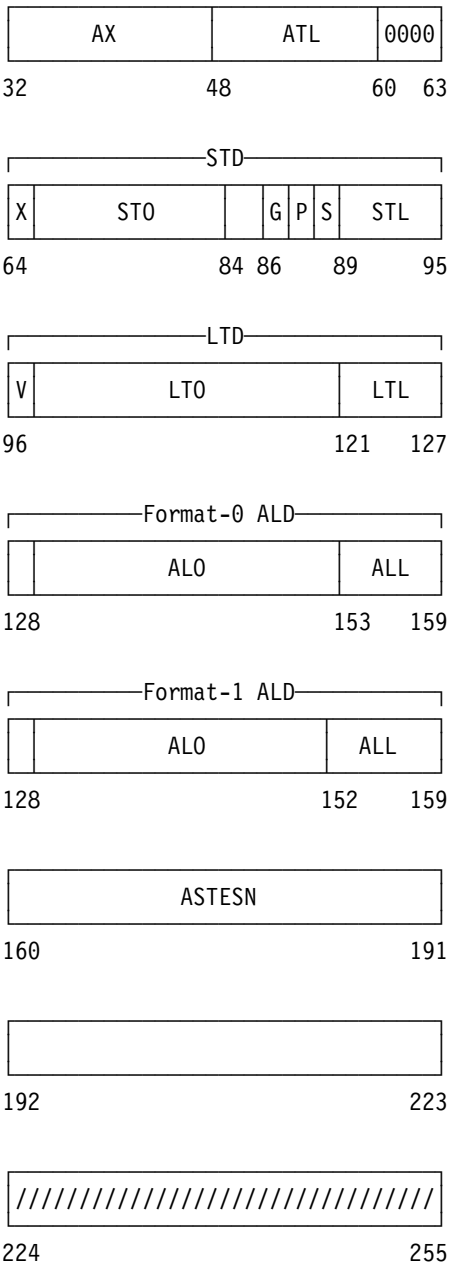
Bits 1-5, 32-64, and 90-95 are reserved for possible future extensions and should be zeros.

In both the dispatchable-unit access list and the primary-space access list, access-list entries 0 and 1 are intended not to be used in access-register translation. Bits 1-127 of access-list entry 0 and bits 1-63 of access-list entry 1 are reserved for possible future extensions and should be zeros. Bit 0 of access-list entries 0 and 1, and bits 64-127 of access-list entry 1, are available for use by programming. The control program should set bit 0 of access-list entries 0 and 1 to one in order to prevent the use of these entries by means of ALETs in which the ALEN is 0 or 1.

## Extended ASN-Second-Table Entries

When the ASF control is one, the length of each entry in the ASN second table is extended from 16 bytes to 64 bytes when the table is used in ASN translation. Also, the ASN second table begins on a 64-byte boundary instead of a 16-byte boundary. Access-register translation, which does not involve ASN translation, always treats the ASN-second-table entry as being 64 bytes on a 64-byte boundary, and access-register translation does not examine the ASF control. The first 32 bytes of the 64-byte ASTE have the following format:




The fields in bit positions 0-127 of the ASTE are defined with respect to certain mechanisms and instructions in “ASN-Second-Table Entries” on page 3-19. The fields in the ASTE are defined with respect to the BRANCH IN SUBSPACE GROUP instruction in “Subspace-Group ASN-Second-Table Entries” on page 5-57. With respect to access-register translation only, and only for an instruction other than BRANCH IN SUBSPACE GROUP, the fields in the ASTE are allocated as follows:


LzLabs GmbH. Ex. 1008-211

***ASX-Invalid Bit (I):*** Bit 0 controls whether the address space associated with the ASTE is available. When bit 0 is zero, access-register translation proceeds. When the bit is one, an ASTE-validity exception is recognized.

***Authority-Table Origin (ATO):*** Bits 1-29, with two zeros appended on the right, form a 31-bit address that designates the beginning of the authority table. This address is treated unpredictably as either a real address or an absolute address, although it is treated as a real address for ASN authorization. The authority table is accessed in access-register translation only if the private bit in the access-list entry is one and the access-list-entry authorization index (ALEAX) in the access-list entry is not equal to the extended authorization index (EAX) in control register 8.

***Base-Space Bit (B):*** Bit 31 is ignored during access-register translation if the subspace-group facility is installed and the ASF control is one. If the subspace-group facility is not installed or the ASF control is zero, bit 31 must be zero; otherwise, an ASN-translation-specification exception may be recognized. Bit 31 is further described in "Subspace-Group ASN-Second-Table Entries" on page 5-57.

***Authorization Index (AX):*** Bits 32-47 are not used in access-register translation.

***Authority-Table Length (ATL):*** Bits 48-59 specify the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four bytes, is one more than the ATL value. The contents of the ATL field are used to establish whether the entry designated by a particular EAX is within the authority table. An extended-authority exception is recognized if the entry is not within the table.

***Segment-Table Designation (STD):*** Bits 65-95 are obtained as the result of access-register translation and are used by DAT to translate the logical address for the storage-operand reference being made. Bit 64, the space-switch-event control, is not used in or as a result of access-register translation.

***Linkage-Table Designation (LTD):*** Bits 96-127 are not used in access-register translation.

***Access-List Designation (ALD):*** When this ASTE is designated by the primary-ASTE origin in control register 5, bits 128-159 are the primary-space access-list designation (PSALD). See the description of the access-list designation in "Dispatchable-Unit-Control Table and Access-List Designations" on page 5-44. During access-register translation when the primary-list bit, bit 7, in the ALET being translated is one, the PSALD is the effective access-list designation. The PSALD is a format-0 ALD or a format-1 ALD, depending on the model.

***ASN-Second-Table-Entry Sequence Number (ASTESN):*** Bits 160-191 are used to control revocation of the accessing capability represented by access-list entries that designate the ASTE. During access-register translation, bits 160-191 are compared against the ASTESN in the access-list entry, and inequality causes an ASTE-sequence exception to be recognized. It is intended that the control program change the value of bits 160-191 when the authorization policies for the address space specified by the ASTE change or when the ASTE is reassigned to specify another address space.

Bits 30, 31, and 60-63 must be zeros during access-register translation if the authority table is to be accessed; otherwise, an ASN-translation-specification exception may be recognized.

Bits 84, 85, 128, and 192-223 are reserved for possible future extensions and should be zeros. Bits 224-255 are available for use by programming. The second 32 bytes of the 64-byte ASTE also are reserved for possible future extensions and should contain all zeros.

## Access-Register-Translation Process

This section describes the access-register-translation process as it is performed during a storage-operand reference in the access-register mode. LOAD REAL ADDRESS when PSW bits 16 and 17 are 01 binary, TEST ACCESS in any translation mode, and TEST PROTECTION in the access-register mode, perform access-register translation the same as described here, except that the following exceptions cause a setting of the condition code



LzLabs GmbH. Ex. 1008-212

instead of being treated as program-interruption conditions:

- ALET specification
- ALEN translation
- ALE sequence
- ASTE validity
- ASTE sequence
- Extended authority

BRANCH IN SUBSPACE GROUP performs access-register translation as described in "BRANCH IN SUBSPACE GROUP" on page 10-13.

Access-register translation operates on the access register designated in a storage-operand reference in order to obtain a segment-table designation for use by DAT. When one of access-registers 1-15 is designated, the access-list-entry token (ALET) that is in the access register is used to obtain the segment-table designation. When access register 0 is designated, an ALET having the value 00000000 hex is used, except that TEST ACCESS uses the actual contents of access register 0.

When the ALET is 00000000 or 00000001 hex, the primary or secondary segment-table designation, respectively, is obtained.

When the ALET is other than 00000000 or 00000001 hex, the leftmost seven bits of the ALET are checked for zeros, the primary-list bit in the ALET and the contents of control register 2 or 5 are used to obtain the effective access-list designation, and the access-list entry number (ALEN) in the ALET is used to select an entry in the effective access list.

The access-list entry is checked for validity and for containing the correct access-list-entry sequence number (ALESN).

The ASN-second-table entry (ASTE) addressed by the access-list entry is checked for validity and for containing the correct ASN-second-table-entry sequence number (ASTESN).

Whether the program is authorized to use the access-list entry is determined through the use of one or more of: (1) the private bit and access-list-entry authorization index (ALEAX) in the access-list entry, (2) the extended authorization index (EAX) in control register 8, and (3) an entry in the authority table addressed by the ASN-second-table entry.

If a store-type reference is to be performed, the fetch-only bit in the access-list entry is checked for being zero.

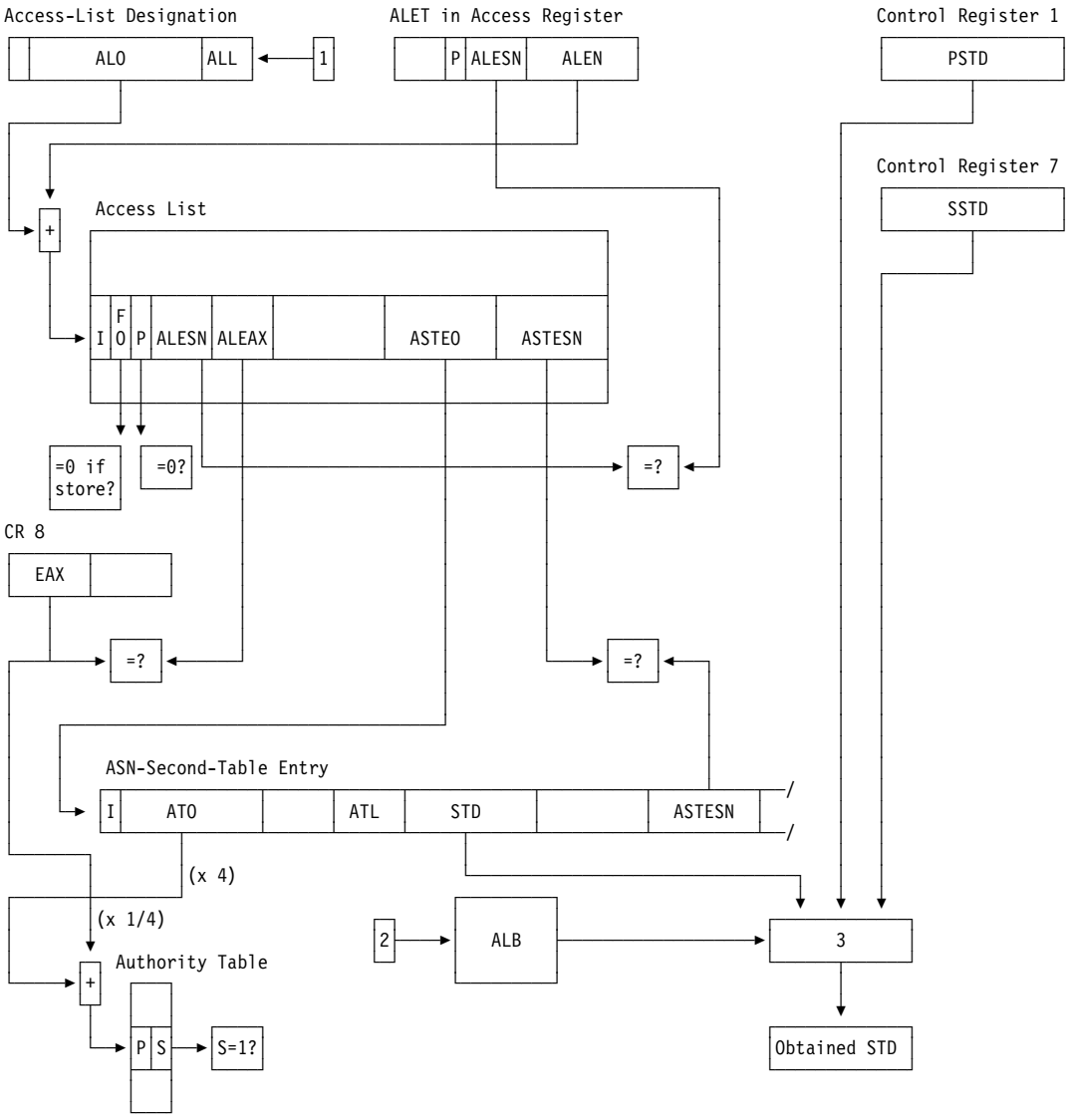
When no exceptions are recognized, the segment-table designation in the ASN-second-table entry is obtained.

In order to avoid the delay associated with references to real or absolute storage, the information fetched from real or absolute storage normally is also placed in a special buffer, the ART-lookaside buffer (ALB), and subsequent translations involving the same information may be performed by using the contents of the ALB. The operation of the ALB is described in "ART-Lookaside Buffer" on page 5-53.

Whenever access to real or absolute storage is made during access-register translation for the purpose of fetching an entry from an access-list-designation source, access list, ASN second table, or authority table, key-controlled protection does not apply.

The principal features of access-register translation, including the effect of the ALB, are shown in Figure 5-9 on page 5-50.


LzLabs GmbH. Ex. 1008-213




**Explanation:**

1. The appropriate ALD is obtained:
   When P in the ALET is zero (and the ALET is not zero or one), the DUALD in the DUCT is obtained.
   When P in the ALET is one, the PSALD in the primary ASTE is obtained.

2. Information, which may include the ALD-source origin, ALET, ALO, and EAX, is used to search the ALB. This information, along with information from the ALE, ASTE, and ATE, may be placed in the ALB.

3. The appropriate STD is obtained:
   When the ALET is zero, the PSTD in CR 1 is obtained.
   When the ALET is one, the SSTD in CR 7 is obtained.
   When the ALET is larger than one:
   If a match exists, the STD from the ALB is used.
   If no match exists, tables from real or absolute storage are fetched. The resulting STD from the ASTE is obtained, and entries may be formed in the ALB.

*Figure 5-9. Access-Register Translation*



LzLabs GmbH. Ex. 1008-214

### Selecting the Access-List-Entry Token

When one of access registers 1-15 is designated, or for the access register designated by the $R_1$ field of TEST ACCESS, access-register translation uses the access-list-entry token (ALET) that is in the access register. When access register 0 is designated, except for TEST ACCESS, an ALET having the value 00000000 hex is used, and the contents of access register 0 are not examined.

### Obtaining the Primary or Secondary Segment-Table Designation

When the ALET being translated is 00000000 hex, the primary segment-table designation in control register 1 is obtained. When the ALET is 00000001 hex, the secondary segment-table designation in control register 7 is obtained. In each of these two cases, access-register translation is completed.

### Checking the First Byte of the ALET

When the ALET being translated is other than 00000000 or 00000001 hex, bits 0-6 of the ALET are checked for being all zeros. If bits 0-6 are not all zeros, an ALET-specification exception is recognized, and the operation is suppressed.

### Obtaining the Effective Access-List Designation

The primary-list bit, bit 7, in the ALET is used to perform a lookup to obtain the effective access-list designation. When bit 7 is zero, the effective ALD is the dispatchable-unit ALD located in bytes 16-19 of the dispatchable-unit control table (DUCT). When bit 7 is one, the effective ALD is the primary-space ALD located in bytes 16-19 of the primary ASN-second-table entry (primary ASTE).

When bit 7 is zero, the 31-bit real address of the dispatchable-unit ALD is obtained by appending six zeros on the right to the DUCT origin, bits 1-25 of control register 2, and adding 16. The addition cannot cause a carry into bit position 0.

When bit 7 is one, the 31-bit real address of the primary-space ALD is obtained by appending six zeros on the right to the primary-ASTE origin, bits 1-25 of control register 5, and adding 16. The addition cannot cause a carry into bit position 0.

The obtained 31-bit real address is used to fetch the effective ALD — either the dispatchable-unit ALD or the primary-space ALD, depending on bit 7 of the ALET. The fetch of the effective ALD appears to be word concurrent, as observed by other CPUs, and is not subject to protection. When the storage address that is generated for fetching the effective ALD refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed. When the primary-space ALD is fetched, bit 0, the ASX-invalid bit, and bits 30, 31, and 60-63 in the primary ASTE are ignored.

### Access-List Lookup

A lookup in the effective access list is performed. The effective access list is the dispatchable-unit access list if bit 7 of the ALET is zero, or it is the primary-space access list if bit 7 is one. The effective access list is treated unpredictably as being in either real or absolute storage.

The access-list-entry-number (ALEN) portion of the ALET is used to select an entry in the effective access list. If the format-0 ALD is implemented, the real or absolute address of the access-list entry is obtained by appending seven zeros on the right to bits 1-24 of the effective ALD and adding the ALEN to this value. If the format-1 ALD is implemented, the real or absolute address of the access-list entry is obtained by appending eight zeros on the right to bits 1-23 of the effective ALD and adding the ALEN to this value. For these additions, the ALEN is extended with four rightmost zeros and 11 leftmost zeros. In either case, when a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the access list to wrap from $2^{31} - 1$ to 0. The result is a 31-bit real or absolute address.

As part of the access-list-lookup process if the format-0 ALD is implemented, the leftmost 13 bits of the ALEN are compared against the effective access-list length, bits 25-31 of the effective ALD, to establish whether the addressed entry is within the access list. For this comparison, the access-list length is extended with six leftmost zeros. If the value formed from the access-list length is less than the value in the 13 leftmost bits of the ALEN, an ALEN-translation exception is recognized, and the operation is nullified. If the format-1 ALD is implemented, the leftmost 12 bits of the ALEN are compared against bits 24-31 of the effective ALD. For this comparison, the


LzLabs GmbH. Ex. 1008-215

access-list length is extended with four leftmost zeros. If the value formed from the access-list length is less than the value in the 12 leftmost bits of the ALEN, an ALEN-translation exception is recognized, and the operation is nullified.

The 16-byte access-list entry is fetched by using the real or absolute address. The fetch of the entry appears to be word concurrent as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining three words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address that is generated for fetching the access-list entry refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the access-list entry indicates whether the access-list entry specifies an address space by designating an ASN-second-table entry. This bit is inspected, and, if it is one, an ALEN-translation exception is recognized, and the operation is nullified.

When bit 0 is zero, the access-list-entry sequence number (ALESN) in bit positions 8-15 of the access-list entry is compared against the ALESN in the ALET to determine whether the ALET designates the conceptually correct access-list entry. Inequality causes an ALE-sequence exception to be recognized and the operation to be nullified.

## Locating the ASN-Second-Table Entry

The ASN-second-table-entry (ASTE) origin in the access-list entry is used to locate the ASTE. Bits 65-89 of the access-list entry, with six zeros appended on the right, form the 31-bit real address of the ASTE.

The 64-byte ASTE is fetched by using the real address. The fetch of the entry appears to be word concurrent as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address that is generated for fetching the ASTE refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the ASTE indicates whether the ASTE specifies an address space. This bit is inspected, and, if it is one, an ASTE-validity exception is recognized, and the operation is nullified.

When bit 0 is zero, the ASTE sequence number (ASTESN) in bit positions 160-191 of the ASTE is compared against the ASTESN in bit positions 96-127 of the access-list entry to determine whether the addressing capability represented by the access-list entry has been revoked. Inequality causes an ASTE-sequence exception to be recognized and the operation to be nullified.

## Authorizing the Use of the Access-List Entry

The private bit, bit 7, in the access-list entry is used to determine whether the program is authorized to use the access-list entry. The access-list-entry authorization index (ALEAX) in bit positions 16-31 of the access-list entry, the extended authorization index (EAX) in bit positions 0-15 of control register 8, and the authority table designated by the ASTE may also be used.

When the private bit is zero, the program is authorized, and the authorization step of access-register translation is completed.

When the private bit is one but the ALEAX is equal to the EAX, the program is authorized, and the authorization step of access-register translation is completed.

When the private bit is one and the ALEAX is not equal to the EAX, bits 30, 31, and 60-63 of the ASTE must be zeros; otherwise, an ASN-translation-specification exception may be recognized, which would cause the operation to be suppressed. A one value of bit 31 does not cause an exception to be recognized if the subspace-group facility is installed and the ASF control is one.

When the private bit is one and the ALEAX is not equal to the EAX, a process called the extended-authorization process is performed. Extended authorization uses the EAX to select an entry in the authority table designated by the ASTE, and it tests the secondary-authority bit in the selected entry for being one. The program is authorized if the tested bit is one.



LzLabs GmbH. Ex. 1008-216

Extended authorization is the same as the secondary-ASN-authorization process described in“ASN Authorization” on page 3-23 , except as follows:

- The authority-table origin is treated as a real or absolute address instead of as a real address.
- The EAX in control register 8 is used instead of the authorization index (AX) in control register 4.
- When the value in bit positions 0-11 of the EAX is greater than the authority-table length (ATL) in the ASTE, an extended-authority exception is recognized instead of a secondary-authority exception. The operation is nullified if the extended-authority exception is recognized.

When the private bit is one, the ALEAX is not equal to the EAX, and the secondary bit in the authority-table entry selected by the EAX is not one, an extended-authority exception is recognized, and the operation is nullified.

### Checking for Access-List-Controlled Protection

If a store-type reference is to be performed and the fetch-only bit, bit 6, in the access-list entry is one, a protection exception is recognized, and the operation is suppressed.

### Obtaining the Segment-Table Designation from the ASN-Second-Table Entry

When the ALET being translated is other than 00000000 or 00000001 hex and no exception is recognized in the steps described above, access-register translation obtains the segment-table designation from bit positions 65-95 of the ASTE. Bit 64 of the ASTE, the space-switch-event control, is ignored.

### Recognition of Exceptions during Access-Register Translation

The exceptions which can be encountered during the access-register-translation process and their priority are shown in the section “Access Exceptions” in Chapter 6, “Interruptions.”

**Programming Note:** When updating an access-list entry or ASN-second-table entry, the program should change the entry from invalid to valid (set bit 0 of the entry to zero) as the last step of the updating. This ensures, because the leftmost word is fetched first, that words of a partially updated entry will not be fetched.

## ART-Lookaside Buffer

To enhance performance, the access-register-translation (ART) mechanism normally is implemented such that access-list designations and information specified in access lists, ASN second tables, and authority tables are maintained in a special buffer, referred to as the ART-lookaside buffer (ALB). Access-list designations, access-list entries, ASN-second-table entries, and authority-table entries are collectively referred to as ART-table entries. The CPU necessarily refers to an ART-table entry in real or absolute storage only for the initial access to that entry. The information in the entry may be placed in the ALB, and subsequent ART operations may be performed using the information in the ALB. The presence of the ALB affects the ART process to the extent that (1) a modification of an ART-table entry in real or absolute storage does not necessarily have an immediate effect, if any, on the translation, (2) the comparison against the access-list length in an access-list designation that is in storage and used in a translation may be omitted if an ALB access-list entry is used, and (3) the comparison against the authority-table length in an ASN-second-table entry that is in storage and used in a translation may be omitted if an ALB authority-table entry is used. In a multiple-CPU configuration, each CPU has its own ALB.

Entries within the ALB are not explicitly addressable by the program.

Information is not necessarily retained in the ALB under all conditions for which such retention is possible. Furthermore, information in the ALB may be cleared under conditions additional to those for which clearing is mandatory.

### ALB Structure

The description of the logical structure of the ALB covers the implementation by all systems operating as defined by ESA/390. The ALB entries are considered as being of four types: ALB access-list designations (ALB ALDs), ALB access-list entries (ALB ALEs), ALB ASN-second-table entries (ALB ASTEs), and ALB authority-table


LzLabs GmbH. Ex. 1008-217

entries (ALB ATEs). An ALB entry is considered as containing within it both the information obtained from the ART-table entry in real or absolute storage and the attributes used to fetch the ART-table entry from real or absolute storage. There is not an indication in an ALB ALD of whether the ALD-source origin used to select the ALD in real storage was the dispatchable-unit-control-table origin or the primary-ASTE origin.

**Note:** The following sections describe the conditions under which information may be placed in the ALB, the conditions under which information from the ALB may be used for access-register translation, and how changes to the tables affect the ART process.

## Formation of ALB Entries

The formation of ALB entries and the effect of any manipulation of the contents of an ART-table entry in real or absolute storage by the program depend on whether the entry is attached to a particular CPU and on whether the entry is valid.

The *attached* state of an ART-table entry denotes that the CPU to which the entry is attached can attempt to use the entry for access-register translation. The ART-table entry may be attached to more than one CPU at a time.

An access-list entry or ASN-second-table entry is *valid* when the invalid bit associated with the entry is zero. Access-list designations and authority-table entries have no invalid bit and are always valid. The primary-space access-list designation is valid regardless of the value of the invalid bit in the primary ASTE.

An ART-table entry may be placed in the ALB whenever the entry is attached and valid.

An access-list designation is attached to a CPU when the designation is within the dispatchable-unit control table designated by the dispatchable-unit-control-table origin in control register 2 or is within the primary ASTE designated by the primary-ASTE origin in control register 5. Control register 5 is considered to contain the primary-ASTE origin regardless of the value of the address-space-function (ASF) control, bit 15 of control register 0; however, see the note below.

An access-list entry is attached to a CPU when the entry is within the access list specified by either an attached access-list designation (ALD) or a usable ALB ALD. A usable ALB ALD is explained in the next section.

An ASN-second-table entry is attached to a CPU when it is designated by the ASTE origin in either an attached and valid access-list entry (ALE) or a usable ALB ALE. A usable ALB ALE is explained in the next section.

An authority-table entry is attached to a CPU when it is within the authority table designated by either an attached and valid ASN-second-table entry (ASTE) or a usable ALB ASTE. A usable ALB ASTE is explained in the next section.

**Note:** During the execution of a PROGRAM CALL, PROGRAM TRANSFER, or LOAD ADDRESS SPACE PARAMETERS instruction that loads control register 5 when the ASF control is zero, an unpredictable access-list-designation (ALD) may be placed in the ALB. This unpredictable ALB ALD may then be usable to place other entries (ALE, ASTE, and ATE) in the ALB. If access-register translation uses any of these erroneous ALB entries, the results are unpredictable. These specific erroneous entries are removed from the ALB either by clearing the entire ALB or by the execution of (1) a PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, PROGRAM TRANSFER, or LOAD ADDRESS SPACE PARAMETERS instruction that loads control register 5 when the ASF control is one, or (2) a LOAD CONTROL instruction that loads control register 5, regardless of the value of the ASF control.

## Use of ALB Entries

The *usable* state of an ALB entry denotes that the CPU can attempt to use the ALB entry for access-register translation. A usable ALB entry attaches the next-lower-level table, if any, and may be usable for a particular instance of access-register translation.

An ALB ALD is in the usable state when the ALDSO field in the ALB ALD matches the current dispatchable-unit-control-table origin or the current primary-ASTE origin.

An ALB ALD may be used for a particular instance of access-register translation when either of the following conditions is met:



LzLabs GmbH. Ex. 1008-218

1. The primary-list bit in the ALET to be translated is zero, and the ALDSO field in the ALB ALD matches the current dispatchable-unit-control-table origin.
2. The primary-list bit in the ALET to be translated is one, and the ALDSO field in the ALB ALD matches the current primary-ASTE origin.

An ALB ALE is in the usable state when the ALO field in the ALB ALE matches the ALO field in an attached ALD or a usable ALB ALD.

An ALB ALE may be used for a particular instance of access-register translation when all of the following conditions are met:

1. The ALET to be translated has a value larger than 1. (If the ALET is 0 or 1, the contents of CR 1 or CR 7 are used.)
2. The ALO field in the ALB ALE matches the ALO field in the ALD or ALB ALD being used in the translation.
3. The ALEN field in the ALB ALE matches the ALEN field in the ALET to be translated.

An ALB ASTE is in the usable state when the ASTEO field in the ALB ASTE matches the ASTEO field in an attached and valid ALE or a usable ALB ALE.

An ALB ASTE may be used for a particular instance of access-register translation when the ASTEO field in the ALB ASTE matches the ASTEO field in the ALE or ALB ALE being used in the translation.

An ALB ATE may be used for a particular instance of access-register translation when both of the following conditions are met:

1. The ATO field in the ALB ATE matches the ATO field in the ASTE or ALB ASTE being used in the translation.
2. The EAX field in the ALB ATE matches the current EAX.

### Modification of ART Tables

When an attached but invalid ART-table entry is made valid, or when an unattached but valid ART-table entry is made attached, and no entry formed from the ART-table entry is already in the ALB, the change takes effect no later than the end of the current instruction.

When an attached and valid ART-table entry is changed, and when, before the ALB is cleared of copies of that entry, an attempt is made to perform ART requiring that entry, unpredictable results may occur, to the following extent. The use of the new value may begin between instructions or during the execution of an instruction, including the instruction that caused the change. Moreover, until the ALB is cleared of copies of the entry, the ALB may contain both the old and the new values, and it is unpredictable whether the old or new value is selected for a particular ART operation. If the old and new values are used as representations of effective space designations, failure to recognize that the effective space designations are the same may occur, with the result that operand overlap may not be recognized. Effective space designations and operand overlap are discussed in “Interlocks within a Single Instruction” on page 5-80.

When LOAD ACCESS MULTIPLE or LOAD CONTROL changes the parameters associated with ART, the values of these parameters at the start of the operation are in effect for the duration of the operation.

All entries are cleared from the ALB by the execution of PURGE ALB, a COMPARE AND SWAP AND PURGE instruction that purges the ALB, and SET PREFIX, and by CPU reset.

## Subspace Groups

The subspace-group facility provides the BRANCH IN SUBSPACE GROUP instruction, new allocations of fields in the segment-table designation, dispatchable-unit control table, and extended ASN-second-table entry, and new operations, called subspace-replacement operations, of the PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS instructions. BRANCH IN SUBSPACE GROUP is introduced in “Subroutine Linkage without the Linkage Stack” on page 5-10 and described in detail in “BRANCH IN SUBSPACE GROUP” on page 10-13.


LzLabs GmbH. Ex. 1008-219

## Subspace-Group Tables

This section describes the use of the dispatchable-unit control table and ASN-second-table entry by the subspace-group facility.

### Subspace-Group Dispatchable-Unit Control Table

The first 32 bytes of the 64-byte dispatchable-unit control table have the following format when the subspace-group facility is installed:

| Hex | Dec | | |
|---|---|---|---|
| 0 | 0 | BASTEO | |
| 4 | 4 | S A | SSASTEO |
| 8 | 8 | | |
| C | 12 | SSASTESN | |
| 10 | 16 | DUALD | |
| 14<br>18 | 20<br>24 | | |
| 1C | 28 | //////////////////// | |

The fields in the dispatchable-unit control table are allocated as follows:

***Base-ASTE Origin (BASTEO):*** Bits 1-25 of bytes 0-3, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the ASN-second-table entry that specifies the base space of a subspace group associated with the dispatchable unit. A comparison of bits 1-25 of bytes 0-3 to the primary-ASTE origin (PASTEO) in control register 5 is made by BRANCH IN SUBSPACE GROUP to determine whether the current primary address space is in the subspace group for the current dispatchable unit. For this comparison, either bits 1-25 may be compared to the PASTEO or the entire contents of bytes 0-3 may be compared to the entire contents of control register 5. A comparison of bits 1-25 of bytes 0-3 to the destination-ASTE origin (DASTEO) obtained from an access-list entry by access-register translation of an ALET other than ALETs 0 and 1 is made by BRANCH IN SUBSPACE GROUP to determine if the destination ASTE is the base-space ASTE. For this comparison, either bits 1-25 may be compared to the DASTEO or the entire contents of bytes 0-3 may be compared to the DASTEO with one leftmost and six rightmost zeros appended. A comparison of bits 1-25 of bytes 0-3 to an ASTE origin (ASTEO) obtained by ASN translation may be made by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS. For this comparison, either bits 1-25 may be compared to the ASTEO or the entire contents of bytes 0-3 may be compared to the ASTEO with one leftmost and six rightmost zeros appended. When BRANCH IN SUBSPACE GROUP uses ALET 0, bits 1-25 of bytes 0-3, with six zeros appended on the right, designate the destination ASTE.

***Subspace-Active Bit (SA):*** Bit 0 of bytes 4-7 indicates, when one, that the last BRANCH IN SUBSPACE GROUP instruction executed for the dispatchable unit transferred control to a subspace of the subspace group associated with the dispatchable unit. Bit 0 being zero indicates any one of the following: the last BRANCH IN SUBSPACE GROUP instruction executed for the dispatchable unit transferred control to the base space of the subspace group, BRANCH IN SUBSPACE GROUP has not yet been executed for the dispatchable unit, or the dispatchable unit is not associated with a subspace group. BRANCH IN SUBSPACE GROUP sets bit 0 of bytes 4-7 to one when it transfers control to a subspace of the subspace group associated with the dispatchable unit, and it sets bit 0 to zero when it transfers control to the base space of the subspace group.

***Subspace-ASTE Origin (SSASTEO):*** Bits 1-25 of bytes 4-7, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the ASN-second-table entry that specifies the subspace last given control by a BRANCH IN SUBSPACE GROUP instruction executed for the dispatchable unit. When BRANCH IN SUBSPACE GROUP transfers control to a subspace by means of an ALET other than ALET 1, it places the ASTEO for the subspace (the destination ASTEO) in bit positions 1-25 of bytes 4-7, places zeros in bit positions 26-31 of bytes 4-7, and sets the subspace-active bit, bit 0 of bytes 4-7, to one. When BRANCH IN SUBSPACE GROUP uses ALET 1 to transfer control to a subspace, bits 1-25 of bytes 4-7, with six zeros appended on the right, designate the destination ASTE, and BRANCH IN SUBSPACE GROUP sets



LzLabs GmbH. Ex. 1008-220

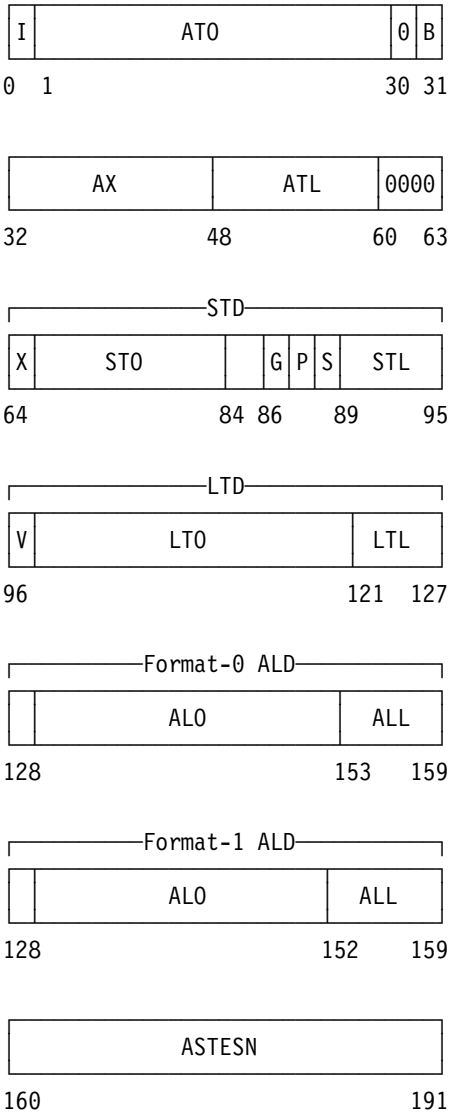
the subspace-active bit to one and either sets bits 26-31 of bytes 4-7 to zeros or leaves those bits unchanged. However, if bits 1-25 are all zeros, a special-operation exception is recognized. When BRANCH IN SUBSPACE GROUP transfers control to the base space of the subspace group, it sets the subspace-active bit to zero, and bits 1-31 of bytes 4-7 remain unchanged. Bits 1-25 of bytes 4-7 may be used by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS to set bits 1-23 and 25-31 of the primary STD in control register 1 or the secondary STD in control register 7 from the same bits of the STD in the subspace ASTE.

***Subspace-ASTE Sequence Number (SSASTESN):*** Bytes 12-15 may be used to revoke the linkage capability represented by the SSASTEO, bits 1-25 of bytes 4-7, in the DUCT. When BRANCH IN SUBSPACE GROUP transfers control to a subspace by means of an ALET other than ALET 1, it obtains the ASTESN in the subspace ASTE and places it in bytes 12-15. When BRANCH IN SUBSPACE GROUP uses ALET 1 to transfer control to a subspace, it compares bytes 12-15 to the ASTESN in the subspace ASTE, and it recognizes an ASTE-sequence exception if they are unequal. When the SSASTEO is used by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS to set bits 1-23 and 25-31 of the primary STD in control register 1 or the secondary STD in control register 7 from the same bits of the STD in the subspace ASTE, those instructions first compare bytes 12-15 to the ASTESN in the subspace ASTE, and they recognize an ASTE-sequence exception if the two fields are unequal.

Bytes 16-19 are described in “Dispatchable-Unit-Control Table and Access-List Designations” on page 5-44. Bytes 32-39 are described in “BRANCH AND SET AUTHORITY” on page 10-7. Bytes 44-47 are described in “TRAP” on page 10-112. Bytes 8-11, 20-27, 40-43, and 48-63 are reserved for possible future extensions and should contain all zeros. Bytes 28-31 are available for use by programming.

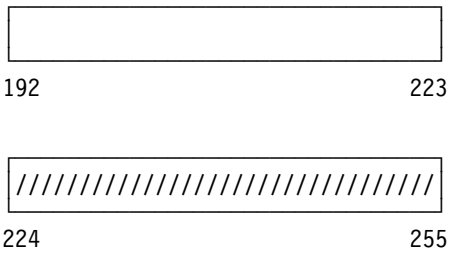
## Subspace-Group ASN-Second-Table Entries

When the ASF control is one, the length of each entry in the ASN second table is extended from 16 bytes to 64 bytes when the table is used in ASN translation. Also, the ASN second table begins on a 64-byte boundary instead of a 16-byte boundary. Access-register translation, which does not involve ASN translation, always treats the ASN-second-table entry as being 64 bytes on a 64-byte boundary, and access-register translation does not examine the ASF control. BRANCH IN SUBSPACE GROUP requires that the ASF control be one. The first 32 bytes of the 64-byte ASTE have the following format:





LzLabs GmbH. Ex. 1008-221




The fields in bit positions 0-127 of the ASTE are defined with respect to certain mechanisms and instructions in "ASN-Second-Table Entries" on page 3-19. The fields in the ASTE are defined for access-register translation for other than BRANCH IN SUBSPACE GROUP in "Extended ASN-Second-Table Entries" on page 5-47. For BRANCH IN SUBSPACE GROUP only, the fields in the ASTE are allocated as follows:

***ASX-Invalid Bit (I):*** Bit 0 controls whether the address space associated with the ASTE is available. When bit 0 is zero during access-register translation of ALET 1 or an ALET other than 0 and 1 for BRANCH IN SUBSPACE GROUP, the translation proceeds. When the bit is one, an ASTE-validity exception is recognized. The bit is ignored during access-register translation of ALET 0. When the ASTE is designated by a subspace-ASTE origin (SSASTEO) in a dispatchable-unit control table, bit 0 is also used as described in the definition of bits 160-191 (ASTESN).

***Authority-Table Origin (ATO):*** Bits 1-29 are not used by BRANCH IN SUBSPACE GROUP.

***Base-Space Bit (B):*** Bit 31 specifies, when one, that the address space associated with the ASTE is the base space of a subspace group. When BRANCH IN SUBSPACE GROUP uses an ALET other than ALETs 0 and 1 to locate a destination ASTE, it recognizes a special-operation exception if the destination-ASTE origin does not equal the base-ASTE origin in the dispatchable-unit control table and bit 31 is one in the destination ASTE.

***Authorization Index (AX):*** Bits 32-47 are not used by BRANCH IN SUBSPACE GROUP.

***Authority-Table Length (ATL):*** Bits 48-59 are not used by BRANCH IN SUBSPACE GROUP.

***Segment-Table Designation (STD):*** Bits 64-95 are obtained as the result of access-register translation done for BRANCH IN SUBSPACE GROUP. When BRANCH IN SUBSPACE GROUP uses an ALET other than ALETs 0 and 1 to locate a destination ASTE, it recognizes a special-operation exception if the destination-ASTE origin does not equal the base-ASTE origin in the dispatchable-unit control table and the subspace-group-control bit, bit 86 (G), in the destination ASTE is zero. When BRANCH IN SUBSPACE GROUP transfers control to the base space of a subspace group associated with the current dispatchable unit, it places bits 64-95 in control register 1; otherwise, when BRANCH IN SUBSPACE GROUP transfers control to a subspace of the subspace group, it places bits 65-87 and 89-95 in the corresponding bit positions of control register 1. Bits 64-95 are used after ASN translation by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS as described in "ASN-Second-Table Entries" on page 3-19.

***Linkage-Table Designation (LTD):*** Bits 96-127 are not used by BRANCH IN SUBSPACE GROUP.

***Access-List Designation (ALD):*** When this ASTE is designated by the primary-ASTE origin in control register 5, bits 128-159 are the primary-space access-list designation (PSALD). During access-register translation when the primary-list bit, bit 7, in the ALET being translated is one, the PSALD is the effective access-list designation. The PSALD is a format-0 ALD or a format-1 ALD, depending on the model.

***ASN-Second-Table-Entry Sequence Number (ASTESN):*** Bits 160-191 are used to control revocation of the accessing capability represented by access-list entries that designate the ASTE. During access-register translation, bits 160-191 are compared against the ASTESN in the access-list entry, and inequality causes an ASTE-sequence exception to be recognized.

When the ASTE is designated by a subspace-ASTE origin (SSASTEO) in a dispatchable-unit control table, bits 160-191 are also used to control revocation of the linkage capability represented by that SSASTEO. When BRANCH IN SUBSPACE GROUP uses ALET 1 to transfer control to the subspace specified by the



LzLabs GmbH. Ex. 1008-222

SSASTEO, or when PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, or LOAD ADDRESS SPACE PARAMETERS uses the SSASTEO to set bits 1-23 and 25-31 of the primary STD in control register 1 or the secondary STD in control register 7 from the same bits of the STD in the subspace ASTE, those instructions first test bit 0 of the subspace ASTE for being zero and recognize an ASTE-validity exception if it is not, and they then compare bits 160-191 to the subspace-ASTE sequence number (SSASTESN) in the dispatchable-unit control table and recognize an ASTE-sequence exception if there is an inequality. However, when either of the two named exception conditions exists for LOAD ADDRESS SPACE PARAMETERS, the instruction sets condition code 1 or 2 instead of recognizing the exception.

Bits 84-85, 128, and 192-223 are reserved for possible future extensions and should be zeros. Bits 224-255 are available for use by programming. The second 32 bytes of the 64-byte ASTE also are reserved for possible future extensions and should contain all zeros.

## Subspace-Replacement Operations

The subspace-group facility includes new operations, called subspace-replacement operations, of PROGRAM CALL, PROGRAM TRANSFER, PROGRAM RETURN, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS. (PROGRAM CALL FAST does not have subspace-replacement operations.) The new operations apply when the dispatchable unit for which any of the five named instructions is executed is in a state called subspace active. A dispatchable unit is subspace active if it has used BRANCH IN SUBSPACE GROUP to transfer control to a subspace of its subspace group and has not subsequently used BRANCH IN SUBSPACE GROUP to return control to the base space of the group.

The definitions of the subspace-replacement operations are included in the definitions of the five named instructions in Chapter 10, "Control Instructions." The operations are described in a general way as follows. Whenever (1) an address space is established as the primary or secondary address space as a result of ASN translation or (2) PROGRAM CALL obtains the origin of the ASN-second-table entry specifying a new primary address space from the entry-table entry used, then, if that address space is in a subspace group, as indicated by the subspace-group-control bit, bit 22 (G), being one in the segment-table designation (STD) for the address space (the new PSTD in control register 1 or SSTD in control register 7), and if the dispatchable unit is subspace-active, as indicated by the subspace-active bit, bit 0 (SA) of word 1, in the dispatchable-unit control table (DUCT) being one, the ASN-second-table-entry (ASTE) origin (ASTEO) for the address space, which was obtained by ASN translation or from the entry-table entry, is compared to the base-ASTE origin (BASTEO), bits 1-25 of word 0, in the DUCT. If that ASTEO and the BASTEO are equal, the following occurs. An ASTE-validity exception is recognized if bit 0 in the ASTE for the last subspace entered by the dispatchable unit, which ASTE is designated by the subspace-ASTE origin (SSASTEO) in the DUCT, is one. An ASTE-sequence exception is recognized if the ASTE-sequence number (ASTESN) in word 5 of the subspace ASTE does not equal the subspace ASTESN (SSASTESN) in word 3 of the DUCT. However, LOAD ADDRESS SPACE PARAMETERS sets a nonzero condition code instead of recognizing the ASTE-validity or ASTE-sequence exception. If no exception exists, bits 1-23 and 25-31 of the STD for the address space (the PSTD in control register 1 or SSTD in control register 7) are replaced by the same bits of the STD in word 2 of the subspace ASTE.

Whenever the address-space-function control, bit 15 of control register 0, is zero, the above additional general definition does not apply, and the definitions of the five instructions are the same as when the subspace-group facility is not installed.

If an addressing exception is recognized when attempting to access the DUCT or subspace ASTE, the instruction execution is suppressed. If an ASTE-validity or ASTE-sequence exception is recognized, the instruction execution is nullified. Such nullification or suppression causes all control register contents to remain unchanged from what they were at the beginning of the instruction execution.

Key-controlled protection does not apply to any accesses to the DUCT or subspace ASTE.


LzLabs GmbH. Ex. 1008-223

For comparing the ASTEO obtained by ASN translation to the BASTEO, either the ASTEO may be compared to the BASTEO or the ASTEO, with one leftmost and six rightmost zeros appended, may be compared to the entire contents of word 0 of the DUCT.

When the SSASTEO in the DUCT is used to access the subspace ASTE, no check is made for whether the SSASTEO is all zeros.

The references to the DUCT and subspace ASTE are single-access references and appear to be word concurrent as observed by other CPUs. The words of the DUCT are accessed in no particular order. The words of the subspace ASTE are accessed in no particular order except that word 0 is accessed first.

The exceptions that can be recognized during a subspace-replacement operation are referred to collectively as the subspace-replacement exceptions and are listed in priority order in "Subspace-Replacement Exceptions" on page 6-45.

## Linkage-Stack Introduction

Many of the functions related to the linkage stack are described in this section and in "Linkage-Stack Operations" on page 5-66. Additionally, tracing of the stacking PROGRAM CALL instruction and of the PROGRAM RETURN instruction is described in Chapter 5, "Program Execution"; interruptions in Chapter 6, "Interruptions"; and the instructions in Chapter 10, "Control Instructions."

## Summary

These major functions are provided:

1. A table-based subroutine-linkage mechanism that provides increased (compared to 370-XA) PSW and control-register status changing and which saves and restores this status and the contents of general registers and access registers through the use of an entry in a linkage stack.
2. A new branch-type linkage mechanism that uses the linkage stack.
3. Instructions for placing an additional two words of status in the current linkage-stack entry and for retrieving all of the status and the general-register and access-register contents that are in the entry.
4. An instruction for determining whether a program is authorized to use a particular access-list-entry token.
5. Aids for program-problem analysis.

In addition, control and authority mechanisms are incorporated to control these functions.

It is intended that a separate linkage stack be associated with and used by each dispatchable unit. The linkage stack for a dispatchable unit resides in the home address space of the dispatchable unit.

It is intended that a dispatchable unit's linkage stack be protected from the dispatchable unit by means of key-controlled protection. Key-controlled protection does not apply to the linkage-stack instructions that place information in or retrieve information from the linkage stack.

The linkage-stack functions are for use by programs considered to be semiprivileged, that is, programs which are executed in the problem state but which are authorized to use additional functions. With these authorization controls, a nonhierarchical organization of programs may be established, with each program in a sequence of calling and called programs having a degree of authority that is arbitrarily different from those of programs before or after it in the sequence. The range of functions available to each program, and the ability to transfer control from one program to another, are prescribed in tables that are managed by the control program.

The linkage-stack instructions, which are semiprivileged, are described in Chapter 10, "Control Instructions." They are:

- BRANCH AND STACK
- EXTRACT STACKED REGISTERS
- EXTRACT STACKED STATE
- MODIFY STACKED STATE
- PROGRAM RETURN
- TEST ACCESS

In addition, the PROGRAM CALL instruction is changed (relative to 370-XA) to optionally form an entry in the linkage stack. A PROGRAM CALL instruction that operates on the linkage stack is called stacking PROGRAM CALL. Recognition of PROGRAM CALL as stacking PROGRAM CALL is



LzLabs GmbH. Ex. 1008-224

under the control of a bit in a 32-byte entry-table entry. The entry-table entry is extended in length from 16 bytes to 32 bytes when the address-space-function (ASF) control, bit 15 of control register 0, is one.

The PROGRAM CALL FAST instruction is available when the program-call-fast facility is installed. PROGRAM CALL FAST has the same operation code as PROGRAM CALL and is a variation of stacking PROGRAM CALL. PROGRAM CALL FAST is not further described in this section. It is described in Chapter 10, "Control Instructions."

## Linkage-Stack Functions

### Transferring Program Control

The use of the linkage stack permits programs operating at arbitrarily different levels of authority to be linked directly without the intervention of the control program. The degree of authority of each program in a sequence of calling and called programs may be arbitrarily different, thus allowing a nonhierarchical organization of programs to be established. Modular authorization control can be obtained principally by associating an extended authorization index with each program module. This allows program modules with different authorities to coexist in the same address space. On the other hand, the extended authorization index in effect during the execution of a called program module can be the one that is associated with the calling program module, thus allowing the called module to be executed with different authorities on behalf of different dispatchable units. Options concerning the PSW-key mask and the secondary ASN are other means of associating different authorities with different programs or with the same called program. The authority of each program is prescribed in tables that are managed by the control program. By setting up the tables so that the same program can be called by means of different PC numbers, the program can be assigned different authorities depending on which PC number is used to call it. The tables also allow control over which PC numbers can be used by a program to call other programs.

The stacking PROGRAM CALL and PROGRAM RETURN linkage operations can link programs residing in different address spaces and having different levels of authority. The execution state and the contents of the general registers and access registers are saved during the execution of stacking PROGRAM CALL and are partially restored during the execution of PROGRAM RETURN. A linkage stack provides an efficient means of saving and restoring both the execution state and the contents of registers during linkage operations. The availability of the linkage stack is controlled by the ASF control in control register 0. When the linkage stack is not available, these two linkage operations cannot be performed.

During the execution of a PROGRAM CALL instruction, the PC-number-translation process is performed to locate a 16-byte or 32-byte entry-table entry, as determined by the ASF control. When a 32-byte entry-table entry is located and a bit, named the PC-type bit, in the entry-table entry is one, the stacking PROGRAM CALL operation is specified; otherwise, the basic PROGRAM CALL operation (the 370-XA operation) is specified.

In addition to the entry information specified in the 16-byte entry-table entry, the 32-byte entry-table entry further contains information that specifies options concerning the address-space control and PSW key in the PSW, and the PSW-key mask, extended authorization index, and secondary ASN in the control registers.

During the stacking PROGRAM CALL operation and by means of the additional information in the entry-table entry, the address-space control in the PSW can be set to specify either the primary-space mode or the access-register mode. The PSW key can be either left unchanged or replaced from the entry-table entry. The PSW-key mask in control register 3 can be either ORed to or replaced from the entry-table entry. The extended authorization index in control register 8 can be either left unchanged or replaced from the entry-table entry. The secondary ASN in control register 3 can be set equal to the primary ASN of either the calling program or the called program; thus, the ability of the called program to have access to the primary address space of the calling program can be controlled.

The stacking PROGRAM CALL operation always forms an entry, called a state entry, in the linkage stack to save the execution state and the contents of general registers 0-15 and access registers 0-15. The saved execution state includes the PC number used, a called-space identification, the updated PSW before any changes are made due


LzLabs GmbH. Ex. 1008-225

to the entry-table entry, and the extended authorization index, PSW-key mask, primary ASN, and secondary ASN existing before the operation. However, the value of the PER mask in the saved updated PSW is unpredictable. The linkage-stack state entry also contains an entry-type code that identifies the entry as one that was formed by PROGRAM CALL.

A space-switching operation occurs when the address-space number (ASN) specified in the entry-table entry is nonzero. When space switching occurs, the operation is called PROGRAM CALL with space switching (PC-ss), and the ASN in the entry-table entry is placed in control register 4 as a new primary ASN. When no space switching occurs, the operation is called PROGRAM CALL to current primary (PC-cp), and there is no change to the primary ASN in control register 4.

PROGRAM CALL with space switching performs ASN translation of the new primary ASN to obtain a new primary-ASTE origin and a new primary segment-table designation, which it places in control registers 5 and 1, respectively. It sets the secondary segment-table designation in control register 7 equal to either the old primary segment-table designation or the new one, depending on whether it set the secondary ASN equal to the old primary ASN or the new one, respectively. PROGRAM CALL to current primary sets the secondary ASN equal to the primary ASN and the secondary segment-table designation equal to the primary segment-table designation.

The instruction PROGRAM RETURN restores most of the information saved in the linkage stack by the stacking PROGRAM CALL operation. It restores the PSW, extended authorization index, PSW-key mask, primary ASN, secondary ASN, and the contents of general registers 2-14 and access-registers 2-14. However, the PER mask in the current PSW remains unchanged, and the resulting condition code is unpredictable. The operation of PROGRAM RETURN is referred to by saying that PROGRAM RETURN unstacks a state entry.

For PROGRAM RETURN, a space-switching operation occurs when the restored primary ASN is not equal to the primary ASN existing before the operation. When space switching occurs, the operation is called PROGRAM RETURN with space switching (PR-ss). When no space switching occurs, the operation is called PROGRAM RETURN to current primary (PR-cp).

PROGRAM RETURN with space switching performs ASN translation of the restored primary ASN to obtain a new primary-ASTE origin and a new primary segment-table designation, which it places in control registers 5 and 1, respectively. For PROGRAM RETURN with space switching or to current primary, (1) if the restored secondary ASN is the same as the restored primary ASN, the secondary segment-table designation in control register 7 is set equal to the new primary segment-table designation in control register 1, or (2) if the the restored secondary ASN is not the same as the restored primary ASN, ASN translation and ASN authorization of the restored secondary ASN are performed to obtain a new secondary segment-table designation, which is placed in control register 7.

The stacking PROGRAM CALL operation and the PROGRAM RETURN operation each can be performed successfully only in the primary-space mode or access-register mode. An exception is recognized when the CPU is in the real mode, secondary-space mode, or home-space mode.

A bit, named the unstack-suppression bit, can be set to one in a linkage-stack state entry to cause an exception if an attempt is made by PROGRAM RETURN to unstack the entry. When the bit is one, the entry still can be operated on by the instructions that add information to or retrieve information from the entry. The unstack-suppression bit is intended to allow the control program to gain control when an attempt is made to unstack a state entry in which the bit is one.

## Branching Using the Linkage Stack

The execution state and the contents of the general registers and access registers can also be saved in the linkage stack by means of the instruction BRANCH AND STACK. BRANCH AND STACK uses a branch address as do the other branching instructions, instead of using a PC number. BRANCH AND STACK, along with PROGRAM RETURN, can link programs residing in the same address space and having the same level of authority; that is, BRANCH AND STACK does not change the execution state except for the instruction address.



LzLabs GmbH. Ex. 1008-226

BRANCH AND STACK forms a linkage-stack state entry that is almost the same as one formed by PROGRAM CALL. When it is necessary to distinguish between these two types of state entry, an entry formed by PROGRAM CALL is called a program-call state entry, and one formed by BRANCH AND STACK is called a branch state entry. A branch state entry differs from a program-call state entry in two ways: (1) it contains a different entry-type code, which identifies it as a branch state entry, and (2) it contains the new value of bits 32-63 of the current PSW, the addressing mode and the branch address, instead of a called-space identification and a PC number. The new value of PSW bits 32-63 is in addition to the complete PSW that is saved in the state entry.

For BRANCH AND STACK, the addressing mode and instruction address that are part of the complete PSW saved in the state entry can be the current addressing mode and the updated instruction address (the address of the next sequential instruction), or they can be specified in a register. This register can be one that had link information placed in it by a BRANCH AND LINK (BALR only), BRANCH AND SAVE, BRANCH AND SAVE AND SET MODE, or BRANCH AND SET MODE instruction. Thus, BRANCH AND STACK can be used either in a calling program or at (or near) the entry point of a called program, and, in either case, a PROGRAM RETURN instruction located at the end of the called program will return correctly to the calling program. The ability to use BRANCH AND STACK at an entry point allows the linkage stack to be used without changing old calling programs.

When the $R_2$ field of BRANCH AND STACK is zero, the instruction is executed without causing branching.

When PROGRAM RETURN unstacks a branch state entry, it ignores the extended authorization index, PSW-key mask, primary ASN, and secondary ASN in the entry. The PROGRAM RETURN instruction restores the PSW and the contents of general registers 2-14 and access registers 2-14 that were saved in the entry. However, the PER mask in the current PSW remains unchanged, and the resulting condition code is unpredictable.

BRANCH AND STACK can be executed successfully only in the primary-space mode or access-register mode. An exception is recognized when the CPU is in the real mode, secondary-space mode, or home-space mode.

The unstack-suppression bit has the same effect in a branch state entry as it does in a program-call state entry.

## Adding and Retrieving Information

The instruction MODIFY STACKED STATE can be used by a program to place two words of information, contained in a designated general-register pair, in an area, called the modifiable area, of the current linkage-stack state entry (a branch state entry or a program-call state entry). This is intended to allow a called program to establish a recovery routine that will be given control by the control program, if necessary.

The instructions EXTRACT STACKED REGISTERS and EXTRACT STACKED STATE can be used by a program to obtain any of the information saved in the current state entry by BRANCH AND STACK or PROGRAM CALL or placed there by MODIFY STACKED STATE. EXTRACT STACKED REGISTERS places the contents of a specified range of general registers and access registers back in the registers from which the contents were saved. EXTRACT STACKED STATE obtains any pair of words of the nonregister information saved or placed in a state entry and places them in a designated general-register pair. EXTRACT STACKED STATE sets the condition code to indicate whether the current state entry is a branch state entry or a program-call state entry.

## Testing Authorization

The instruction TEST ACCESS has as operands an access-list-entry token (ALET) in a designated access register and an extended authorization index (EAX) in a designated general register. TEST ACCESS applies the access-register-translation process, which uses the specified EAX instead of the current EAX in control register 8, to the ALET, and it sets the condition code to indicate the result. The condition code may indicate: (1) the ALET is 00000000 hex, (2) the ALET designates an entry in the dispatchable-unit access list and can be translated without exceptions in access-register translation, (3) the ALET designates an entry in the primary-space access list and can be translated without exceptions in access-register translation, or


LzLabs GmbH. Ex. 1008-227

(4) the ALET is 00000001 hex or causes exceptions in access-register translation.

The principal purpose of TEST ACCESS is to allow a called program to determine whether an ALET passed to it by the calling program is authorized for use by the calling program by means of the calling program's EAX. This is in support of a possible programming convention in which a called program will not operate on an AR-specified address space by means of its own EAX unless the calling program is authorized to operate on that space by means of the calling program's EAX. The called program can obtain the calling program's EAX, for use by TEST ACCESS, from the current linkage-stack state entry by means of the EXTRACT STACKED STATE instruction.

Another purpose of TEST ACCESS is to indicate the special cases in which the ALET is 00000000 hex, designating the primary address space, or 00000001 hex, designating the secondary address space. Because PROGRAM CALL may change the primary and secondary address spaces, ALETs 00000000 hex and 00000001 hex may designate different address spaces when used by the called program than when used by the calling program.

Still another purpose of TEST ACCESS is to indicate whether the ALET designates an entry in the primary-space access list since such a designation after the primary address space was changed by a space-switching program-linkage operation may be an error.

### Program-Problem Analysis

To aid program-problem analysis, the option is provided of having a trace entry made implicitly for three additional linkage operations when the linkage stack is used. When branch tracing is on, a trace entry is made each time a BRANCH AND STACK instruction is executed and causes branching. When ASN tracing is on, a trace entry is made each time the stacking PROGRAM CALL operation is performed and each time PROGRAM RETURN unstacks a linkage-stack state entry formed by PROGRAM CALL or PROGRAM CALL FAST. A detailed definition of tracing is contained in “Tracing” on page 4-10.

As a further analysis aid, BRANCH AND STACK when it causes branching, stacking PROGRAM CALL, and PROGRAM RETURN are also recognized as PER successful-branching events. For PROGRAM RETURN, the unstacked state entry may have been formed by BRANCH AND STACK or PROGRAM CALL.

The execution of a space-switching stacking PROGRAM CALL or PROGRAM RETURN instruction causes a space-switch event if the primary space-switch-event control is one before or after the operation or if a PER event is to be indicated.

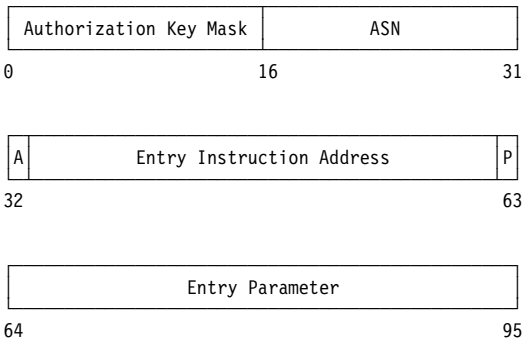
## Extended Entry-Table Entries

When the address-space-function (ASF) control, bit 15 of control register 0, is one, the entry-table entry is extended in length from 16 bytes to 32 bytes. Bit 128 of the 32-byte entry-table entry specifies whether the basic or the stacking PROGRAM CALL operation is to be performed, and bit positions 131-139 and 144-159 contain information that is used only if stacking is specified.

This section describes the use of the 32-byte entry-table entry in both the basic and the stacking PROGRAM CALL operations. The description here of the use in the basic PROGRAM CALL operation is the same as the description in “Entry-Table Entries” on page 5-28.

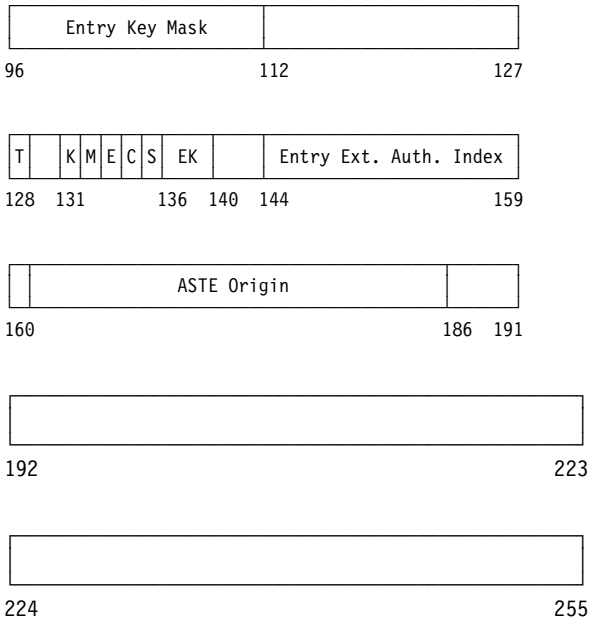
The PROGRAM CALL FAST instruction does not use an entry-table entry; it uses a PCF-entry-table entry. The PCF-entry-table entry is described in the definition of PROGRAM CALL FAST in Chapter 10, “Control Instructions.”

The 32-byte entry-table entry has the following format:






LzLabs GmbH. Ex. 1008-228




The fields in the 32-byte entry-table entry are allocated as follows:

***Authorization Key Mask:*** Bits 0-15 are used to verify whether the program issuing the PROGRAM CALL instruction, when in the problem state, is authorized to call this entry point. The authorization key mask and the current PSW-key mask in control register 3 are ANDed, and the result is checked for all zeros. If the result is all zeros, a privileged-operation exception is recognized. The test is not performed in the supervisor state.

***ASN:*** Bits 16-31 specify whether a PC-ss or PC-cp operation is to occur. When bits 16-31 are all zeros, a PC-cp operation is specified. When bits 16-31 are not all zeros, a PC-ss operation is specified, and the bits are the ASN that replaces the primary ASN.

***Entry Addressing Mode (A):*** Bit 32 replaces the addressing-mode bit, bit 32 of the current PSW, as part of the PROGRAM CALL operation. When bit 32 is zero, bits 33-39 must also be zeros; otherwise, a PC-translation-specification exception is recognized.

***Entry Instruction Address:*** Bits 33-62, with a zero appended on the right, form the instruction address that replaces the instruction address in the PSW as part of the PROGRAM CALL operation.

***Entry Problem State (P):*** Bit 63 replaces the problem-state bit, bit 15 of the current PSW, as part of the PROGRAM CALL operation.

***Entry Parameter:*** Bits 64-95 are placed in general register 4 as part of the PROGRAM CALL operation.

***Entry Key Mask:*** Bits 96-111 are ORed into the PSW-key mask in control register 3 when bit 132, the PSW-key-mask control, is zero, or replace the PSW-key mask in control register 3 when bit 132 is one, as part of the stacking PROGRAM CALL operation. Bits 96-111 are ORed into the PSW-key mask as part of the basic PROGRAM CALL operation.

***PC-Type Bit (T):*** Bit 128, when one, specifies that the PROGRAM CALL instruction is to perform the stacking PROGRAM CALL operation. When this bit is zero, PROGRAM CALL performs the basic PROGRAM CALL operation.

***PSW-Key Control (K):*** Bit 131, when one, specifies that bits 136-139 are to replace the PSW key in the PSW as part of the stacking PROGRAM CALL operation. When this bit is zero, the PSW key remains unchanged. Bit 131 is ignored during the basic PROGRAM CALL operation.

***PSW-Key-Mask Control (M):*** Bit 132, when one, specifies that bits 96-111 are to replace the PSW-key mask in control register 3 as part of the stacking PROGRAM CALL operation. When this bit is zero, bits 96-111 are ORed into the PSW-key mask in control register 3 as part of the stacking PROGRAM CALL operation. Bit 132 is ignored during the basic PROGRAM CALL operation.

***Extended-Authorization-Index Control (E):*** Bit 133, when one, specifies that bits 144-159 are to replace the current extended authorization index in control register 8 as part of the stacking PROGRAM CALL operation. When this bit is zero, the current extended authorization index remains unchanged. Bit 133 is ignored during the basic PROGRAM CALL operation.

***Address-Space-Control Control (C):*** Bit 134, when one, specifies that bit 17 of the current PSW is to be set to one as part of the stacking PROGRAM CALL operation. When this bit is zero, bit 17 is set to zero. Because the CPU must


LzLabs GmbH. Ex. 1008-229

be in either the primary-space mode or the access-register mode when a stacking PROGRAM CALL instruction is issued, the result is that the CPU is placed in the access-register mode if bit 134 is one or the primary-space mode if bit 134 is zero. Bit 134 is ignored during the basic PROGRAM CALL operation.

***Secondary-ASN Control (S):*** Bit 135, when one, specifies that bits 16-31 are to become the new secondary ASN, and the new SSTD is to be set equal to the new PSTD, as part of the stacking PROGRAM CALL with-space-switching (PC-ss) operation. When this bit is zero, the new SASN and SSTD are set equal to the PASN and PSTD, respectively, of the calling program. Bit 135 is ignored during the basic PROGRAM CALL operation and the stacking PROGRAM CALL to-current-primary (PC-cp) operation.

***Entry Key (EK):*** Bits 136-139 replace the PSW key in the PSW as part of the stacking PROGRAM CALL operation if the PSW-key control, bit 131, is one. Bits 136-139 are ignored, and the current PSW key remains unchanged, if bit 131 is zero. Bits 136-139 are ignored during the basic PROGRAM CALL operation.

***Entry Extended Authorization Index:*** Bits 144-159 replace the current extended authorization index, bits 0-15 of control register 8, as part of the stacking PROGRAM CALL operation if the extended-authorization-index control, bit 133, is one. Bits 144-159 are ignored, and the current extended authorization index remains unchanged, if bit 133 is zero. Bits 144-159 are ignored during the basic PROGRAM CALL operation.

***ASTE Origin:*** When bits 16-31 are not all zeros, bits 161-185, with six zeros appended on the right, form the real ASN-second-table-entry (ASTE) address that should result from applying the ASN-translation process to bits 16-31. It is unpredictable whether PC-ss uses bits 161-185 or uses ASN translation to obtain the ASTE address.

Bits 33-39 must be zeros when bit 32 is zero; otherwise, a PC-translation-specification exception is recognized.

Bits 112-127, 129, 130, 140-143, 160, and 186-255 are reserved for possible future extensions and should be zeros.

## Linkage-Stack Operations

A linkage stack may be formed by the control program for each dispatchable unit. The linkage stack is used to save the execution state and the contents of the general registers and access registers during the BRANCH AND STACK, stacking PROGRAM CALL, and PROGRAM CALL FAST operations. The linkage stack is also used to restore a portion of the execution state and general-register and access-register contents during the PROGRAM RETURN operation.

PROGRAM CALL FAST is a variation of stacking PROGRAM CALL and is further referred to in this section only when there is a distinction between it and stacking PROGRAM CALL.

A linkage stack resides in virtual storage. The linkage stack for a dispatchable unit is in the home address space for that dispatchable unit. The home address space is designated by the home segment-table designation in control register 13.

The linkage stack is intended to be protected from problem-state programs so that these programs cannot examine or modify the information saved in the linkage stack, except by means of the EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE instructions. This protection can be obtained by means of key-controlled protection.

A linkage stack may consist of a number of linkage-stack sections chained together. A linkage-stack section is variable in length. The maximum length of each linkage-stack section is 65,560 bytes.

There are three types of entry in the linkage stack: header entry, trailer entry, and state entry. A header entry and a trailer entry are at the beginning and end, respectively, of a linkage-stack section, and they are used to chain linkage-stack sections together. Header entries and trailer entries are formed by the control program. A state entry is used to contain the execution state and register contents that are saved during the BRANCH AND STACK or stacking PROGRAM CALL operation, and it is formed during the operation. A state entry is further distinguished as being a branch state entry if it was formed by



LzLabs GmbH. Ex. 1008-230

BRANCH AND STACK or as being a program-call state entry if it was formed by PROGRAM CALL.

The actions of forming a state entry and saving information in it during the BRANCH AND STACK and stacking PROGRAM CALL operations are called the stacking process. The actions of restoring information from a state entry and logically deleting the entry during the PROGRAM RETURN operation are called the unstacking process. The part of the unstacking process that locates a state entry is also performed during the EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE operations.

Each type of linkage-stack entry has a length that is a multiple of eight bytes. A header entry and trailer entry each has a length of 16 bytes. A state entry has a length of 168 bytes.

Each of the header entry, trailer entry, and state entry has a common eight-byte area at its end, called the entry descriptor. The linkage-stack-entry address in control register 15 designates the leftmost byte of the entry descriptor of the last linkage-stack entry, other than the trailer entry, in a linkage-stack section. This entry is called the current linkage-stack entry, and the section is called the current linkage-stack section.

Each entry descriptor in a linkage-stack section, except the one in the trailer entry of the section, includes a field that specifies the amount of space existing between the end of the entry descriptor and the beginning of the trailer entry. This field is named the remaining-free-space field. The remaining-free-space field in a trailer entry is unused.

When a new state entry is to be formed in the linkage stack during the stacking process, the new entry is placed immediately after the entry descriptor of the current linkage-stack entry, provided that there is enough remaining free space in the current linkage-stack section to contain the new entry. If there is not enough remaining free space in the current section, and if the trailer entry in the current section indicates that another section follows the current section, the new entry is placed immediately after the entry descriptor of the header entry of that following section, provided that there is enough remaining free space in that section. If the trailer entry indicates that there is not a following section, an exception is recognized, and a program interruption occurs. It is then the responsibility of the control program to allocate another section, chain it to the current section, and cause the BRANCH AND STACK or stacking PROGRAM CALL instruction to be reexecuted. If there is a following section but there is not enough remaining free space in it, an exception is recognized.

If the remaining-free-space value that is used to locate a trailer entry is not a multiple of 8, an exception is recognized. The remaining-free-space value in the header entry of a linkage-stack section must be set to a multiple of 8 to ensure that the remaining-free-space value that may be used to locate the trailer entry of the section will be a multiple of 8.

When the stacking process is successful in forming a new state entry, it updates the linkage-stack-entry address in control register 15 so that the address designates the leftmost byte of the entry descriptor of the new entry, which thus becomes the new current linkage-stack entry.

When, during the unstacking process in PROGRAM RETURN, the current linkage-stack entry is a state entry, the process operates on that entry and then updates the linkage-stack-entry address so that it designates the entry descriptor of the preceding entry in the same linkage-stack section. The preceding entry thus becomes the current entry. The new current entry may be another state entry, or it may be a header entry.

The header entry of a linkage-stack section indicates whether there is a preceding section. If there is a preceding section, the header entry contains the address of the last linkage-stack entry, other than the trailer entry, in the preceding section. That last entry should be a state entry (not another header entry), unless there is an error in the linkage stack.

If the unstacking process is performed when the current linkage-stack entry is a header entry, and if the header entry indicates that a preceding linkage-stack section exists, the unstacking process proceeds by treating the entry designated in the preceding section as if it were the current entry, provided that this entry is a state entry. If the header entry does not indicate a preceding section, or if the entry designated in the preceding


LzLabs GmbH. Ex. 1008-231

section is not a state entry, an exception is recognized.

When the unstacking process is performed in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY STACKED STATE, the process locates a state entry but does not change the linkage-stack-entry address in control register 15.

Each entry descriptor in a linkage-stack section includes a field that specifies the length of the next linkage-stack entry, other than the trailer entry, in the section. When a state entry is created during the stacking process, zeros are placed in this field in the created entry, and the length of the state entry is placed in this field in the preceding entry. When a state entry is logically deleted during the unstacking process in PROGRAM RETURN, zeros are placed in this field in the preceding entry. This field is named the next-entry-size field.

When the stacking or unstacking process operates on the linkage stack, key-controlled protection does not apply, but low-address and page protection do apply.

## Linkage-Stack-Operations Control

The use of the linkage stack is controlled by the ASF control, bit 15 of control register 0, the home segment-table designation in control register 13, and the linkage-stack-entry address in control register 15. The home segment-table designation is described in "Dynamic Address Translation" on page 3-26. The ASF control and linkage-stack-entry address are described below.

### Control Register 0

Bit 15 of control register 0 is the address-space-function (ASF) control. When bit 15 is zero, the entry-table entry is 16 bytes, and PROGRAM CALL is necessarily basic PROGRAM CALL. Bit 15 also controls whether the linkage stack is available. The bit must be one for the following instructions to be executed successfully:

- BRANCH AND STACK
- EXTRACT STACKED REGISTERS
- EXTRACT STACKED STATE
- MODIFY STACKED STATE
- PROGRAM CALL FAST
- PROGRAM RETURN
- TEST ACCESS

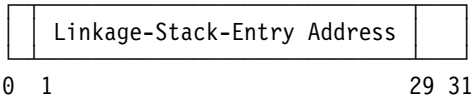
Otherwise, a special-operation exception is recognized.

TEST ACCESS does not use the linkage stack. For TEST ACCESS, the ASF control controls whether the access-list-designation sources are available.

A complete description of the effects of the ASF control is in "Address-Space-Function Control" on page 5-42.

### Control Register 15

The location of the entry descriptor of the current linkage-stack entry is specified in control register 15. The register has the following format:




***Linkage-Stack-Entry Address:*** Bits 1-28 of control register 15, with three zeros appended on the right, form the home virtual address of the entry descriptor of the current linkage-stack entry in the current linkage-stack section. Bits 1-28 are changed during the stacking process in BRANCH AND STACK and stacking PROGRAM CALL and during the unstacking process in PROGRAM RETURN. Bits 0 and 29-31 of control register 15 are set to zeros when bits 1-28 are changed.

## Linkage Stack

The linkage stack consists of one or more linkage-stack sections containing linkage-stack entries. There are three principal types of linkage-stack entry: header entry, trailer entry, and state entry. A state entry is further distinguished as being either a branch state entry or a program-call state entry.

Each type of linkage-stack entry has an entry descriptor at its end. The leftmost byte of the entry descriptor of the current linkage-stack entry in the current linkage-stack section is designated by the linkage-stack-entry address in control register 15.

The linkage stack resides in the home address space, designated by the home segment-table designation in control register 13. The linkage

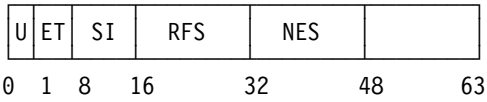


LzLabs GmbH. Ex. 1008-232

stack is available only when the ASF control, bit 15 of control register 0, is one.

## Entry Descriptors

An entry descriptor is at the end of each linkage-stack entry. The entry descriptor is eight bytes in length and has the following format:




The fields in the entry descriptor are allocated as follows:

***Unstack-Suppression Bit (U):*** When bit 0 is one in the entry descriptor of a header entry or state entry encountered during the unstacking process in PROGRAM RETURN, a stack-operation exception is recognized. Bit 0 is ignored in a trailer entry and during the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE. The control program can temporarily set bit 0 to one in the current linkage-stack entry (a header entry or state entry) to prevent PROGRAM RETURN from being executed successfully while still allowing EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE to be executed successfully. Bit 0 is set to zero in the entry descriptor of a state entry when the entry is formed during the stacking process.

***Entry Type (ET):*** Bits 1-7 are a code that specifies the type of the linkage-stack entry containing the entry descriptor. The assigned codes are:

| Code (in Binary) | Entry Type |
|---|---|
| 0000001 | Header entry |
| 0000010 | Trailer entry |
| 0000100 | Branch state entry |
| 0000101 | Program-call state entry |

Codes 0000000, 0000011, and 0000110 through 0111111 binary are reserved for possible future assignments. Codes 1000000 through 1111111 binary are available for use by programming.

Bits 1-7 are set to 0000100 or 0000101 binary in the entry descriptor of a state entry when the entry is formed during the stacking process.

A stack-type exception is recognized during the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN if bits 1-7 in the current linkage-stack entry do not indicate that the entry is a state entry or a header entry; or, when the current entry is a header entry, if bits 1-7 in the entry designated by the backward stack-entry address in the header entry do not indicate that the designated entry is a state entry. However, a stack-specification exception is recognized, instead of a stack-type exception, if both the current entry and the designated entry are header entries.

***Section Identification (SI):*** Bits 8-15 are an identification, provided by the control program, of the linkage-stack section containing the entry descriptor. In the state entry formed by a stacking process, the process sets bits 8-15 equal to the contents of the section-identification field in the preceding linkage-stack entry.

***Remaining Free Space (RFS):*** Bits 16-31 specify the number of bytes between the end of this entry descriptor and the beginning of the trailer entry in the same linkage-stack section, except that this field in a trailer entry has no meaning. Thus, in the last state entry in a section, or in the header entry if there is no state entry, bits 16-31 specify the number of bytes available in the section for performance of the stacking process. In the state entry formed by a stacking process, the process sets bits 16-31 equal to the contents of the remaining-free-space field in the preceding linkage-stack entry minus the size, in bytes, of the new entry. Bits 16-31 must be a multiple of 8 (bits 29-31 must be zeros) in the entry descriptor of the header entry in a linkage-stack section; otherwise, a value that is not a multiple of 8 will be propagated to bits 16-31 in the entry descriptor of each state entry in the section, and a stack-specification exception will be recognized if the stacking process attempts to locate the trailer entry in the section in order to proceed to the next section.

***Next-Entry Size (NES):*** Bits 32-47 specify the size in bytes of the next linkage-stack entry, other than a trailer entry, in the same linkage-stack section. This field in the current linkage-stack entry contains all zeros. This field in a trailer entry has no meaning. When the stacking process forms a state entry, it places zeros in the next-entry-size field of the new entry, and it places the size of the new entry in the next-entry-size field of


LzLabs GmbH. Ex. 1008-233

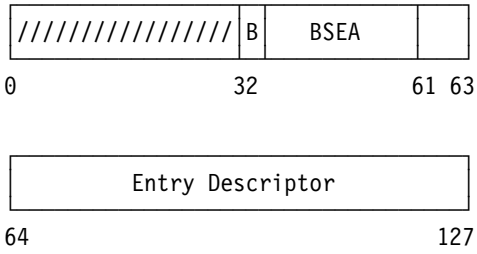
the preceding entry. When the unstacking process logically deletes a state entry, it places zeros in the next-entry-size field of the preceding entry, which entry becomes the current entry.

Bits 48-63 are set to zeros in a state entry when the entry is formed during the stacking process. In a header entry, trailer entry, or state entry, bits 48-63 are reserved for possible future extensions and should always be zeros.

**Programming Note:** No entry-type code will be assigned in which the leftmost bit of the code is one. The control program can temporarily set the leftmost bit to one in the entry-type code of the current linkage-stack entry (a header entry or a state entry) to prevent the successful execution of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN.

## Header Entries

A header entry is at the beginning of each linkage-stack section. The header entry is 16 bytes in length and has the following format:




The fields in the first eight bytes of the header entry are allocated as follows:

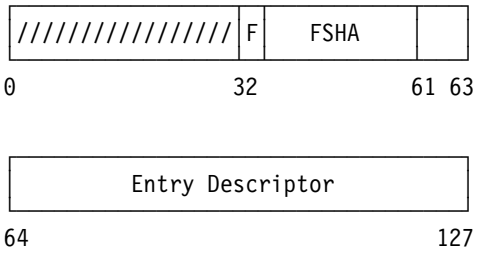
***Backward Stack-Entry Validity Bit (B):*** Bit 32, when one, specifies that the preceding linkage-stack section is available and that the backward stack-entry address, bits 33-60, is valid. Bit 32 is set to one during the stacking process when the process proceeds to this section from the preceding one because there is not enough space available in the preceding section to perform the process. During the unstacking process when this header entry is the current linkage-stack entry, a stack-empty exception is recognized if bit 32 is zero.

***Backward Stack-Entry Address (BSEA):*** When bit 32 is one, bits 33-60, with three zeros appended on the right, form the 31-bit home virtual address of the entry descriptor of the last linkage-stack entry, other than the trailer entry, in the preceding linkage-stack section. However, if the current linkage-stack entry is in the preceding or an earlier linkage-stack section, bits 33-60 may have no meaning because the entry they designate, and earlier entries, may have been logically deleted. Bits 33-60 are set during the stacking process when the process proceeds to this section from the preceding one because there is not enough space available in the preceding section to perform the process. During the unstacking process when this header entry is the current linkage-stack entry and bit 32 is one, the entry designated by bits 33-60 is treated as the current entry.

Bits 61-63 are set to zeros when bits 32-60 are set during the stacking process. Bits 0-31 are available for use by programming. Bits 61-63 are reserved for possible future extensions.

## Trailer Entries

A trailer entry is at the end of each linkage-stack section. The trailer entry begins immediately after the area specified by the remaining-free-space field in the entry descriptors of the header entry and each state entry in the same linkage-stack section. The trailer entry is 16 bytes in length and has the following format:




The fields in the first eight bytes of the trailer entry are allocated as follows:

***Forward-Section Validity Bit (F):*** Bit 32, when one, specifies that the next linkage-stack section is available and that the forward-section-header address, bits 33-60, is valid. During the stacking process when there is not enough space available in the current linkage-stack section to perform the process, a stack-full exception is recognized if bit 32 in the trailer entry of the current section is zero.



LzLabs GmbH. Ex. 1008-234

***Forward-Section-Header Address (FSHA):*** When bit 32 is one, bits 33-60, with three zeros appended on the right, form the 31-bit home virtual address of the entry descriptor of the header entry in the next linkage-stack section. During the stacking process when there is not enough space available in the current section to perform the process and bit 32 is one, the header entry designated by bits 33-60 becomes the current linkage-stack entry.

Bits 0-31 are available for use by programming. Bits 61-63 are reserved for possible future extensions.

**Programming Note:** All of the fields in the trailer entry are set only by the control program.

## State Entries

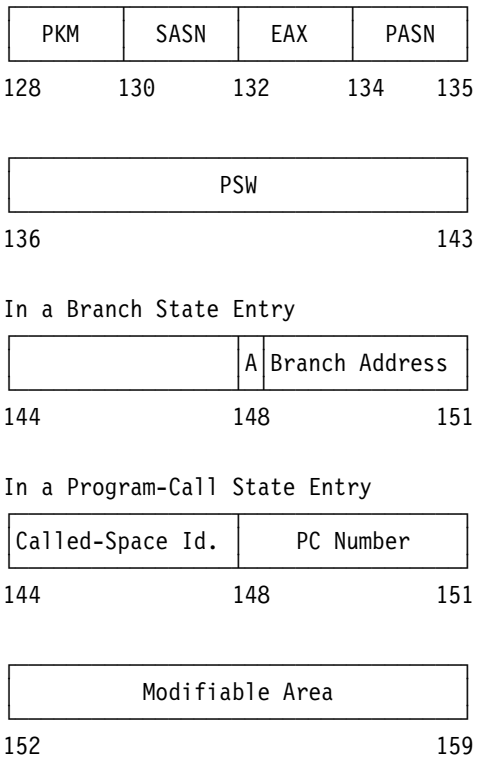
Zero, one, or more state entries may follow the header entry in each linkage-stack section. A state entry may be a branch state entry, formed by a BRANCH AND STACK instruction, or a program-call state entry, formed by a stacking PROGRAM CALL instruction. The state entry is 168 bytes in length and has the following format:

| Hex | Dec | | |
|---|---|---|---|
| 0<br>8<br>/<br>30<br>38 | 0<br>8<br>/<br>48<br>56 | Contents of<br>/ General Registers<br>0-15 | 64 Bytes |
| 40<br>48<br>/<br>70<br>78 | 64<br>72<br>/<br>112<br>120 | Contents of<br>/ Access Registers<br>0-15 | 64 Bytes |
| 80<br>88<br>90<br>98 | 128<br>136<br>144<br>152 | Other Status<br>Information | 32 Bytes |
| A0 | 160 | Entry Descriptor | 8 Bytes |

Bytes 0-63 of the state entry contain the contents of general registers 0-15 in the ascending order of the register numbers. Bytes 64-127 contain the contents of access registers 0-15 in the ascending order of the register numbers. The contents of these fields are moved from the registers to the state entry during the BRANCH AND STACK and stacking PROGRAM CALL operations. The contents of general registers 2-14 and access registers 2-14 are restored from the state entry to the registers during the PROGRAM RETURN operation. The contents of a specified range of general registers and access registers can be restored from the state entry to the registers by EXTRACT STACKED REGISTERS.

Bytes 128-159 of the state entry contain the other status information that is placed in the entry by BRANCH AND STACK, stacking PROGRAM CALL, and MODIFY STACKED STATE. A portion of this status information is restored to the PSW and control registers by PROGRAM RETURN, and all of the information can be examined by means of EXTRACT STACKED STATE. Bytes 160-167 contain the entry descriptor. EXTRACT STACKED STATE sets the condition code to indicate whether the entry-type code in the entry descriptor specifies a branch state entry or a program-call state entry.

Bytes 128-159 of the state entry have the following detailed format:




The fields in bytes 128-159 are allocated as follows. In the following, “of the calling program” means the value existing at the beginning of the execution of the BRANCH AND STACK or


LzLabs GmbH. Ex. 1008-235

stacking PROGRAM CALL instruction that forms the state entry.

***PSW-Key Mask (PKM):*** Bytes 128-129 contain the PSW-key mask, bits 0-15 of control register 3, of the calling program. The PSW-key mask is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL, and it is restored to the control register by a PROGRAM RETURN instruction that unstacks an entry formed by stacking PROGRAM CALL.

***Secondary ASN (SASN):*** Bytes 130-131 contain the secondary ASN, bits 16-31 of control register 3, of the calling program. The SASN is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL, and it is restored to the control register by a PROGRAM RETURN instruction that unstacks an entry formed by stacking PROGRAM CALL.

***Extended Authorization Index (EAX):*** Bytes 132-133 contain the extended authorization index, bits 0-15 of control register 8, of the calling program. The EAX is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL, and it is restored to the control register by a PROGRAM RETURN instruction that unstacks an entry formed by stacking PROGRAM CALL.

***Primary ASN (PASN):*** Bytes 134-135 contain the primary ASN, bits 16-31 of control register 4, of the calling program. The PASN is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL, and it is restored to the control register by a PROGRAM RETURN instruction that unstacks an entry formed by stacking PROGRAM CALL.

***Program-Status Word (PSW):*** In a branch state entry formed by a BRANCH AND STACK instruction in which the $R_1$ field is zero, and in a program-call state entry, bytes 136-143 contain the updated PSW of the calling program. Thus, the addressing-mode bit in this PSW specifies the addressing mode of the calling program, and the instruction address designates the next sequential instruction following the BRANCH AND STACK or stacking PROGRAM CALL instruction that formed the state entry, or following an EXECUTE instruction that had the BRANCH AND STACK or stacking PROGRAM CALL instruction as its target instruction. In a branch state entry formed by a BRANCH AND STACK instruction in which the $R_1$ field is nonzero, bytes 136-143 contain the PSW of the calling program, except that the addressing-mode bit and instruction address in bytes 140-143 are as specified by the contents of the general register designated by the $R_1$ field. See the definition of BRANCH AND STACK in Chapter 10, “Control Instructions” for how the addressing-mode bit and instruction address are specified. The value of the PER mask in bytes 136-143 is always unpredictable. The PSW is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL and is restored as the current PSW by PROGRAM RETURN, except that the PER mask and the condition code, bits 1 and 18-19 of the PSW, are not restored. PROGRAM RETURN does not change the PER mask in the current PSW, and it sets the condition code to an unpredictable value.

***Addressing Mode (A):*** In a branch state entry, bit position 0 of bytes 148-151 contains the addressing-mode bit, bit 32 of the PSW, at the end of the execution of the BRANCH AND STACK instruction that formed the state entry. The addressing-mode bit is saved in bit position 0 of bytes 148-151 by BRANCH AND STACK. BRANCH AND STACK does not change the addressing-mode bit in the PSW.

***Branch Address:*** In a branch state entry, bit positions 1-31 of bytes 148-151 contain the instruction address, bits 33-63 of the PSW, at the end of the execution of the BRANCH AND STACK instruction that formed the state entry. The instruction address is saved in bit positions 1-31 of bytes 148-151 by BRANCH AND STACK. When the $R_2$ field of BRANCH AND STACK is nonzero, the instruction causes branching, and bits 1-31 of bytes 148-151 are the branch address. When the $R_2$ field of BRANCH AND STACK is zero, the instruction is executed without branching, and bits 1-31 of bytes 148-151 designate the next sequential instruction following the BRANCH AND STACK instruction, or following an EXECUTE instruction that had the BRANCH AND STACK instruction as its target instruction.

***Called-Space Identification:*** In a program-call state entry when the called-space-identification facility is installed, bytes 144-147 contain the called-space identification (CSI). The CSI is saved in the state entry by stacking PROGRAM CALL. If the PROGRAM CALL operation was space switching, bytes 0 and 1 of the CSI (bytes



LzLabs GmbH. Ex. 1008-236

144 and 145 of the state entry) contain the new primary ASN that was placed in control register 4 by the PROGRAM CALL instruction, and bytes 2 and 3 of the CSI (bytes 146 and 147 of the state entry) contain the rightmost two bytes of the ASTE sequence number (ASTESN) in the new primary ASTE whose address was placed in control register 5 by the PROGRAM CALL instruction. If the PROGRAM CALL operation was the to-current-primary operation, or if the operation was PROGRAM CALL FAST, the CSI is all zeros. In a program-call state entry when the called-space-identification facility is not installed, or in a branch state entry, the contents of bytes 144-147 are unpredictable.

***PC Number:*** In a program-call state entry, bit positions 12-31 of bytes 148-151 contain the PC number used by the stacking PROGRAM CALL instruction that formed the entry. Stacking PROGRAM CALL places the PC number in bit positions 12-31 of bytes 148-151, and it places zeros in bit positions 0-11.

***Modifiable Area:*** Bytes 152-159 are the field that is set by MODIFY STACKED STATE. BRANCH AND STACK and stacking PROGRAM CALL place all zeros in bytes 152-159.

The contents placed in bytes 144-147 by BRANCH AND STACK and stacking PROGRAM CALL are unpredictable. Bytes 144-147 are reserved for possible future extensions.

## Stacking Process

The stacking process is performed as part of a BRANCH AND STACK or stacking PROGRAM CALL (or PROGRAM CALL FAST) operation. (PROGRAM CALL FAST is referred to only when there is a distinction between it and stacking PROGRAM CALL.) The process locates space for a new linkage-stack state entry, forms the entry, updates the next-entry-size field in the preceding entry, and updates the linkage-stack-entry address in control register 15 so that the new entry becomes the current linkage-stack entry.

For the stacking process to be performed successfully, the address-space-function control, bit 15 of control register 0, must be one, DAT must be on, and the CPU must be in the primary-space mode or access-register mode; otherwise, a special-operation exception is recognized, and the operation is suppressed.

Except as just mentioned, the stacking process is performed independent of the current addressing mode and translation mode, as specified by bits 32, 16, and 17 of the current PSW. All addresses used during the stacking process are always 31-bit home virtual addresses.

During the stacking process when any address is formed through the addition or subtraction of a value to or from another address, a carry out of, or a borrow into, bit position 1 of the address, if any, is ignored.

When the stacking process fetches or stores by using an address that designates, after translation, a location that is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Key-controlled protection does not apply to the accesses made during the stacking process, but page protection and low-address protection do apply. A protection exception causes the operation to be suppressed.

### Locating Space for a New Entry

The linkage-stack-entry address in control register 15 is used to locate the current linkage-stack entry. Bits 1-28 of control register 15, with three zeros appended on the right, form the 31-bit home virtual address of the leftmost byte of the entry descriptor of the current linkage-stack entry.

The first word of the entry descriptor of the current linkage-stack entry is fetched by using the 31-bit home virtual address. This fetch is for the purpose of obtaining the section-identification and remaining-free-space fields in the word; the unstack-suppression bit and entry-type field in the word are not examined.

The 16-bit unsigned binary value in the remaining-free-space field, bits 16-31 of the entry descriptor, is compared against the size in bytes of the linkage-stack entry to be formed. The size of a state entry is 168 bytes. If the value in the field is equal to or greater than the size of the entry to be formed, processing continues as described in “Forming the New Entry” on page 5-74; otherwise, processing continues as described below.


LzLabs GmbH. Ex. 1008-237

When the remaining-free-space field in the current linkage-stack entry indicates that there is not enough space available in the current linkage-stack section to form the new entry, the second word of the trailer entry of the current section is fetched. The address for fetching this word is determined as follows: to the address formed from the contents of control register 15, add 8 to address the first byte after the entry descriptor of the current entry, then add the contents of the remaining-free-space field of the current entry to address the first byte of the trailer entry, and then add 4 to address the second word of the trailer entry. The remaining-free-space value used in the addition must be a multiple of 8; otherwise, a stack-specification exception is recognized, and the operation is nullified.

If the forward-section-validity bit, bit 32, of the trailer entry is zero, a stack-full exception is recognized, and the operation is nullified; otherwise, the forward-section-header address in the trailer entry is used to locate the header entry in the next linkage-stack section. Bits 33-60 of the trailer entry, with three zeros appended on the right, form the 31-bit home virtual address of the leftmost byte of the entry descriptor of the header entry in the next section.

The first word of the entry descriptor of the header entry in the next linkage-stack section is fetched. This fetch is for the purpose of obtaining the section-identification and remaining-free-space fields in the word; the unstack-suppression bit and entry-type field in the word are not examined.

The value in the remaining-free-space field of the header entry in the next linkage-stack section is compared against the size in bytes of the entry to be formed. If the value in the field is equal to or greater than the size of the entry to be formed, the following occurs:

- The linkage-stack-entry address, bits 1-28 of control register 15, is placed, as the backward stack-entry address, in bit positions 33-60 of the header entry in the next linkage-stack section, and zeros are placed in bit positions 61-63.
- The backward stack-entry validity bit, bit 32, in the header entry in the next section is set to one.
- Bits 1-28 of the 31-bit home virtual address of the entry descriptor of the header entry in the next section are placed in bit positions 1-28 of control register 15, and zeros are placed in bit positions 0 and 29-31 of control register 15. Thus, the header entry in the next section becomes the current linkage-stack entry, and the next section becomes the current linkage-stack section.
- Processing continues as described in "Forming the New Entry."

If the value in the remaining-free-space field of the header entry in the next section (before the next section becomes the current section) is less than the size of the linkage-stack entry to be formed, a stack-specification exception is recognized, and the operation is nullified.

## Forming the New Entry

When the remaining-free-space field in the current linkage-stack entry indicates that there is enough space available in the current linkage-stack section to form the new entry, the new entry is formed beginning immediately after the entry descriptor of the current entry.

The new entry is a state entry. The contents of general registers 0-15 are stored in bytes 0-63 of the new entry, in the ascending order of the register numbers. The contents of access registers 0-15 are stored in bytes 64-127 of the new entry, in the ascending order of the register numbers. The PSW-key mask, bits 0-15 of control register 3; secondary ASN, bits 16-31 of control register 3; extended authorization index, bits 0-15 of control register 8; and primary ASN, bits 16-31 of control register 4, are stored in bytes 128-129, 130-131, 132-133, and 134-135, respectively, of the new entry. The current PSW, in which the instruction address has been updated, is stored in bytes 136-143 of the new entry. However, the value of the PER mask, bit 1 in the PSW stored, is unpredictable. Also, if the instruction being executed is a BRANCH AND STACK instruction in which the $R_1$ field is nonzero, the addressing-mode bit and instruction address stored in bytes 140-143 of the new entry are as specified by the contents of the general register designated by the $R_1$ field.

When the called-space-identification facility is installed and the instruction is PROGRAM CALL or PROGRAM CALL FAST, the called-space identification is stored in bytes 144-147 of the new entry. When the instruction is performing the space-switching PROGRAM CALL operation, the



LzLabs GmbH. Ex. 1008-238

called-space identification is the two-byte ASN, bytes 2 and 3, in the entry-table entry used by the instruction, followed by bytes 2 and 3 of the ASTE sequence number, bytes 2 and 3 being bits 176-191, in the ASN-second-table entry specified by the ASN. When the instruction is performing the to-current-primary PROGRAM CALL operation or the space-switching or to-current-primary PROGRAM CALL FAST operation, the called-space identification is all zeros.

When the instruction is BRANCH AND STACK, the addressing-mode bit and instruction address, PSW bits 32-63, existing at the end of the execution of the instruction are stored in bytes 148-151 of the new entry. When the instruction is PROGRAM CALL, the 20-bit PC number used, with 12 zeros appended on the left, is stored in bytes 148-151. Zeros are stored in bytes 152-159 of the new entry.

When the called-space-identification facility is not installed or the instruction is BRANCH AND STACK, the contents of bytes 144-147 of the new entry are unpredictable.

Bytes 160-167 of the new entry are its entry descriptor. The unstack-suppression bit, bit 0, of this entry descriptor is set to zero. The code 0000100 binary is stored in the entry-type field, bits 1-7, of this entry descriptor if the instruction being executed is BRANCH AND STACK. The code 0000101 binary is stored if the instruction is PROGRAM CALL. The value in the section-identification field of the current linkage-stack entry is stored in the section-identification field, bits 8-15, of this entry descriptor. The value in the remaining-free-space field of the current entry, minus the size in bytes of the new entry, is stored in the remaining-free-space field of this entry descriptor. Zeros are stored in the next-entry-size field, bits 32-47, and in bit positions 48-63 of this entry descriptor.

The stores into the new entry appear to be word concurrent as observed by other CPUs. The order in which the stores occur is unpredictable.

### Updating the Current Entry

The size in bytes of the new linkage-stack entry is stored in the next-entry-size field of the current entry. The remainder of the current entry remains unchanged.

The order of the stores into the current entry and the new entry is unpredictable.

### Updating Control Register 15

Bits 1-28 of the 31-bit home virtual address of the entry descriptor of the new linkage-stack entry are placed in bit positions 1-28 of control register 15, the linkage-stack-entry address. Zeros are placed in bit positions 0 and 29-31 of control register 15. Thus, the new entry becomes the current linkage-stack-entry.

### Recognition of Exceptions during the Stacking Process

The exceptions which can be encountered during the stacking process and their priority are described in the definitions of the BRANCH AND STACK, PROGRAM CALL, and PROGRAM CALL FAST instructions.

**Programming Note:** Any exception recognized during the execution of BRANCH AND STACK and PROGRAM CALL (and PROGRAM CALL FAST) causes either nullification or suppression. Therefore, if an exception is recognized, the stacking process does not store into any linkage-stack entry or change the contents of control register 15.

## Unstacking Process

The unstacking process is performed as part of the PROGRAM RETURN operation. The process locates the last state entry in the linkage stack, restores a portion of the information in the entry to the CPU registers, updates the next-entry-size field in the preceding entry, and updates the linkage-stack-entry address in control register 15 so that the preceding entry becomes the current linkage-stack entry. The part of the unstacking process that locates the last state entry is also performed as part of the EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE operations.

For the unstacking process to be performed successfully, the address-space-function control, bit 15 of control register 0, must be one, DAT must


LzLabs GmbH. Ex. 1008-239

be on, and the CPU must be in the primary-space mode or access-register mode; otherwise, a special-operation exception is recognized, and the operation is suppressed. However, when the unstacking process is performed as part of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY STACKED STATE, the CPU may be in the primary-space, access-register, or home-space mode.

Except as just mentioned, the unstacking process is performed independent of the current addressing mode and translation mode, as specified by bits 32, 16, and 17 of the current PSW. All addresses used during the unstacking process are always 31-bit home virtual addresses.

During the unstacking process when any address is formed through the addition or subtraction of a value to or from another address, a carry out of, or a borrow into, bit position 1 of the address, if any, is ignored.

When the unstacking process fetches or stores by using an address that designates, after translation, a location that is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Key-controlled protection does not apply to the accesses made during the unstacking process, but page protection and low-address protection do apply. A protection exception causes the operation to be suppressed.

## Locating the Current Entry and Processing a Header Entry

The linkage-stack-entry address in control register 15 is used to locate the current linkage-stack entry. Bits 1-28 of control register 15, with three zeros appended on the right, form the 31-bit home virtual address of the leftmost byte of the entry descriptor of the current linkage-stack entry.

The first word of the entry descriptor of the current linkage-stack entry is fetched by using the 31-bit home virtual address. If the entry-type code in bits 1-7 of the entry descriptor is not 0000001 binary, indicating that the entry is not a header entry, processing continues as described in "Checking for a State Entry" on page 5-77; otherwise, processing continues as described below.

When the entry-type code in the current linkage-stack entry is 0000001 binary, indicating a header entry, the next processing depends on which instruction is being executed. When the unstacking process is performed as part of the PROGRAM RETURN operation and the unstack-suppression bit, bit 0, in the entry descriptor of the current entry is one, a stack-operation exception is recognized, and the operation is nullified. When the unstacking process is performed as part of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY STACKED STATE, the unstack-suppression bit is ignored.

When there is not an exception due to the unstack-suppression bit, the second word of the current linkage-stack entry (a header entry) is fetched. The address of this word is determined by subtracting 4 from the address of the entry descriptor of the current entry.

If the backward stack-entry validity bit, bit 32, of the current entry is zero, a stack-empty exception is recognized, and the operation is nullified; otherwise, the backward stack-entry address in the current entry is used to locate a linkage-stack entry referred to here as the designated entry. Bits 33-60 of the current entry, with three zeros appended on the right, form the 31-bit home virtual address of the leftmost byte of the entry descriptor of the designated entry.

It is assumed in this definition of the unstacking process that the designated linkage-stack entry is the last entry, other than the trailer entry, in the preceding linkage-stack section. This assumption does not imply any processing that is not explicitly described.

The first word of the entry descriptor of the designated entry is fetched. If the entry-type code in this entry descriptor is not 0000001 binary, indicating that the entry is not a header entry, the following occurs:

- When the unstacking process is performed as part of the PROGRAM RETURN operation, bits 1-28 of the 31-bit home virtual address of the entry descriptor of the designated entry are placed in bit positions 1-28 of control register 15, and zeros are placed in bit positions 0 and 29-31 of control register 15. Thus, the designated entry becomes the current linkage-stack entry, and the preceding section (based



LzLabs GmbH. Ex. 1008-240

on the assumption) becomes the current linkage-stack section. When the unstacking process is performed as part of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY STACKED STATE, the contents of control register 15 remain unchanged, but the designated entry is temporarily, during the remainder of the definition of the instruction, referred to as the current linkage-stack entry.

- Processing continues as described in "Checking for a State Entry."

If the entry-type code in the designated entry is 0000001 binary, indicating a header entry, a stack-specification exception is recognized, and the operation is nullified.

## Checking for a State Entry

When the entry-type code in the current linkage-stack entry indicates that the entry is not a header entry, the code is checked for being 0000100 or 0000101 binary, which are the codes assigned to a state entry.

If the current linkage-stack entry is a state entry, the next processing depends on which instruction is being executed. When the unstacking process is performed as part of the PROGRAM RETURN operation, processing continues as described in "Restoring Information." When the process is performed as part of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY STACKED STATE, the process is completed; that is, no additional processing occurs as a part of the unstacking process.

If the current linkage-stack entry is not a state entry (and necessarily not a header entry either), a stack-type exception is recognized, and the operation is nullified.

## Restoring Information

The remaining parts of the unstacking process occur only in the PROGRAM RETURN operation.

The current linkage-stack entry is a state entry. If the unstack-suppression bit in the entry is one, a stack-operation exception is recognized, and the operation is nullified.

When there is not an exception due to the unstack-suppression bit, a portion of the contents of the current linkage-stack entry are restored to the CPU registers. The contents of general registers 2-14 and access registers 2-14 are restored to those registers from where they were saved in the current entry by the stacking process. When the entry-type code in the current entry is 0000101 binary, indicating a program-call state entry, the PSW-key mask and secondary ASN in control register 3, extended authorization index in control register 8, and primary ASN in control register 4 are similarly restored. During this restoration, the authorization index in control register 4 and the monitor masks in control register 8 remain unchanged. (The authorization index may be changed by the part of the PROGRAM RETURN execution that occurs after the unstacking process.) When the entry-type code is 0000100 binary, indicating a branch state entry, the PSW-key mask, secondary ASN, extended authorization index, and primary ASN in the current entry are ignored, and all contents of the control registers remain unchanged. When the current entry is either a branch state entry or a program-call state entry, the current PSW is restored from bytes 136-143 of the entry, except that the PER mask and the condition code are not restored. The PER mask in the current PSW remains unchanged, and the condition code is set to a unpredictable value. Bytes 144-159 of the current entry are ignored.

The fetches from the current entry appear to be word concurrent as observed by other CPUs. The order in which the fetches occur is unpredictable.

## Updating the Preceding Entry

Zeros are stored in the next-entry-size field, bits 32-47, of the entry descriptor of the preceding linkage-stack entry. The remainder of the preceding entry remains unchanged. The address of the entry descriptor of the preceding entry is determined by subtracting the size in bytes of the current entry from the address of the entry descriptor of the current entry.

The order of the store into the preceding entry and the fetches from the current entry is unpredictable.

## Updating Control Register 15

Bits 1-28 of the 31-bit home virtual address of the entry descriptor of the preceding linkage-stack entry are placed in bit positions 1-28 of control register 15, the linkage-stack-entry address. Zeros are placed in bit positions 0 and 29-31 of control register 15. Thus, the preceding entry becomes the current linkage-stack entry.


LzLabs GmbH. Ex. 1008-241

### Recognition of Exceptions during the Unstacking Process

The exceptions which can be encountered during the unstacking process and their priority are described in the definition of the PROGRAM RETURN instruction. The exceptions which apply to EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE are described in the definitions of those instructions.

**Programming Notes:**

1. Any exceptions recognized during the execution of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN cause either nullification or suppression. Therefore, if an exception is recognized, the unstacking process does not change the contents of any CPU register (except for updating the instruction address in the PSW in the case of suppression) or store into any linkage-stack entry.
2. The unstacking process in PROGRAM RETURN does not restore the PER mask in the PSW so that an act of turning PER on or off after the execution of the related BRANCH AND STACK or PROGRAM CALL instruction but before the execution of the PROGRAM RETURN instruction will not be counteracted. When PROGRAM CALL or PROGRAM RETURN is space switching, the space-switch event can be used as a signal to turn PER on or off, if desired.

## Sequence of Storage References

The following sections describe the effects which can be observed in storage due to overlapped operations and piecemeal execution of a CPU program. Most of the effects described in these sections are observable only when two or more CPUs or channel programs are in simultaneous execution and access common storage locations. Thus, most of the effects need be taken into account by a program only if the program interacts with another CPU or a channel program.

Some of the effects described in the following sections are independent of interaction with another CPU or a channel program. These effects, which are therefore more readily observable, relate to prefetched instructions and overlapping operands of a single instruction. These effects are described in "Conceptual Sequence" and in "Interlocks for Virtual-Storage References" on page 5-79.

### Conceptual Sequence

In the real mode, primary-space mode, or secondary-space mode, the CPU conceptually processes instructions one at a time, with the execution of one instruction preceding the execution of the following instruction. The execution of the instruction designated by a successful branch follows the execution of the branch. Similarly, an interruption takes place between instructions or, for interruptible instructions, between units of operation of such instructions.

The sequence of events implied by the processing just described is sometimes called the conceptual sequence.

Each operation of instruction execution appears to the program itself to be performed sequentially, with the current instruction being fetched after the preceding operation is completed and before the execution of the current operation is begun. This appearance is maintained even though the storage-implementation characteristics and overlap of instruction execution with storage accessing may cause actual processing to be different. The results generated are those that would have been obtained had the operations been performed in the conceptual sequence. Thus, it is possible for an instruction to modify the next succeeding instruction in storage.

Operations in the access-register mode or home-space mode are the same as in the other translation modes, with one exception: an instruction that is a store-type operand of a preceding instruction may appear to be fetched before the store occurs. Thus, it is not assured that an instruction can modify the succeeding instructions. This exception applies if either the storing instruction or the instruction stored is executed in the access-register or home-space mode.

Regardless of the translation mode, there are two other cases in which the copies of prefetched instructions are not necessarily discarded: (1) when the fetch and the store are done by means of different effective addresses that map to the same real address, and (2) when the store is



LzLabs GmbH. Ex. 1008-242

caused by the execution of a vector-facility instruction. The case involving different effective addresses is described in more detail in "Interlocks for Virtual-Storage References" on page 5-79.

### Overlapped Operation of Instruction Execution

In simple models in which operations are not overlapped, the conceptual and actual sequences are essentially the same. However, in more complex machines, overlapped operation, buffering of operands and results, and execution times which are comparable to the propagation delays between units can cause the actual sequence to differ considerably from the conceptual sequence. In these machines, special circuitry is employed to detect dependencies between operations and ensure that the results obtained, as observed by the CPU which generates them, are those that would have been obtained if the operations had been performed in the conceptual sequence. However, other CPUs and channel programs may, unless otherwise constrained, observe a sequence that differs from the conceptual sequence.

### Divisible Instruction Execution

It can normally be assumed that the execution of each instruction occurs as an indivisible event. However, in actual operation, the execution of an instruction consists in a series of discrete steps. Depending on the instruction, operands may be fetched and stored in a piecemeal fashion, and some delay may occur between fetching operands and storing results. As a consequence, intermediate or partially completed results may be observable by other CPUs and by channel programs.

When a program interacts with the operation on another CPU, or with a channel program, the program may have to take into consideration that a single operation may consist in a series of storage references, that a storage reference may in turn consist in a series of accesses, and that the conceptual and observed sequences of these accesses may differ.

Storage references associated with instruction execution are of the following types: instruction fetches, ART-table and DAT-table fetches, and storage-operand references. For the purpose of describing the sequence of storage references, accesses to storage in order to perform ASN translation, PC-number translation, tracing, and the linkage-stack stacking and unstacking processes are considered to be storage-operand references.

**Programming Note:** The sequence of execution of a CPU may differ from the simple conceptual definition in the following ways:

- As observed by the CPU itself, instructions may appear to be prefetched in the access-register or home-space mode regardless of whether the mode exists at the time of the conceptual store or during the execution of the prefetched instruction. They may also appear to be prefetched because of a vector-facility store or when different effective addresses are used. (See "Interlocks for Virtual-Storage References.")
- As observed by other CPUs and by channel programs, the execution of an instruction may appear to be performed as a sequence of piecemeal steps. This is described for each type of storage reference in the following sections.
- As observed by other CPUs and by channel programs, the storage-operand accesses associated with one instruction are not necessarily performed in the conceptual sequence. (See "Relation between Operand Accesses" on page 5-90.)
- As observed by channel programs, in certain unusual situations, the contents of storage may appear to change and then be restored to the original value. (See "Storage Change and Restoration for DAT-Associated Access Exceptions" on page 5-20.)

## Interlocks for Virtual-Storage References

As described in the immediately preceding sections, CPU operation appears, with certain exceptions, to be performed sequentially as observed by the CPU itself; the stores performed by one instruction generally appear to be completed before the next instruction and its operands are fetched. This appearance is maintained in overlapped machines by means of interlock circuitry that detects accesses to a common storage location.

For those instructions which alter the contents of storage and have more than one operand, the


LzLabs GmbH. Ex. 1008-243

instruction definition normally describes the results that are obtained when the operands overlap in storage, this definition being in terms of a sequence of stores and fetches. The interlock circuitry is used in determining whether operand overlap exists.

The purpose of this section is to define those cases in which the machine must appear to operate sequentially, and in which operands of a single instruction must or must not be treated as overlapping.

Proper operation is provided in part by comparing effective addresses. For the purpose of this definition, the term "effective address" means an address before translation, if any, regardless of whether the address is virtual, real, or absolute. If two effective addresses have the same value, the effective addresses are said to be the same even though one may be real or in a different address space.

The values of two virtual effective addresses do not necessarily indicate whether or not the addresses designate the same storage location. The address-translation tables may be set up so that different effective addresses map to the same real address, or so that the same effective address in different address spaces maps to different real addresses.

The interlocks for virtual-storage references are considered in two situations: storage references of one instruction as they affect storage references of another instruction, and multiple storage references of a single instruction.

## Interlocks between Instructions

As observed by the CPU itself, the storage accesses for operands for each instruction appear to occur in the conceptual sequence independent of the effective address used. That is, the operand stores for one instruction appear to be completed before the operand fetches for the next instruction occur. For instruction fetches, the operand stores for one instruction necessarily appear to be completed before the next instruction is fetched only when the same effective address is used for the operand store and the instruction fetch, and then only in the real mode, primary-space mode, or secondary-space mode and when the store is not done by the vector facility.

When an instruction changes the contents of a main-storage location in which a conceptually subsequent instruction is to be executed, either directly or by means of EXECUTE, and when different effective addresses are used to designate that location for storing the result and fetching the instruction, the instruction may appear to be fetched before the store occurs. When either the storing instruction or the subsequent instruction is executed in the access-register mode or home-space mode or when the store is done by the vector facility, changes to the contents of storage are not necessarily recognized even if the effective address used to store the value and the effective address used to fetch the instruction are the same. If an intervening operation causes the prefetched instructions to be discarded, then the updated value is recognized. A definition of when prefetched instructions must be discarded is included in "Instruction Fetching" on page 5-82.

Any change to the storage key appears to be completed before the conceptually following reference to the associated storage block is made, regardless of whether the reference to the storage location is made by means of a virtual, real, or absolute address. Analogously, any conceptually prior references to the storage block appear to be completed when the key for that block is changed or inspected.

## Interlocks within a Single Instruction

For those instructions which alter the contents of storage and have more than one operand, the instruction definition normally describes the results which are obtained when the operands overlap in storage. This result is normally defined in terms of the sequence of the storage accesses; that is, a portion of the results of a store-type operand must appear to be placed in storage before some portion of the other operand is fetched. This definition applies provided that the store and fetch accesses are specified by means of the same effective addresses and the same effective space designations.

When multiple address spaces are involved in the access-register mode, the term "effective space designation" is used to denote the value used by the machine to determine whether two spaces are the same. In the access-register mode, the 32-bit access-list-entry-token (ALET) value associated with each storage-operand address is called the effective space designation. When a B field of



LzLabs GmbH. Ex. 1008-244

zero is specified, a value of all zeros is used for the effective space designation. If the effective space designations are different, the spaces are considered to be different even if both ALETs map to the same segment-table-designation value.

When the store and the fetch accesses are specified by means of different effective space designations or by means of different effective addresses, the operand fetch may appear to precede the operand store.

Figure 5-10 on page 5-82 summarizes the cases of overlap and the specified results, including when MOVE LONG (MVCL) sets condition code 3, for each case.

Effective space designations may be represented by ALB entries, and the test for whether two effective space designations are the same may be performed by comparing ALB entries. If the program changes an attached and valid ART-table entry without subsequently causing the execution of PURGE ALB or a COMPARE AND SWAP AND PURGE instruction that purges the ALB, two effective space designations that are the same may have different representations in the ALB, and failure to recognize operand overlap may result. The use of the ALB never causes overlap to be recognized when the effective space designations are different.

**Programming Note:** A single main-storage location can be accessed by means of more than one address in several ways:

1. The DAT tables may be set up such that multiple addresses in a single address space, or addresses in different address spaces, map to a single real address.
2. The translation of logical, instruction, and virtual addresses may be changed by loading the DAT parameters in the control registers, by changing the address-space-control bits in the PSW, or, for logical and instruction addresses, by turning DAT on or off.
3. In the access-register mode, different address spaces may be selected by means of each access register. In addition, the primary address space is selected for instruction fetching and the target of EXECUTE.
4. STORE USING REAL ADDRESS performs a store by means of a real address.
5. Certain other instructions also use real addresses, and the instructions MOVE TO PRIMARY and MOVE TO SECONDARY access two address spaces.
6. Accesses to storage for the purpose of storing and fetching information for interruptions is performed by means of real addresses, and, for the store-status function, by means of absolute addresses, whereas accesses by the program may be by means of virtual addresses.
7. The real-to-absolute mapping may be changed by means of the SET PREFIX instruction or a reset.
8. A main-storage location may be accessed by channel programs by means of an absolute address and by the CPU by means of a real or a virtual address.
9. A main-storage location may be accessed by another CPU by means of one type of address and by this CPU by means of a different type of address.

The primary purpose of this section on interlocks is to describe the effects caused in cases 1, 3, and 4, above.

For case 2, no effect is observable because prefetched instructions are discarded when the translation parameters are changed, and the delay of stores by a CPU is not observable by the CPU itself.

For case 5, for those instructions which fetch by using real addresses (for example, LOAD REAL ADDRESS, which fetches a segment-table entry and a page-table entry), no effect is observable because only operand accesses between instructions are involved. All instructions that store by using a real address, except STORE USING REAL ADDRESS (or vector-facility instructions executed with DAT off), or that store across address spaces, except in the access-register mode, cause prefetched instructions to be discarded, and no effect is observable.

Cases 6 and 7 are situations which are defined to cause serialization, with the result that prefetched instructions are discarded. In these cases, no effect is observable.

The handling of cases 8 and 9 involves accesses as observed by other CPUs and by channel pro-


LzLabs GmbH. Ex. 1008-245

| Effective Space Designations Equal? | Effective Addresses Overlap Destructively? | Operands Overlap Destructively in Real Storage? | Is Overlap Recognized? | |
|---|---|---|---|---|
| | | | MVCL Sets CC 3 | Operand Results |
| Yes | No | No | No | No |
| Yes | No | Yes | No | Unp. |
| Yes | Yes | No | * | * |
| Yes | Yes | Yes | Yes | Yes |
| No | No | No | No | No |
| No | No | Yes | No | Unp. |
| No | Yes | No | No | No |
| No | Yes | Yes | No | Unp. |

**Explanation:**

* This case cannot occur.
Unp. It is unpredictable whether or not the overlap is recognized.

*Figure 5-10. Virtual-Storage Interlocks within a Single Instruction*

grams and is covered in the following sections in this chapter.

## Instruction Fetching

Instruction fetching consists in fetching the one, two, or three halfwords designated by the instruction address in the current PSW. The immediate field of an instruction is accessed as part of an instruction fetch. If, however, an instruction designates a storage operand at the location occupied by the instruction itself, the location is accessed both as an instruction and as a storage operand. The fetch of the target instruction of EXECUTE is considered to be an instruction fetch.

The bytes of an instruction may be fetched piecemeal and are not necessarily accessed in a left-to-right direction. The instruction may be fetched multiple times for a single execution; for example, it may be fetched for testing the addressability of operands or for inspection of PER events, and it may be refetched for actual execution.

Instructions are not necessarily fetched in the sequence in which they are conceptually executed and are not necessarily fetched each time they are executed. In particular, the fetching of an instruction may precede the storage-operand references for an instruction that is conceptually earlier. The instruction fetch occurs prior to all storage-operand references for all instructions that are conceptually later.

An instruction may be prefetched by using a virtual address only when the associated DAT table entries are attached and valid or when entries which qualify for substitution for the table entries exist in the TLB. An instruction that has been prefetched may be interpreted for execution only for the same virtual address for which the instruction was prefetched.

No limit is established on the number of instructions which may be prefetched, and multiple copies of the contents of a single storage location may be fetched. As a result, the instruction executed is not necessarily the most recently fetched copy. Storing caused by other CPUs and by channel programs does not necessarily change the copy of prefetched instructions. However, if a non-vector-facility store that is conceptually earlier is made by the same CPU using the same effective address as that by which the instruction is subsequently fetched, and the CPU is in any of the real, primary-space, and secondary-space modes when the the storing instruction is executed and is in any of those modes when the subsequent instruction is executed, the updated information is obtained. If the store is caused by a vector-facility instruction, if the effective addresses are different, or if the CPU is in the access-register mode or home-space mode during either the storing execution or the execution of the instruction that is the destination of the store, the updated information is not necessarily obtained. However, the updated information is obtained if either execution is in the real mode since pre-



LzLabs GmbH. Ex. 1008-246

fetched instructions are discarded if DAT is turned on or off.

All copies of prefetched instructions are discarded when:

- A serializing function is performed.
- The CPU enters the operating state.
- DAT is turned on or off.
- A change is made to a translation parameter in control register 1 when in the primary-space, secondary-space, or access-register mode, or in control register 13 when in the home-space mode.

The SET ADDRESS SPACE CONTROL instruction can change the translation mode between any of the primary-space, secondary-space, access-register, and home-space modes, and it performs serialization. The SET ADDRESS SPACE CONTROL FAST instruction can perform the same mode changes, but it does not serialize.

**Programming Notes:**

1. As observed by a CPU itself, its own instruction prefetching may be apparent when storing is done by the vector facility, when different effective addresses map to a single real address, or when the CPU is in the access-register or home-space mode. This is described in "Conceptual Sequence" on page 5-78 and "Interlocks for Virtual-Storage References" on page 5-79.
2. Any means of changing PSW bits 16 and 17, except the SET ADDRESS SPACE CONTROL FAST instruction, causes serialization to be performed and prefetched instructions to be discarded. Turning DAT on or off causes prefetched instructions to be discarded. Therefore, any change of the translation mode, except a change made by SET ADDRESS SPACE CONTROL FAST, always causes prefetched instructions to be discarded.
3. The following are some effects of instruction prefetching on one CPU as observed by other CPUs and by channel programs.

   It is possible for one CPU to prefetch the contents of a storage location, after which another CPU or a channel program can change the contents of that storage location and then set a flag to indicate that the change has been made. Subsequently, the first CPU can test and find the flag set, branch to the modified location, and execute the original prefetched contents.

   It is possible, if another CPU or a channel program concurrently modifies the instruction, for one CPU to recognize the changes to some but not all bit positions of an instruction.

   It is possible for one CPU to prefetch an instruction and subsequently, before the instruction is executed, for another CPU to change the storage key. As a result, the first CPU may appear to execute instructions from a protected storage location. However, the copy of the instructions executed is the copy prefetched before the location was protected.

## ART-Table and DAT-Table Fetches

The access-register-translation (ART) table entries are access-list designations, access-list entries, ASN-second-table entries, and authority-table entries. The dynamic-address-translation (DAT) table entries are segment-table entries and page-table entries. The fetching of these entries may occur as follows:

1. An ART-table entry may be prefetched into the ART-lookaside buffer (ALB) and used from the ALB without refetching from storage, until the entry is cleared by a COMPARE AND SWAP AND PURGE, PURGE ALB, or SET PREFIX instruction or by CPU reset. A DAT-table entry may be prefetched into the translation-lookaside buffer (TLB) and used from the TLB without refetching from storage, until the entry is cleared by a COMPARE AND SWAP AND PURGE, INVALIDATE PAGE TABLE ENTRY, PURGE TLB, or SET PREFIX instruction or by CPU reset. ART-table and DAT-table entries are not necessarily fetched in the sequence conceptually called for; they may be fetched at any time they are attached and valid, including during the execution of conceptually previous instructions.
2. The fetching of access-list designations, access-list entries, ASN-second-table entries, and DAT-table entries appears to be word concurrent as observed by other CPUs. However, the reference to an entry may appear to access a single byte at a time as observed by channel programs.


LzLabs GmbH. Ex. 1008-247

3. The order in which the words of an access-list entry or ASN-second-table entry are fetched is unpredictable, except that the leftmost word of an entry is fetched first. However, the leftmost word of an ASN-second-table entry is not fetched when access-list-entry token 00000000 hex is translated for BRANCH IN SUBSPACE GROUP.
4. An ART-table or DAT-table entry may be fetched even after some operand references for the instruction have already occurred. The fetch may occur as late as just prior to the actual byte access requiring the ART-table or DAT-table entry.
5. An ART-table or DAT-table entry may be fetched for each use of the address, including any trial execution, and for each reference to each byte of each operand.
6. The DAT page-table-entry fetch precedes the reference to the page. When no copy of the page-table entry is in the TLB, the fetch of the associated segment-table entry precedes the fetch of the page-table entry.
7. When no copy of a segment-table entry designated by means of an ART-obtained segment-table designation is in the TLB, the ART fetch of the ASN-second-table entry precedes the DAT segment-table-entry fetch. When no copy of a required authority-table entry is in the ALB, the ART fetch of the associated ASN-second-table entry precedes the fetch of the authority-table entry. When no copy of a required ASN-second-table entry is in the ALB, the fetch of the associated access-list entry precedes the fetch of the ASN-second-table entry. When no copy of a required access-list entry is in the ALB, the fetch of the associated access-list designation precedes the fetch of the access-list entry.

## Storage-Key Accesses

References to the storage key are handled as follows:

1. Whenever a reference to storage is made and key-controlled protection applies to the reference, the four access-control bits and the fetch-protection bit associated with the storage location are inspected concurrently and concurrently with the reference to the storage location.
2. When storing is performed by a CPU, the change bit is set to one in the associated storage key concurrently with the completion of the store access, as observed by the CPU itself. When storing is performed by a CPU or a channel program, the change bit is set to one in the associated storage key either before or after the completion of the store access, as observed by other (if the store was performed by a CPU) or all (if the store was performed by a channel program) CPUs. As observed by other or all CPUs, the change bit is set no earlier than (1) after the last serialization function performed previously by the CPU or channel program performing the store, and (2) after the execution, by any CPU in the configuration, of the SET STORAGE KEY EXTENDED instruction that last set the associated storage key before the completion of the store. As observed by other or all CPUs, a change-bit setting necessarily occurs only when any of the following occurs subsequent to the storing operation:
   - The CPU or channel program that performed the store performs a serialization function.
   - The store was performed by a CPU or a channel program, and any CPU in the configuration sets the subject change bit by executing SET STORAGE KEY EXTENDED after the store access is completed. The change-bit setting due to the store access occurs before the setting by SET STORAGE KEY EXTENDED.
   - The store was performed by a CPU and is or will be completed, and any CPU in the configuration executes a COMPARE AND SWAP AND PURGE, INVALIDATE DAT TABLE ENTRY, or INVALIDATE PAGE TABLE ENTRY instruction that clears from the ALB or TLB of the storing CPU any entry used to complete the store. Completion of the clearing instruction is delayed until the subject store and change-bit setting have been completed.
   - The store was performed by a CPU, and that CPU examines the subject change bit by means of an INSERT STORAGE KEY EXTENDED or RESET REFERENCE BIT EXTENDED instruction. See “Relation between Storage-Key Accesses” on page 5-90.



LzLabs GmbH. Ex. 1008-248

3. The instruction SET STORAGE KEY EXTENDED causes all seven bits to be set concurrently in the storage key. The access to the storage key for SET STORAGE KEY EXTENDED follows the sequence rules for storage-operand store references and is a single-access reference.
4. The INSERT STORAGE KEY EXTENDED instruction provides a consistent image of bits 0-6 of the storage key. Similarly, the instructions INSERT VIRTUAL STORAGE KEY and TEST PROTECTION provide a consistent image of bits 0-4 of the storage key. The access to the storage key for all of these instructions follows the sequence rules for storage-operand fetch references and is a single-access reference.
5. The instruction RESET REFERENCE BIT EXTENDED modifies only the reference bit. All other bits of the storage key remain unchanged. The reference bit and change bit are examined concurrently to set the condition code. The fetch and store accesses to the storage key for RESET REFERENCE BIT EXTENDED follow the sequence rules for storage-operand fetch and store references, respectively, and are single-access references.

The record of references provided by the reference bit is not necessarily accurate. However, in the majority of situations, reference recording approximately coincides with the related storage reference.

The change bit may be set in cases when no storing has occurred. See "Exceptions to Nullification and Suppression" on page 5-19.

## Storage-Operand References

A storage-operand reference is the fetching or storing of the explicit operand or operands in the storage locations designated by the instruction.

During the execution of an instruction, all or some of the storage operands for that instruction may be fetched, intermediate results may be maintained for subsequent modification, and final results may be temporarily held prior to placing them in storage. Stores caused by other CPUs and by channel programs do not necessarily affect these intermediate results.

Storage-operand references are of three types: fetches, stores, and updates.

### Storage-Operand Fetch References

When the bytes of a storage operand participate in the instruction execution only as a source, the operand is called a fetch-type operand, and the reference to the location is called a storage-operand fetch reference. A fetch-type operand is identified in individual instruction definitions by indicating that the access exception is for fetch.

All bits within a single byte of a fetch-type operand are accessed concurrently. When an operand consists of more than one byte, the bytes may be fetched from storage piecemeal, one byte at a time. Unless otherwise specified, the bytes are not necessarily fetched in any particular sequence.

The storage-operand fetch references of one instruction occur after those of all preceding instructions and before those of subsequent instructions, as observed by other CPUs and by channel programs. The operands of any one instruction are fetched in the sequence specified for that instruction. The CPU may fetch the operands of instructions before the instructions are executed. There is no defined limit on the length of time between when an operand is fetched and when it is used. Still, as observed by the CPU itself, its storage-operand references are performed in the conceptual sequence.

### Storage-Operand Store References

When the bytes of a storage operand participate in the instruction execution only as a destination, to the extent of being replaced by the result, the operand is called a store-type operand, and the reference to the location is called a storage-operand store reference. A store-type operand is identified in individual instruction definitions by indicating that the access exception is for store.

All bits within a single byte of a store-type operand are accessed concurrently. When an operand consists of more than one byte, the bytes may be placed in storage piecemeal, one byte at a time. Unless otherwise specified, the bytes are not necessarily stored in any particular sequence.

The CPU may delay placing results in storage. There is no defined limit on the length of time that results may remain pending before they are


LzLabs GmbH. Ex. 1008-249

stored. This delay does not affect the sequence in which results are placed in storage.

The results of one instruction are placed in storage after the results of all preceding instructions have been placed in storage and before any results of the succeeding instructions are stored, as observed by other CPUs and by channel programs. The results of any one instruction are stored in the sequence specified for that instruction.

The CPU does not fetch operands, ART-table entries, or DAT-table entries from a storage location until all information destined for that location by the CPU has been stored. Prefetched instructions may appear to be updated before the information appears in storage.

The stores are necessarily completed only as a result of a serializing operation and before the CPU enters the stopped state.

## Storage-Operand Update References

In some instructions, the storage-operand location participates both as a source and as a destination. In these cases, the reference to the location consists first in a fetch and subsequently in a store. The operand is called an update-type operand, and the combination of the two accesses is referred to as an update reference. Instructions such as MOVE ZONES, TRANSLATE, OR (OC, OI), and ADD DECIMAL cause an update to the first-operand location. An update-type operand is identified in the individual instruction definition by indicating that the access exception is for both fetch and store.

For most instructions which have update-type operands, the fetch and store accesses associated with an update reference do not necessarily occur one immediately after the other, and it is possible for other CPUs and channel programs to make fetch and store accesses to the same location during this time. Such an update reference is sometimes called a noninterlocked-update storage reference.

For certain special instructions, the update reference is interlocked against certain accesses by other CPUs. Such an update reference is called an interlocked-update reference. The fetch and store accesses associated with an interlocked-update reference do not necessarily occur one immediately after the other, but all store accesses by other CPUs and the fetch and store accesses associated with interlocked-update references by other CPUs are prevented from occurring at the same location between the fetch and the store accesses of an interlocked-update reference. Accesses by channel programs may occur to the location during the interlock period.

The storage-operand update reference for the following instructions appears to be an interlocked-update reference as observed by other CPUs. The instructions TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP perform an interlocked-update reference. On models in which the STORE CHARACTERS UNDER MASK instruction with a mask of zero fetches and stores the byte designated by the second-operand address, the fetch and store accesses are an interlocked-update reference.

Within the limitations of the above requirements, the fetch and store accesses associated with an update reference follow the same rules as the fetches and stores described in the previous sections.

**Programming Notes:**

1. When two CPUs attempt to update information at a common main-storage location by means of a noninterlocked-update reference, it is possible for both CPUs to fetch the information and subsequently make the store access. The change made by the first CPU to store the result in such a case is lost. Similarly, if one CPU updates the contents of a field by means of a noninterlocked-update reference, but another CPU makes a store access to that field between the fetch and store parts of the update reference, the effect of the store is lost. If, instead of a store access, a CPU makes an interlocked-update reference to the common storage field between the fetch and store portions of a noninterlocked-update reference due to another CPU, any change in the contents produced by the interlocked-update reference is lost.
2. The instructions TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP facilitate updating of a common storage field by two or more CPUs. To ensure that no changes are lost, all CPUs must use an instruction providing an



LzLabs GmbH. Ex. 1008-250

interlocked-update reference. In addition, the program must ensure that channel programs do not store into the same storage location since such stores may occur between the fetch and store portions of an interlocked-update reference.

3. Only those bytes which are included in the result field of both operations are considered to be part of the common main-storage location. However, all bits within a common byte are considered to be common even if the bits modified by the two operations do not overlap. As an example, if (1) one CPU executes the instruction OR (OC) with a length of 1 and the value 80 hex in the second-operand location, (2) the other CPU executes AND (NC) with a length of 1 and the value FE hex in the second-operand location, and (3) the first operand of both instructions is the same byte, then the result of one of the updates can be lost.

4. When the store access is part of an update reference by the CPU, the execution of the storing is not necessarily contingent on whether the information to be stored is different from the original contents of the location. In particular, the contents of all designated byte locations are replaced, and, for each byte in the field, the entire contents of the byte are replaced.

   Depending on the model, an access to store information may be performed, for example, in the following cases:

   a. Execution of the OR instruction (OI or OC) with a second operand of all zeros.

   b. Execution of OR (OC) with the first-and second-operand fields coinciding.

   c. For those locations of the first operand of TRANSLATE where the argument and function values are the same.

# Storage-Operand Consistency

## Single-Access References

A fetch reference is said to be a single-access reference if the value is fetched in a single access to each byte of the data field. In the case of overlapping operands, the location may be accessed once for each operand. A store-type reference is said to be a single-access reference if a single store access occurs to each byte location within the data field. An update reference is said to be single access if both the fetch and store accesses are each single access.

Except for the accesses associated with multiple-access references and the stores associated with storage change and restoration for DAT-associated access exceptions, all storage-operand references are single-access references.

## Multiple-Access References

In some cases, multiple accesses may be made to all or some of the bytes of a storage operand. The following cases may involve multiple-access references:

1. The storage operands of the following instructions:

   - CHECKSUM
   - CIPHER MESSAGE
   - CIPHER MESSAGE WITH CHAINING
   - COMPARE AND FORM CODEWORD
   - COMPARE UNTIL SUBSTRING EQUAL
   - COMPUTE INTERMEDIATE MESSAGE DIGEST
   - COMPUTE LAST MESSAGE DIGEST
   - COMPUTE MESSAGE AUTHENTICATION CODE
   - CONVERT TO BINARY
   - CONVERT TO DECIMAL
   - CONVERT UNICODE TO UTF-8
   - CONVERT UTF-8 TO UNICODE
   - LOAD ADDRESS SPACE PARAMETERS
   - LOAD REVERSED
   - MOVE INVERSE
   - MOVE PAGE
   - MOVE WITH OFFSET
   - PACK
   - PACK ASCII
   - PACK UNICODE
   - RESUME PROGRAM
   - STORE REVERSED
   - STORE SYSTEM INFORMATION
   - TEST BLOCK
   - TRANSLATE
   - TRANSLATE EXTENDED
   - TRANSLATE ONE TO ONE
   - TRANSLATE ONE TO TWO
   - TRANSLATE TWO TO ONE
   - TRANSLATE TWO TO TWO
   - UNPACK
   - UNPACK ASCII
   - UNPACK UNICODE


LzLabs GmbH. Ex. 1008-251

- UPDATE TREE

2. The stores into that portion of the first operand of MOVE LONG, MOVE LONG EXTENDED, or MOVE LONG UNICODE which is filled with padding bytes.
3. The storage operands of the decimal instructions.
4. The main-storage operands of PAGE IN and PAGE OUT.
5. The storage operands of the I/O instructions.
6. The stores into a trace entry.
7. The storage operands of vector-facility instructions.
8. The stores associated with the stop-and-store-status, store-status-at-address, and store-extended-status-at-address SIGNAL PROCESSOR orders.
9. The trap control block and trap save area used by TRAP.

When a storage-operand store reference to a location is not a single-access reference, the value placed at a byte location is not necessarily the same for each store access; thus, intermediate results in a single-byte location may be observed by other CPUs and by channel programs.

**Programming Notes:**

1. When multiple fetch or store accesses are made to a single byte that is being changed by another CPU or by a channel program, the result is not necessarily limited to that which could be obtained by fetching or storing the bits individually. For example, the execution of MULTIPLY DECIMAL may consist in repetitive additions and subtractions, each of which causes the second operand to be fetched from storage and the first operand to be updated in storage.
2. When CPU instructions which make multiple-access references are used to modify storage locations being simultaneously accessed by another CPU or by a channel program, multiple store accesses to a single byte by the CPU may result in intermediate values being observed by the other CPU or by the channel program. To avoid these intermediate values (for example, when modifying a CCW chain), only instructions making single-access references should be used.
3. An instruction fetch, including the fetch of the target of EXECUTE, is a multiple-access reference.

## Block-Concurrent References

For some references, the accesses to all bytes within a halfword, word, or doubleword are specified to appear to be block concurrent as observed by other CPUs. These accesses do not necessarily appear to channel programs to include more than a byte at a time. The halfword, word, or doubleword is referred to in this section as a block. When a fetch-type reference is specified to appear to be concurrent within a block, no store access to the block by another CPU is permitted during the time that bytes contained in the block are being fetched. Accesses to the bytes within the block by channel programs may occur between the fetches. When a store-type reference is specified to appear to be concurrent within a block, no access to the block, either fetch or store, is permitted by another CPU during the time that the bytes within the block are being stored. Accesses to the bytes in the block by channel programs may occur between the stores.

## Consistency Specification

For all instructions in the S, RX, or RXE format, with the exception of CONVERT TO BINARY, CONVERT TO DECIMAL, LOAD REVERSED, RESUME PROGRAM, STORE CLOCK EXTENDED, STORE REVERSED, STORE SYSTEM INFORMATION, TRAP, and the I/O instructions, when the operand is addressed on a boundary which is integral to the size of the operand, the storage-operand references appear to be block concurrent as observed by other CPUs.

For the instructions COMPARE AND SWAP, COMPARE AND SWAP AND PURGE, and COMPARE DOUBLE AND SWAP, all accesses to the storage operand appear to be block concurrent as observed by other CPUs.

For the instruction PERFORM LOCKED OPERATION, The accesses to the even-numbered storage operands appear to be word concurrent, as observed by other CPUs, for function codes that are a multiple of 4 and appear to be doubleword concurrent, as observed by other CPUs, for function codes that are one more than a multiple of 4. The accesses to the doublewords in



LzLabs GmbH. Ex. 1008-252

the parameter list appear to be doubleword concurrent, as observed by other CPUs, regardless of the function code.

The instructions LOAD MULTIPLE and STORE MULTIPLE, when the operand starts on a word boundary, and the instructions COMPARE LOGICAL (CLC), COMPARE LOGICAL CHARACTERS UNDER MASK, INSERT CHARACTERS UNDER MASK, and STORE CHARACTERS UNDER MASK access their storage operands in a left-to-right direction, and all bytes accessed within each doubleword appear to be accessed concurrently as observed by other CPUs.

The instructions LOAD ACCESS MULTIPLE, LOAD CONTROL, STORE ACCESS MULTIPLE, and STORE CONTROL access the storage operand in a left-to-right direction, and all bytes accessed within each word appear to be accessed concurrently as observed by other CPUs.

When destructive overlap does not exist, the operands of MOVE (MVC), MOVE WITH KEY, MOVE TO PRIMARY, and MOVE TO SECONDARY are accessed as follows:

1. The first operand is accessed in a left-to-right direction, and all bytes accessed within a doubleword appear to be accessed concurrently as observed by other CPUs.
2. The second operand is accessed in a left-to-right direction, and all bytes within a doubleword in the second operand that are moved into a single doubleword in the first operand appear to be fetched concurrently as observed by other CPUs. Thus, if the first and second operands begin on the same byte offset within a doubleword, the fetch of the second operand appears to be doubleword concurrent as observed by other CPUs. If the offsets within a doubleword differ by 4, the fetch of the second operand appears to be word concurrent as observed by other CPUs.

Destructive overlap is said to exist when the result location is used as a source after the result has been stored, assuming processing to be performed one byte at a time.

The operands of MOVE WITH SOURCE KEY, MOVE WITH DESTINATION KEY, and MOVE STRING are accessed the same as those of MOVE (MVC), except that destructive overlap is assumed not to exist.

Except as noted in the individual instruction descriptions, accesses to operands of MOVE LONG, MOVE LONG EXTENDED, and MOVE LONG UNICODE do not necessarily appear to occur in a left-to-right direction as observed by other CPUs and by channel programs. The operands of these instructions do appear to be accessed doubleword concurrent, as observed by other CPUs, when all of the following are true:

- Both operands start on doubleword boundaries and are an integral number of doublewords in length.
- The operands do not overlap.
- The nonpadding part of the operation is being executed.

The operands of COMPARE LOGICAL LONG, COMPARE LOGICAL LONG EXTENDED, and COMPARE LOGICAL LONG UNICODE appear to be accessed doubleword concurrent, as observed by other CPUs, when both operands start on doubleword boundaries and are an integral number of doublewords in length.

The operands of COMPARE LOGICAL STRING appear to be accessed doubleword concurrent, as observed by other CPUs, when both operands start on doubleword boundaries. The operand of SEARCH STRING appears to be accessed doubleword concurrent, as observed by other CPUs, when it starts on a doubleword boundary.

For EXCLUSIVE OR (XC), the operands are processed in a left-to-right direction, and, when the first and second operands coincide, all bytes accessed within a doubleword appear to be accessed concurrently as observed by other CPUs.

**Programming Note:** In the case of EXCLUSIVE OR (XC) designating operands which coincide exactly, the bytes within the field may appear to be accessed as many as three times, by two fetches and one store: once as the fetch portion of the first operand update, once as the second-operand fetch, and then once as the store portion of the first-operand update. Each of the three accesses appears to be doubleword concurrent as observed by other CPUs, but the three accesses do not necessarily appear to occur one immediately after the other. One or both fetch accesses may be omitted since the instruction can be completed without fetching the operands.


LzLabs GmbH. Ex. 1008-253

## Relation between Operand Accesses

As observed by other CPUs and by channel programs, storage-operand fetches associated with one instruction execution appear to precede all storage-operand references for conceptually subsequent instructions. A storage-operand store specified by one instruction appears to precede all storage-operand stores specified by conceptually subsequent instructions, but it does not necessarily precede storage-operand fetches specified by conceptually subsequent instructions. However, a storage-operand store appears to precede a conceptually subsequent storage-operand fetch from the same main-storage location.

When an instruction has two storage operands both of which cause fetch references, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. When the two operands overlap, the common locations may be fetched independently for each operand.

When an instruction has two storage operands the first of which causes a store and the second a fetch reference, it is unpredictable how much of the second operand is fetched before the results are stored. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

When an instruction has two storage operands the first of which causes an update reference and the second a fetch reference, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. Similarly, it is unpredictable how much of the result is processed before it is returned to storage. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

The independent fetching of a single location for each of two operands may affect the program execution in the following situation. When the same storage location is designated by two operand addresses of an instruction, and another CPU or a channel program causes the contents of the location to change during execution of the instruction, the old and new values of the location may be used simultaneously. For example, comparison of a field to itself may yield a result other than equal, or EXCLUSIVE-ORing of a field with itself may yield a result other than zero.

## Other Storage References

The restart, program, supervisor-call, external, input/output, and machine-check PSWs appear to be accessed doubleword concurrent as observed by other CPUs. These references appear to occur after the conceptually previous unit of operation and before the conceptually subsequent unit of operation. The relationship between the new-PSW fetch, the old-PSW store, and the interruption-code store is unpredictable.

Store accesses for interruption codes are not necessarily single-access stores. The store accesses for the external and supervisor-call-interruption codes appear to occur between the conceptually previous and conceptually subsequent operations. The store accesses for the program-interruption codes may precede the storage-operand references associated with the instruction which results in the program interruption.

## Relation between Storage-Key Accesses

As observed by other CPUs, storage-key fetches and stores due to instructions that explicitly manipulate a storage key (INSERT STORAGE KEY EXTENDED, INSERT VIRTUAL STORAGE KEY, RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED) are ordered among themselves and among storage-operand references as if the storage-key accesses were themselves storage-operand fetches and stores, respectively.

Accesses of the access-control and fetch-protection bits due to storage-operand references are concurrent with the references. Accesses of the reference and change bits due to storage-operand references are in no particular order within the interval in which they are defined to occur. (See the description of when the change bit is set in “Storage-Key Accesses” on page 5-84.) However, whether due to an instruction that explicitly manipulates a storage key or due to a storage-operand reference, a storage-key



LzLabs GmbH. Ex. 1008-254

store appears to precede a conceptually subsequent storage-key fetch from the same storage key.

## Serialization

The sequence of functions performed by a CPU is normally independent of the functions performed by other CPUs and by channel programs. Similarly, the sequence of functions performed by a channel program is normally independent of the functions performed by other channel programs and by CPUs. However, at certain points in its execution, serialization of the CPU occurs. Serialization also occurs at certain points for channel programs.

## CPU Serialization

All interruptions and the execution of certain instructions cause a serialization of CPU operations. A serialization operation consists in completing all conceptually previous storage accesses and related reference-bit and change-bit settings by the CPU, as observed by other CPUs and by channel programs, before the conceptually subsequent storage accesses and related reference-bit and change-bit settings occur. Serialization affects the sequence of all CPU accesses to storage and to the storage keys, except for those associated with ART-table-entry and DAT-table-entry fetching.

Serialization is performed by CPU reset, all interruptions, and by the execution of the following instructions:

- The general instructions BRANCH ON CONDITION (BCR) with the $M_1$ and $R_2$ field containing all ones and all zeros, respectively, and COMPARE AND SWAP, COMPARE DOUBLE AND SWAP, STORE CLOCK, STORE CLOCK EXTENDED, SUPERVISOR CALL, and TEST AND SET.
- LOAD PSW and SET STORAGE KEY EXTENDED.
- All I/O instructions.
- PURGE ALB, PURGE TLB, and SET PREFIX. PURGE ALB and SET PREFIX also cause the ALB to be cleared of all entries. PURGE TLB and SET PREFIX also cause the TLB to be cleared of all entries.
- SIGNAL PROCESSOR.
- INVALIDATE PAGE TABLE ENTRY.
- TEST BLOCK.
- MOVE TO PRIMARY, MOVE TO SECONDARY, PROGRAM CALL, PROGRAM CALL FAST, PROGRAM TRANSFER, SET ADDRESS SPACE CONTROL, and SET SECONDARY ASN.
- PROGRAM RETURN when the state entry to be unstacked is a program-call state entry.
- PERFORM LOCKED OPERATION. Serialization is performed immediately after the lock is obtained and again immediately before it is released. However, values fetched from the parameter list before the lock is obtained are not necessarily refetched.
- The three trace functions — branch tracing, ASN tracing, and explicit tracing — cause serialization to be performed before the trace action and after completion of the trace action.
- PAGE IN and PAGE OUT.
- COMPARE AND SWAP AND PURGE, which can also cause the ALB and the TLB to be cleared of all entries on all CPUs

The sequence of events associated with a serializing operation is as follows:

1. All conceptually previous storage accesses by the CPU are completed as observed by other CPUs and by channel programs. This includes all conceptually previous stores and changes to the storage keys.
2. The normal function associated with the serializing operation is performed. In the case of instruction execution, operands are fetched, and the storing of results is completed. The exceptions are LOAD PSW and SET PREFIX, in which the operand may be fetched before previous stores have been completed, and interruptions, in which the interruption code and associated fields may be stored prior to the serialization. The fetching of the serializing instruction occurs before the execution of the instruction and may precede the execution of previous instructions, but may not precede the completion of any previous serializing operation. In the case of an interruption, the old PSW, the interruption code, and other information, if any, are stored, and the new


LzLabs GmbH. Ex. 1008-255

PSW is fetched, but not necessarily in that sequence.

3. Finally, instruction fetch and operand accesses for conceptually subsequent operations may begin.

A serializing function affects the sequence of storage accesses that are under the control of the CPU in which the serializing function takes place. It does not affect the sequence of storage accesses under the control of other CPUs and of channel programs.

**Programming Notes:**

1. The following are some effects of a serializing operation:
   a. When the execution of an instruction changes the contents of a storage location that is used as a source of a following instruction and when different addresses are used to designate the same absolute location for storing the result and fetching the instruction, a serializing operation following the change ensures that the modified instruction is executed.
   b. When a serializing operation takes place, other CPUs and channel programs observe instruction and operand fetching and result storing to take place in the sequence established by the serializing operation.
2. Storing into a location from which a serializing instruction is fetched does not necessarily affect the execution of the serializing instruction unless a serializing function has been performed after the storing and before the execution of the serializing instruction.
3. Following is an example showing the effects of serialization. Location A initially contains X'FF'.

```
        CPU 1               CPU 2

MVI  A,X'00'      G CLI  A,X'00'
BCR  15,0           BNE  G
```

   The BCR 15,0 instruction executed by CPU 1 is a serializing instruction that ensures that the store by CPU 1 at location A is completed. However, CPU 2 may loop indefinitely, or until the next I/O or external interruption on CPU 2, because CPU 2 may already have fetched from location A for every execution of the CLI instruction. A serializing instruction must be in the CPU-2 loop to ensure that CPU 2 will again fetch from location A.

## Channel-Program Serialization

Serialization of a channel program occurs as follows:

1. All storage accesses and storage-key accesses by the channel program follow initiation of the execution of START SUBCHANNEL, or, if suspended, RESUME SUBCHANNEL, as observed by CPUs and by other channel programs. This includes all accesses for the CCWs, IDAWs, and data.
2. All storage accesses and storage-key accesses by the channel program are completed, as observed by CPUs and by other channel programs, before the subchannel status indicating status-pending with primary status is made available to any CPU.
3. If a CCW contains a PCI flag or a suspend flag which is one, all storage accesses and storage-key accesses due to CCWs preceding it in the CCW chain are completed, as observed by CPUs and by other channel programs, before the subchannel status indicating status-pending with intermediate status (PCI or suspended) is made available to any CPU.

The serialization of a channel program does not affect the sequence of storage accesses or storage-key accesses performed by other channel programs or by a CPU.



LzLabs GmbH. Ex. 1008-256

# Chapter 6. Interruptions


Interruption Action . . . . . 6-2
Interruption Code . . . . . 6-5
Enabling and Disabling . . . . . 6-6
Handling of Floating Interruption Conditions 6-6
Instruction-Length Code . . . . . 6-7
Zero ILC . . . . . 6-7
ILC on Instruction-Fetching Exceptions . 6-7
Exceptions Associated with the PSW . . . 6-9
Early Exception Recognition . . . . . 6-9
Late Exception Recognition . . . . . 6-9
External Interruption . . . . . 6-10
Clock Comparator . . . . . 6-11
CPU Timer . . . . . 6-11
Emergency Signal . . . . . 6-11
ETR . . . . . 6-12
External Call . . . . . 6-12
Interrupt Key . . . . . 6-12
Malfunction Alert . . . . . 6-12
Service Signal . . . . . 6-12
TOD-Clock Sync Check . . . . . 6-13
I/O Interruption . . . . . 6-13
Machine-Check Interruption . . . . . 6-14
Program Interruption . . . . . 6-14
Exception-Extension Code . . . . . 6-15
Data-Exception Code (DXC) . . . . . 6-15
Priority of Program Interruptions for Data Exceptions . . . . . 6-15
Program-Interruption Conditions . . . . . 6-16
Addressing Exception . . . . . 6-16
AFX-Translation Exception . . . . . 6-19
ALEN-Translation Exception . . . . . 6-19
ALE-Sequence Exception . . . . . 6-19
ALET-Specification Exception . . . . . 6-19
ASN-Translation-Specification Exception . . . . . 6-19
ASTE-Sequence Exception . . . . . 6-20
ASTE-Validity Exception . . . . . 6-20
ASX-Translation Exception . . . . . 6-21
Crypto-Operation Exception . . . . . 6-21
Data Exception . . . . . 6-21
Decimal-Divide Exception . . . . . 6-22
Decimal-Overflow Exception . . . . . 6-22
Execute Exception . . . . . 6-22
EX-Translation Exception . . . . . 6-22
Extended-Authority Exception . . . . . 6-22
Fixed-Point-Divide Exception . . . . . 6-23
Fixed-Point-Overflow Exception . . . . . 6-23
HFP-Divide Exception . . . . . 6-23
HFP-Exponent-Overflow Exception . . . 6-23
HFP-Exponent-Underflow Exception . . 6-24
HFP-Significance Exception . . . . . 6-24
HFP-Square-Root Exception . . . . . 6-24
LX-Translation Exception . . . . . 6-24
Monitor Event . . . . . 6-24
Operand Exception . . . . . 6-25
Operation Exception . . . . . 6-25
Page-Translation Exception . . . . . 6-26
PC-Translation-Specification Exception 6-27
PER Event . . . . . 6-27
Primary-Authority Exception . . . . . 6-27
Privileged-Operation Exception . . . . . 6-27
Protection Exception . . . . . 6-28
Secondary-Authority Exception . . . . . 6-29
Segment-Translation Exception . . . . . 6-29
Space-Switch Event . . . . . 6-29
Special-Operation Exception . . . . . 6-30
Specification Exception . . . . . 6-31
Stack-Empty Exception . . . . . 6-33
Stack-Full Exception . . . . . 6-33
Stack-Operation Exception . . . . . 6-33
Stack-Specification Exception . . . . . 6-33
Stack-Type Exception . . . . . 6-34
Trace-Table Exception . . . . . 6-34
Translation-Specification Exception . . . 6-34
Unnormalized-Operand Exception . . . 6-35
Vector-Operation Exception . . . . . 6-35
Collective Program-Interruption Names . . 6-35
Recognition of Access Exceptions . . . . . 6-35
Multiple Program-Interruption Conditions . 6-38
Access Exceptions . . . . . 6-40
ASN-Translation Exceptions . . . . . 6-45
Subspace-Replacement Exceptions . . 6-45
Trace Exceptions . . . . . 6-45
Restart Interruption . . . . . 6-46
Supervisor-Call Interruption . . . . . 6-46
Priority of Interruptions . . . . . 6-46


The interruption mechanism permits the CPU to change its state as a result of conditions external to the configuration, within the configuration, or within the CPU itself. To permit fast response to conditions of high priority and immediate recognition of the type of condition, interruption conditions are grouped into six classes: external, input/output, machine check, program, restart, and supervisor call.

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-257

## Interruption Action

An interruption consists in storing the current PSW as an old PSW, storing information identifying the cause of the interruption, and fetching a new PSW. Processing resumes as specified by the new PSW.

The old PSW stored on an interruption normally contains the address of the instruction that would have been executed next had the interruption not occurred, thus permitting resumption of the interrupted program. For program and supervisor-call interruptions, the information stored also contains a code that identifies the length of the last-executed instruction, thus permitting the program to respond to the cause of the interruption. In the case of some program conditions for which the normal response is reexecution of the instruction causing the interruption, the instruction address directly identifies the instruction last executed.

Except for restart, an interruption can occur only when the CPU is in the operating state. The restart interruption can occur with the CPU in either the stopped or operating state.

The details of source identification, location determination, and instruction execution are explained in later sections and are summarized in Figure 6-1 on page 6-3.



LzLabs GmbH. Ex. 1008-258

| Source Identification | Interruption Code | | PSW-Mask Bits | Mask Bits in Ctrl Registers Reg, Bit | ILC Set | Execution of Instruction Identified by Old PSW |
|---|---|---|---|---|---|---|
| MACHINE CHECK (old PSW 48, new PSW 112) | Locations 232-239[1] | | | | | |
| Exigent condition | | | 13 | | u | terminated or nullified[2] |
| Repressible cond | | | 13 | 14, 3-7 | u | unaffected[2] |
| SUPERVISOR CALL (old PSW 32, new PSW 96) | Locations 138-139 | | | | | |
| Instruction bits | 00000000 ssssssss | | | | 1,2 | completed |
| PROGRAM (old PSW 40, new PSW 104) | Locations 142-143 | | | | | |
| | Binary | Hex[3] | | | | |
| Operation | 00000000 p0000001 | 0001 | | | 1,2,3 | suppressed |
| Privileged oper | 00000000 p0000010 | 0002 | | | 2,3 | suppressed |
| Execute | 00000000 p0000011 | 0003 | | | 2 | suppressed |
| Protection | 00000000 p0000100 | 0004 | | | 1,2,3 | suppressed or terminated |
| Addressing | 00000000 p0000101 | 0005 | | | 1,2,3 | suppressed or terminated |
| Specification | 00000000 p0000110 | 0006 | | | 0,1,2,3 | suppressed or completed |
| Data | 00000000 p0000111 | 0007 | | | 1,2,3 | suppressed, terminated, or completed |
| Fixed-pt overflow | xxxxxxxx p0001000 | 0008 | 20 | | 1,2 | completed |
| Fixed-point divide | 00000000 p0001001 | 0009 | | | 1,2 | suppressed or completed |
| Decimal overflow | 00000000 p0001010 | 000A | 21 | | 2,3 | completed |
| Decimal divide | 00000000 p0001011 | 000B | | | 2,3 | suppressed |
| HFP exp. overflow | xxxxxxxx p0001100 | 000C | | | 1,2,3 | completed |
| HFP exp. underflow | xxxxxxxx p0001101 | 000D | 22 | | 1,2,3 | completed |
| HFP significance | xxxxxxxx p0001110 | 000E | 23 | | 1,2 | completed |
| HFP divide | xxxxxxxx p0001111 | 000F | | | 1,2 | suppressed or inhibited[4] |
| Segment transl | 00000000 p0010000 | 0010 | | | 1,2,3 | nullified |
| Page translation | 00000000 p0010001 | 0011 | | | 1,2,3 | nullified |
| Translation spec | 00000000 p0010010 | 0012 | | | 1,2,3 | suppressed |
| Special operation | 00000000 p0010011 | 0013 | | 0, 1 | 1,2,3 | suppressed |
| Operand | 00000000 p0010101 | 0015 | | | 2 | suppressed |
| Trace table | 00000000 p0010110 | 0016 | | | 1,2 | nullified |
| ASN-transl spec | 00000000 p0010111 | 0017 | | | 1,2,3 | suppressed |
| Vector operation[4] | 00000000 p0011001 | 0019 | | | 2,3 | nullified |
| Space-switch event | 00000000 p0011100 | 001C | | 1, 0 | 0,1,2 | completed |
| HFP square root | xxxxxxxx p0011101 | 001D | | | 2,3 | suppressed or inhibited |
| Unnormalized operand[4] | xxxxxxxx p0011110 | 001E | | | 2 | inhibited[4] |
| PC-transl spec | 00000000 p0011111 | 001F | | | 2 | suppressed |
| AFX translation | 00000000 p0100000 | 0020 | | | 1,2 | nullified |
| ASX translation | 00000000 p0100001 | 0021 | | | 1,2 | nullified |
| LX translation | 00000000 p0100010 | 0022 | | | 2 | nullified |

*Figure 6-1 (Part 1 of 4). Interruption Action*


LzLabs GmbH. Ex. 1008-259

| Source Identification | Interruption Code | | PSW-Mask Bits | Mask Bits in Ctrl Registers Reg, Bit | ILC Set | Execution of Instruction Identified by Old PSW |
|---|---|---|---|---|---|---|
| EX translation | 00000000 p0100011 | 0023 | | | 2 | nullified |
| Primary authority | 00000000 p0100100 | 0024 | | | 2 | nullified |
| Secondary auth | 00000000 p0100101 | 0025 | | | 1,2 | nullified |
| ALET specification | 00000000 p0101000 | 0028 | | | 1,2,3 | suppressed |
| ALEN translation | 00000000 p0101001 | 0029 | | | 1,2,3 | nullified |
| ALE sequence | 00000000 p0101010 | 002A | | | 1,2,3 | nullified |
| ASTE validity | 00000000 p0101011 | 002B | | | 1,2,3 | nullified |
| ASTE sequence | 00000000 p0101100 | 002C | | | 1,2,3 | nullified |
| Extended authority | 00000000 p0101101 | 002D | | | 1,2,3 | nullified |
| Stack full | 00000000 p0110000 | 0030 | | | 2 | nullified |
| Stack empty | 00000000 p0110001 | 0031 | | | 1,2 | nullified |
| Stack specification | 00000000 p0110010 | 0032 | | | 1,2 | nullified |
| Stack type | 00000000 p0110011 | 0033 | | | 1,2 | nullified |
| Stack operation | 00000000 p0110100 | 0034 | | | 1,2 | nullified |
| Monitor event | 00000000 p1000000 | 0040 | | 8, 16-31 | 2 | completed |
| PER event | xxxxxxxx 1nnnnnnn[5] | 0080 | 1 | 9, 0-4[8] | 0,1,2,3 | completed[6] |
| Crypto operation | 00000001 p0011001 | 0119 | | | 2 | nullified |
| EXTERNAL (old PSW 24, new PSW 88) | Locations 134-135 | | | | | |
| | Binary | Hex[3] | | | | |
| Interrupt key | 00000000 01000000 | 0040 | 7 | 0, 25 | u | unaffected |
| TOD-clock sync chk | 00010000 00000011 | 1003 | 7 | 0, 19 | u | unaffected |
| Clock comparator | 00010000 00000100 | 1004 | 7 | 0, 20 | u | unaffected |
| CPU timer | 00010000 00000101 | 1005 | 7 | 0, 21 | u | unaffected |
| Malfunction alert | 00010010 00000000 | 1200 | 7 | 0, 16 | u | unaffected |
| Emergency signal | 00010010 00000001 | 1201 | 7 | 0, 17 | u | unaffected |
| External call | 00010010 00000010 | 1202 | 7 | 0, 18 | u | unaffected |
| ETR | 00010100 00000110 | 1406 | 7 | 0, 27 | u | unaffected |
| Service signal | 00100100 00000001 | 2401 | 7 | 0, 22 | u | unaffected |

*Figure 6-1 (Part 2 of 4). Interruption Action*

| Source Identification | Interruption Code | PSW-Mask Bits | Mask Bits in Ctrl Registers Reg, Bit | ILC Set | Execution of Instruction Identified by Old PSW |
|---|---|---|---|---|---|
| INPUT/OUTPUT (old PSW 56, new PSW 120) | Locations 184-191 | | | | |
| I/O-interruption subclass | | 6 | 6, 0-7[7] | u | unaffected |
| RESTART (old PSW 8, new PSW 0) | None | | | | |
| Restart key | | | | u | unaffected |

*Figure 6-1 (Part 3 of 4). Interruption Action*



LzLabs GmbH. Ex. 1008-260

**Explanation:**

Locations for the old PSWs, new PSWs, and interruption codes are real locations.
[1] A model-independent machine-check interruption code of 64 bits is stored at real locations 232-239.
[2] The effect of the machine-check condition is indicated by bits in the machine-check-interruption code. The setting of these bits indicates the extent of the damage and whether the unit of operation is nullified, terminated, or unaffected.
[3] The interruption code in the column labeled "Hex" is the hex code for the basic interruption; this code does not show the effects of concurrent interruption conditions represented by n, p, or x in the column labeled "Binary."
[4] Vector-operation and unnormalized-operand exceptions are associated with the vector facility. "Inhibited" is a type of ending which occurs only for instructions associated with the vector facility. These are described in the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207.
[5] When the interruption code indicates a PER event, an ILC of 0 may be stored only when bits 8-15 of the interruption code are 10000110 (PER, specification).
[6] The unit of operation is completed, unless a program exception concurrently indicated causes the unit of operation to be inhibited, nullified, suppressed, or terminated.
[7] Bits 0-7 of control register 6 provide detailed masking of I/O-interruption subclasses 0-7 respectively.
[8] Additional masks in control register 9, bit positions 16-31, provide detailed control over the source of PER general-register-alteration events which are masked by control register 9, bit 3.
n A possible nonzero code indicating another concurrent program-interruption condition
p If one, the bit indicates a concurrent PER-event interruption condition.
s Bits of the I field of SUPERVISOR CALL.
u Not stored.
x Exception-extension code. This field is described in the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207. This field is set to zero except by vector instructions.

*Figure 6-1 (Part 4 of 4). Interruption Action*

## Interruption Code

The six classes of interruptions (external, I/O, machine check, program, restart, and supervisor call) are distinguished by the storage locations at which the old PSW is stored and from which the new PSW is fetched. For most classes, the causes are further identified by an interruption code and, for some classes, by additional information placed in permanently assigned real storage locations during the interruption. (See also "Assigned Storage Locations" on page 3-43.) For external, program, and supervisor-call interruptions, the interruption code consists of 16 bits.

For external interruptions, the interruption code is stored at real locations 134-135. A parameter may be stored at real locations 128-131, or a CPU address may be stored at real locations 132-133.

For I/O interruptions, the I/O-interruption code is stored at real locations 184-191. The I/O-interruption code consists of a 32-bit subsystem-identification word and a 32-bit interruption parameter.

For machine-check interruptions, the interruption code consists of 64 bits and is stored at real locations 232-239. Additional information for identifying the cause of the interruption and for recovering the state of the machine may be provided by the contents of the machine-check failing-storage address and the contents of the fixed-logout and machine-check-save areas. (See Chapter 11, "Machine-Check Handling.")

For program interruptions, the interruption code is stored at real locations 142-143, and the instruction-length code is stored in bit positions 5 and 6 of real location 141. Further information may be provided in the form of the data-exception code (DXC), translation-exception identification, monitor-class number, PER code, addressing-and-translation-mode identification, PER address, monitor code, exception access identification, and


LzLabs GmbH. Ex. 1008-261

PER access identification, which are stored at real locations 144-161.

## Enabling and Disabling

By means of mask bits in the current PSW, floating-point-control (FPC) register, and control registers, the CPU may be enabled or disabled for all external, I/O, and machine-check interruptions and for some program interruptions. When a mask bit is one, the CPU is enabled for the corresponding class of interruptions, and those interruptions can occur.

When a mask bit is zero, the CPU is disabled for the corresponding interruptions. The conditions that cause I/O interruptions remain pending. External-interruption conditions either remain pending or persist until the cause is removed. Machine-check-interruption conditions, depending on the type, are ignored, remain pending, or cause the CPU to enter the check-stop state. The disallowed program-interruption conditions are ignored, except that some causes are indicated also by the setting of the condition code, and IEEE exceptions set flags in the FPC register. The setting of the HFP-significance and HFP-exponent-underflow program-mask bits affects the manner in which HFP operations are completed when the corresponding condition occurs. Similarly, the setting of the IEEE mask bits in the FPC register affects the manner in which BFP operations are completed when the corresponding condition occurs.

**Programming Notes:**

1. Mask bits in the PSW provide a means of disallowing most maskable interruptions; thus, subsequent interruptions can be disallowed by the new PSW introduced by an interruption. Furthermore, the mask bits can be used to establish a hierarchy of interruption priorities, where a condition in one class can interrupt the program handling a condition in another class but not vice versa. To prevent an interruption-handling routine from being interrupted before the necessary housekeeping steps are performed, the new PSW must disable the CPU for further interruptions within the same class or within a class of lower priority.
2. Because the mask bits in control registers are not changed as part of the interruption procedure, these masks cannot be used to prevent an interruption immediately after a previous interruption in the same class. The mask bits in control registers provide a means for selectively enabling the CPU for some sources and disabling it for others within the same class.
3. Controlling bits exist for several program interruptions, but with no mask bit in the PSW. Such bits include the IEEE mask bits in the FPC register, the monitor masks in bit positions 16-31 of control register 8, and the primary space-switch-event-control bit, bit 0 of control register 1. A bit of this nature is somewhat arbitrarily considered to be a “mask” bit only if the polarity is such that interruption is enabled when the bit is one. Thus, for example, the SSM-suppression-control bit, bit 1 of control register 0, is considered to be a mask bit, while the AFP-register-control bit, bit 13 of control register 0, is not. Regardless of the polarity of such control bits, to avoid another program interruption, an interruption-handling routine must avoid issuing instructions subject to these bits until they have been set appropriately.

## Handling of Floating Interruption Conditions

An interruption condition which can be presented to any CPU in the configuration is called a floating interruption condition. The condition is presented to the first CPU in the configuration which is enabled for the corresponding interruption and which can perform the interruption, and then the condition is cleared and not presented to any other CPU in the configuration. A CPU cannot perform the interruption when it is in the check-stop state, has an invalid prefix, is in a string of program interruptions due to a specification exception of the type which is recognized early, or is in the stopped state. However, a CPU with the rate control set to instruction step can perform the interruption when the start key is activated.

Service signal, I/O, and certain machine-check conditions are floating interruption conditions.



LzLabs GmbH. Ex. 1008-262

## Instruction-Length Code

The instruction-length code (ILC) occupies two bit positions and provides the length of the last instruction executed. It permits identifying the instruction causing the interruption when the instruction address in the old PSW designates the next sequential instruction. The ILC is provided also by the BRANCH AND LINK instructions in the 24-bit addressing mode.

The ILC for program and supervisor-call interruptions is stored in bit positions 5 and 6 of the bytes at real locations 141 and 137, respectively. For external, I/O, machine-check, and restart interruptions, the ILC is not stored since it cannot be related to the length of the last-executed instruction.

For supervisor-call and program interruptions, a nonzero ILC identifies in halfwords the length of the instruction that was last executed. That instruction may be one for which a specification exception was recognized due to an odd instruction address or for which an access exception (addressing, page-translation, protection, segment-translation, or translation-specification) was recognized during the fetching of the instruction. Whenever an instruction is executed by means of EXECUTE, instruction-length code 2 is set to indicate the length of EXECUTE and not that of the target instruction.

The value of a nonzero instruction-length code is related to the leftmost two bits of the instruction. The value does not depend on whether the operation code is assigned or on whether the instruction is installed. The following table summarizes the meaning of the instruction-length code:

| ILC | | Instr. Bits 0-1 | Instruction Length |
|---|---|---|---|
| Decimal | Binary | | |
| 0 | 00 | | Not available |
| 1 | 01 | 00 | One halfword |
| 2 | 10 | 01 | Two halfwords |
| 2 | 10 | 10 | Two halfwords |
| 3 | 11 | 11 | Three halfwords |

### Zero ILC

Instruction-length code 0, after a program interruption, indicates that the instruction address stored in the old PSW does not identify the instruction causing the interruption.

An ILC of 0 occurs when a specification exception due to a PSW-format error is recognized as part of early exception recognition and the PSW has been introduced by LOAD PSW, PROGRAM CALL FAST, PROGRAM RETURN, or an interruption. (See "Exceptions Associated with the PSW" on page 6-9.) In the case of LOAD PSW, PROGRAM CALL FAST, or PROGRAM RETURN, the instruction address of the instruction or of EXECUTE has been replaced in the PSW by the new instruction address. When the invalid PSW is introduced by an interruption, the PSW-format error cannot be attributed to an instruction.

In the case of LOAD PSW, PROGRAM CALL FAST, PROGRAM RETURN, and the supervisor-call interruption, a PER event may be indicated concurrently with a specification exception for which the ILC is 0.

In the case of a PROGRAM RETURN instruction that causes both a space-switch event and a PSW-format error, the space-switch event is recognized, but it is unpredictable whether the ILC is 0 or 1, or 0 or 2 if EXECUTE was used.

### ILC on Instruction-Fetching Exceptions

When a program interruption occurs because of an exception that prohibits access to the instruction, the instruction is considered to have been executed without being fetched, and the instruction-length code cannot be set on the basis of the first two bits of the instruction. As far as the significance of the ILC for this case is concerned, the following two situations are distinguished:

1. When an odd instruction address causes a specification exception to be recognized or when an addressing, protection, or translation-specification exception is encountered on fetching an instruction, the ILC is set to 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the instruction address is incremented by 2, 4, or 6. By reducing the instruction address in the old PSW by the number of halfword locations indicated in the ILC, the instruction address ori-


LzLabs GmbH. Ex. 1008-263

ginally appearing in the PSW may be obtained.

2. When a segment-translation or page-translation exception is recognized while fetching an instruction, the ILC is arbitrarily set to 1, 2, or 3. In this case, the operation is nullified, and the instruction address is not incremented.

The ILC is not necessarily related to the first two bits of the instruction when the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword. The ILC may be arbitrarily set to 1, 2, or 3 in these cases. The instruction address is or is not updated, as described in situations 1 and 2 above.

When any exceptions are encountered on fetching the target instruction of EXECUTE, the ILC is 2.

**Programming Notes:**

1. A nonzero instruction-length code for a program interruption indicates the number of halfword locations by which the instruction address in the program old PSW must be reduced to obtain the instruction address of the last instruction executed, unless one of the following situations exists:

   a. The interruption is caused by an exception resulting in nullification.

   b. An interruption for a PER event occurs before the execution of an interruptible instruction is completed, and no other program-interruption condition is indicated concurrently.

   c. The interruption is caused by a PER event or space-switch event due to LOAD PSW or a branch or linkage instruction, including SUPERVISOR CALL (but not including MONITOR CALL).

   d. The interruption is caused by an addressing exception or protection exception for the storage operand of a LOAD CONTROL instruction that loads the control register (1 or 13) containing the segment-table designation that specifies the address space from which instructions are fetched.

   For situations a and b above, the instruction address in the PSW is not incremented, and the instruction designated by the instruction address is the same as the last one executed. These situations are the only ones in which the instruction address in the old PSW identifies the instruction causing the exception. Situation b can be distinguished from a PER event indicated after completion of an interruptible or noninterruptible instruction in that, for situation b, the instruction address in the PSW is the same as the PER address in the word at real location 152.

   For situation c, the instruction address has been replaced as part of the operation, and the address of the last instruction executed cannot be calculated using the one appearing in the program old PSW.

   For situation d, the effective address of the last instruction executed can be calculated, but, since the segment-table designation for the instruction address space is unpredictable, the corresponding real address is unknown.

2. The instruction-length code (ILC) is redundant when a PER event is indicated since the PER address in the word at real location 152 identifies the instruction causing the interruption (or the EXECUTE instruction, as appropriate). Similarly, the ILC is redundant when the operation is nullified, since in this case the instruction address in the PSW is not incremented. If the ILC value is required in this case, it can be derived from the operation code of the instruction identified by the old PSW.

3. The address of the last instruction executed before a program interruption is insufficient to locate the program problem if one of the following situations exists:

   a. The interruption is caused by an access exception encountered in fetching an instruction, and the instruction address was introduced into the PSW by a means other than sequential operation (by a branch or linkage instruction, LOAD PSW, an interruption, or conclusion of an IPL sequence).

   b. The interruption is caused by a specification exception due to an odd instruction address, which necessarily also results from introduction of an instruction address into the PSW.



LzLabs GmbH. Ex. 1008-264

c. The interruption is caused by an early specification exception due to a STORE THEN OR SYSTEM MASK or SET SYSTEM MASK instruction that switches to or from the real mode while introducing invalid values in bit positions 0-7 of the PSW.

For situations a and b, the instruction address was replaced by the operation preceding the last instruction execution, and the address of the program location related to that preceding operation is unavailable.

For situation c, the address of the last instruction executed is available, but the corresponding real address is unknown.

4. The address of the last instruction executed is not available when an interruption is caused by an early specification exception due to a LOAD PSW or PROGRAM RETURN instruction or an interruption.

## Exceptions Associated with the PSW

Exceptions associated with erroneous information in the current PSW may be recognized when the information is introduced into the PSW or may be recognized as part of the execution of the next instruction. Errors in the PSW which are specification-exception conditions are called PSW-format errors.

### Early Exception Recognition

For the following error conditions, a program interruption for a specification exception occurs immediately after the PSW becomes active:

- A one is introduced into an unassigned bit position of the PSW (that is, any of bit positions 0, 2-4, or 24-31).
- A zero is introduced into bit position 32 of the PSW, but bits 33-39 are not all zeros.
- A zero is introduced into bit position 12 of the PSW.

The interruption occurs regardless of whether the wait state is specified. If the invalid PSW causes the CPU to become enabled for a pending I/O, external, or machine-check interruption, the program interruption occurs instead, and the pending interruption is subject to the mask bits of the new PSW introduced by the program interruption.

When an interruption or the execution of LOAD PSW, PROGRAM CALL FAST, or PROGRAM RETURN introduces a PSW with one of the above error conditions, the instruction-length code is set to 0, and the newly introduced PSW is stored unmodified as the old PSW. When one of the above error conditions is introduced by execution of SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the instruction-length code is set to 2, and the instruction address is incremented by 4. The PSW containing the invalid value introduced into the system-mask field is stored as the old PSW.

When a PSW with one of the above error conditions is introduced during initial program loading, the loading sequence is not completed, and the load indicator remains on.

### Late Exception Recognition

For the following conditions, the exception is recognized as part of the execution of the next instruction:

- A specification exception is recognized due to an odd instruction address in the PSW (PSW bit 63 is one).
- An access exception (addressing, page-translation, protection, segment-translation, or translation-specification) is associated with the location designated by the instruction address or with the location of the second or third halfword of the instruction starting at the designated instruction address.

The instruction-length code and instruction address stored in the program old PSW under these conditions are discussed in “ILC on Instruction-Fetching Exceptions” on page 6-7, and an example is given in Figure 4-6 on page 4-28.

If an I/O, external, or machine-check-interruption condition is pending and the PSW causes the CPU to be enabled for that condition, the corresponding interruption occurs, and the PSW is not inspected for exceptions which are recognized late. Similarly, a PSW specifying the wait state is not inspected for exceptions which are recognized late.


LzLabs GmbH. Ex. 1008-265

**Programming Notes:**

1. The execution of LOAD ADDRESS SPACE PARAMETERS, LOAD PSW, PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, PROGRAM TRANSFER, RESUME PROGRAM, SET PREFIX, SET SECONDARY ASN, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK is suppressed on an addressing or protection exception, and hence the program old PSW provides information concerning the program causing the exception.
2. When the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword, the ILC is not necessarily related to the operation code.
3. If the new PSW introduced by an interruption contains a PSW-format error, a string of interruptions may occur. (See "Priority of Interruptions" on page 6-46.)

## External Interruption

The external interruption provides a means by which the CPU responds to various signals originating from either inside or outside the configuration.

An external interruption causes the old PSW to be stored at real locations 24-31 and a new PSW to be fetched from real locations 88-95.

The source of the interruption is identified in the interruption code which is stored at real locations 134-135. The instruction-length code is not stored.

Additionally, for the malfunction-alert, emergency-signal, and external-call conditions, a 16-bit CPU address is associated with the source of the interruption and is stored at real locations 132-133. When the CPU address is stored, bit 6 of the interruption code is set to one. For all other conditions, no CPU address is stored, bit 6 of the interruption code is set to zero, and zeros are stored at real locations 132-133.

For the ETR and service-signal interruptions, a 32-bit parameter is associated with the interruption and is stored at real locations 128-131. Bit 5 of the external-interruption code indicates that a parameter has been stored. When bit 5 is zero, the contents of real locations 128-131 remain unchanged.

External-interruption conditions are of two types: those for which an interruption-request condition is held pending, and those for which the condition directly requests the interruption. Clock comparator, CPU timer, and TOD-clock sync check are conditions which directly request external interruptions. If a condition which directly requests an external interruption is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from a single occurrence of the condition.

When several interruption requests for a single source are generated before the interruption occurs, and the interruption condition is of the type which is held pending, only one request for that source is preserved and remains pending.

An external interruption for a particular source can occur only when the CPU is enabled for interruption by that source. The external interruption occurs at the completion of a unit of operation. The external mask, PSW bit 7, and external subclass-mask bits in control register 0 control whether the CPU is enabled for a particular source. Each source for an external interruption has a subclass-mask bit assigned to it, and the source can cause an interruption only when the external-mask bit is one and the corresponding subclass-mask bit is one.

When the CPU becomes enabled for a pending external-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

More than one source may present a request for an external interruption at the same time. When the CPU becomes enabled for more than one concurrently pending request, the interruption occurs for the pending condition or conditions having the highest priority.

The priorities for external-interruption requests in descending order are as follows:

- Interrupt key



LzLabs GmbH. Ex. 1008-266

- Malfunction alert
- Emergency signal
- External call
- TOD-clock sync check
- Clock comparator
- CPU timer
- ETR
- Service signal

All requests are honored one at a time. When more than one emergency-signal request exists at a time or when more than one malfunction-alert request exists at a time, the request associated with the smallest CPU address is honored first.

## Clock Comparator

An interruption request for the clock comparator exists whenever either of the following conditions is met:

1. The TOD clock is in the set or not-set state, and the value of the clock comparator is less than the value in the compared portion of the TOD clock, both compare values being considered unsigned binary integers.
2. The TOD clock is in the error or not-operational state.

If the condition responsible for the request is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from a single occurrence of the condition.

When the TOD clock accessed by a CPU is set or changes state, interruption conditions, if any, that are due to the clock comparator may or may not be recognized for up to 1.048576 seconds after the change.

The subclass-mask bit is in bit position 20 of control register 0. This bit is initialized to zero.

The clock-comparator condition is indicated by an external-interruption code of 1004 hex.

## CPU Timer

An interruption request for the CPU timer exists whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). If the value is made positive before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may occur from a single occurrence of the condition.

When the TOD clock accessed by a CPU is set or changes state, interruption conditions, if any, that are due to the CPU timer may or may not be recognized for up to 1.048576 seconds after the change.

The subclass-mask bit is in bit position 21 of control register 0. This bit is initialized to zero.

The CPU-timer condition is indicated by an external-interruption code of 1005 hex.

## Emergency Signal

An interruption request for an emergency signal is generated when the CPU accepts the emergency-signal order specified by a SIGNAL PROCESSOR instruction addressing this CPU. The instruction may have been executed by this CPU or by another CPU in the configuration. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Facilities are provided for holding a separate emergency-signal request pending in the receiving CPU for each CPU in the configuration, including the receiving CPU itself.

The subclass-mask bit is in bit position 17 of control register 0. This bit is initialized to zero.

The emergency-signal condition is indicated by an external-interruption code of 1201 hex. The address of the CPU that executed the SIGNAL PROCESSOR instruction is stored at real locations 132-133.


LzLabs GmbH. Ex. 1008-267

## ETR

An interruption request for the ETR is generated when a port-availability change occurs at any port in the current CPC-port group or when an ETR alert occurs. The terms specific to the ETR are not defined in this publication.

If the same ETR condition occurs more than once before the interruption occurs, the request is generated only once. The request is generated for all CPUs in the configuration.

The subclass-mask bit is in bit position 27 of control register 0. This bit is initialized to zero.

The ETR condition is indicated by an external-interruption code of 1406 hex.

## External Call

An interruption request for an external call is generated when the CPU accepts the external-call order specified by a SIGNAL PROCESSOR instruction addressing this CPU. The instruction may have been executed by this CPU or by another CPU in the configuration. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Only one external-call request, along with the processor address, may be held pending in a CPU at a time.

The subclass-mask bit is in bit position 18 of control register 0. This bit is initialized to zero.

The external-call condition is indicated by an external-interruption code of 1202 hex. The address of the CPU that executed the SIGNAL PROCESSOR instruction is stored at real locations 132-133.

## Interrupt Key

An interruption request for the interrupt key is generated when the operator activates that key. The request is preserved and remains pending in the CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

When the interrupt key is activated while the CPU is in the load state, it depends on the model whether an interruption request is generated or the condition is lost.

The subclass-mask bit is in bit position 25 of control register 0. This bit is initialized to one.

The interrupt-key condition is indicated by an external-interruption code of 0040 hex.

## Malfunction Alert

An interruption request for a malfunction alert is generated when another CPU in the configuration enters the check-stop state or loses power. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Facilities are provided for holding a separate malfunction-alert request pending in the receiving CPU for each of the other CPUs in the configuration. Removal of a CPU from the configuration does not generate a malfunction-alert condition.

The subclass-mask bit is in bit position 16 of control register 0. This bit is initialized to zero.

The malfunction-alert condition is indicated by an external-interruption code of 1200 hex. The address of the CPU that generated the condition is stored at real locations 132-133.

## Service Signal

An interruption request for a service signal is generated upon the completion of certain configuration-control and maintenance functions, such as those initiated by means of the model-dependent DIAGNOSE instruction. A 32-bit parameter is provided with the interruption to assist the program in determining the operation for which the interruption is reported.

Service signal is a floating interruption condition and is presented to the first CPU in the configuration which can perform the interruption. The interruption condition is cleared when it causes an interruption in any one of the CPUs and also by subsystem reset.



LzLabs GmbH. Ex. 1008-268

The subclass-mask bit is in bit position 22 of control register 0. This bit is initialized to zero.

The service-signal condition is indicated by an external-interruption code of 2401 hex. A 32-bit parameter is stored at real locations 128-131.

## TOD-Clock Sync Check

The TOD-clock-sync-check condition indicates that more than one TOD clock exists in the configuration and that bits 32 through the rightmost incremented bit of the clocks are not running in synchronism. When a single clock exists in the configuration, a TOD-clock sync check does not occur.

An interruption request for a TOD-clock sync check exists when the TOD clock accessed by this CPU is running (that is, the clock is in the set or not-set state), the clock accessed by any other CPU in the configuration is running, and bits 32 through the rightmost incremented bit of the two clocks do not match. When a clock is set or changes state, or when a running clock is added to the configuration, a delay of up to 1.048576 seconds ($2^{20}$ microseconds) may occur before the mismatch condition is recognized.

When only two TOD clocks are in the configuration and either or both of the clocks are in the error, stopped, or not-operational state, it is unpredictable whether a TOD-clock-sync-check condition is recognized; if the condition is recognized, it may continue to persist up to 1.048576 seconds after both clocks have been running with bit 32 through the rightmost incremented bit matching. However, in this case, the condition does not persist if one of the TOD clocks is removed from the configuration.

When more than one CPU shares a TOD clock, only the CPU with the smallest CPU address among those sharing the clock indicates a TOD-clock-sync-check condition associated with that clock.

If the condition responsible for the request is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from a single occurrence of the condition.

The subclass-mask bit is in bit position 19 of control register 0. This bit is initialized to zero.

The TOD-clock-sync-check condition is indicated by an external-interruption code of 1003 hex.

## I/O Interruption

The input/output (I/O) interruption provides a means by which the CPU responds to conditions originating in I/O devices and the channel subsystem.

A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests are preserved and remain pending until accepted by a CPU, or until cleared by some other means, such as subsystem reset.

The I/O interruption occurs at the completion of a unit of operation. Priority is established among requests so that in each CPU only one interruption request is processed at a time. Priority among requests for interruptions of differing I/O-interruption subclasses is according to the numerical value of the I/O-interruption subclass (with zero having the highest priority), in conjunction with the I/O-interruption subclass-mask settings in control register 6. For more details, see Chapter 16, “I/O Interruptions.”

When a CPU becomes enabled for I/O interruptions and the channel subsystem has established priority for a pending I/O-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

An I/O interruption causes the old PSW to be stored at real locations 56-63 and a new PSW to be fetched from real locations 120-127. Additional information, in the form of an eight-byte I/O-interruption code, is stored at real locations 184-191. The I/O-interruption code consists of a 32-bit subsystem-identification word and a 32-bit interruption parameter.

An I/O interruption can occur only while a CPU is enabled for the interruption subclass presenting the request. The I/O-mask bit, bit 6 of the PSW, and the I/O-interruption subclass mask in control


LzLabs GmbH. Ex. 1008-269

register 6 determine whether the CPU is enabled for a particular I/O interruption.

I/O interruptions are grouped into eight I/O-interruption subclasses, numbered from 0-7. Each I/O-interruption subclass has an associated I/O-interruption subclass-mask bit in bit positions 0-7 of control register 6. Each subchannel has an I/O-interruption subclass value associated with it. The CPU is enabled for I/O interruptions of a particular I/O-interruption subclass only when PSW bit 6 is one and the associated I/O-interruption subclass-mask bit in control register 6 is also one. If the corresponding I/O-interruption subclass-mask bit is zero, then the CPU is disabled for I/O interruptions with that subclass value. I/O interruptions for all subclasses are disallowed when PSW bit 6 is zero.

## Machine-Check Interruption

The machine-check interruption is a means for reporting to the program the occurrence of equipment malfunctions. Information is provided to assist the program in determining the source of the fault and extent of the damage.

A machine-check interruption causes the old PSW to be stored at real locations 48-55 and a new PSW to be fetched from real locations 112-119.

The cause and severity of the malfunction are identified by a 64-bit machine-check-interruption code stored at real locations 232-239. Further information identifying the cause of the interruption and the location of the fault may be stored at real locations 216-511.

The interruption action and the storing of the associated information are under the control of PSW bit 13 and bits in control register 14. See Chapter 11, “Machine-Check Handling” for more detailed information.

## Program Interruption

Program interruptions are used to report exceptions and events which occur during execution of the program.

A program interruption causes the old PSW to be stored at real locations 40-47 and a new PSW to be fetched from real locations 104-111.

The cause of the interruption is identified by the interruption code. The interruption code is placed at real locations 142-143, the instruction-length code is placed in bit positions 5 and 6 of the byte at real location 141 with the rest of the bits set to zeros, and zeros are stored at real location 140. For some causes, additional information identifying the reason for the interruption is stored at real locations 144-161.

Except for PER events and the crypto-operation exception, the condition causing the interruption is indicated by a coded value placed in the rightmost seven bit positions of the interruption code. Only one condition at a time can be indicated. Bits 0-7 of the interruption code are set to zeros, except when they are set with an exception-extension code by a vector instruction.

PER events are indicated by setting bit 8 of the interruption code to one. When this is the only condition, bits 0-7 and 9-15 are also set to zeros. When a PER event is indicated concurrently with another program-interruption condition, bit 8 is one, and bits 0-7 and 9-15 are set as for the other condition.

The crypto-operation exception is indicated by an interruption code of 0119 hex, or 0199 hex if a PER event is also indicated.

When there is a corresponding mask bit, a program interruption can occur only when that mask bit is one. The program mask in the PSW controls four of the exceptions, the IEEE masks in the FPC register control the IEEE exceptions, bit 1 in control register 0 controls whether SET SYSTEM MASK causes a special-operation exception, bits 16-31 in control register 8 control interruptions due to monitor events, and a hierarchy of masks control interruptions due to PER events. When any controlling mask bit is zero, the condition is ignored; the condition does not remain pending.

**Programming Notes:**

1. When the new PSW for a program interruption has a PSW-format error or causes an exception to be recognized in the process of instruction fetching, a string of program interruptions may occur. See “Priority of Interruptions” on page 6-46 for a description of how such strings are terminated.



LzLabs GmbH. Ex. 1008-270

2. Some of the conditions indicated as program exceptions may be recognized also by the channel subsystem, in which case the exception is indicated in the subchannel-status word or extended-status word.

## Exception-Extension Code

When an arithmetic exception is recognized during execution of an interruptible vector instruction, a nonzero exception-extension code is stored in bits 0-7 of the program-interruption code. This code is set to a nonzero value only for arithmetic exceptions occurring during the execution of vector instructions. For more details, see the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207.

## Data-Exception Code (DXC)

When a data exception causes a program interruption and the basic-floating-point-extensions facility is installed, a data-exception code (DXC) is stored at location 147, and zeros are stored at locations 144-146. The DXC distinguishes between the various types of data-exception conditions. When the AFP-register (additional floating-point register) control bit, bit 13 of control register 0, is one, the DXC is also placed in the DXC field of the floating-point-control (FPC) register. The DXC field in the FPC register remains unchanged when any other program exception is reported. The DXC is an 8-bit code indicating the specific cause of a data exception. The data-exception codes and data exceptions are shown in Figure 6-2 and Figure 6-3 on page 6-16.

### Priority of Program Interruptions for Data Exceptions

When more than one data exception applies and is enabled, the exception with the smallest DXC value is reported. Thus, for example, DXC 2 (BFP instruction) takes precedence over any IEEE exception condition.

When both a specification exception and an AFP-register data exception apply, it is unpredictable which one is reported.

| DXC (Hex) | Data Exception |
|---|---|
| 00 | Decimal operand |
| 01 | AFP register |
| 02 | BFP instruction |
| 08 | IEEE inexact and truncated |
| 0C | IEEE inexact and incremented |
| 10 | IEEE underflow, exact |
| 18 | IEEE underflow, inexact and truncated |
| 1C | IEEE underflow, inexact and incremented |
| 20 | IEEE overflow, exact |
| 28 | IEEE overflow, inexact and truncated |
| 2C | IEEE overflow, inexact and incremented |
| 40 | IEEE division by zero |
| 80 | IEEE invalid operation |

*Figure 6-2. Data-Exception Codes (DXC)*


LzLabs GmbH. Ex. 1008-271

| Exception | Applicable Instruction Types | Effect of CR0.13 | FPC Mask | FPC Flag | DXC (Binary) | Interruption Action | DXC Placed in Real Loc 147 | DXC Placed in FPC Byte 2 |
|---|---|---|---|---|---|---|---|---|
| Decimal operand | Decimal[1] | 0 | none | none | 0000 0000 | Suppress or Terminate | Yes | No |
| | | 1 | | | | | Yes | Yes |
| AFP register | FPS & HFP | 0* | none | none | 0000 0001 | Suppress | Yes | No |
| BFP instruction | BFP | 0* | none | none | 0000 0010 | Suppress | Yes | No |
| IEEE invalid operation | BFP | 1* | 0.0 | 1.0 | 1000 0000 | Suppress | Yes | Yes |
| IEEE division by zero | BFP | 1* | 0.1 | 1.1 | 0100 0000 | Suppress | Yes | Yes |
| IEEE overflow | BFP | 1* | 0.2 | 1.2 | 0010 xy00 | Complete | Yes | Yes |
| IEEE underflow | BFP | 1* | 0.3 | 1.3 | 0001 xy00 | Complete | Yes | Yes |
| IEEE inexact | BFP | 1* | 0.4 | 1.4 | 0000 1y00 | Complete | Yes | Yes |

**Explanation:**

[1] Decimal-operand data exception applies to the decimal instructions (Chapter 8) and the general instructions CONVERT TO BINARY (Chapter 7) and COMPRESSION CALL (SA22-7208).

0* This exception is recognized only when CR0.13 is zero.

1* This exception is recognized only when CR0.13 is one.

xy For IEEE overflow and IEEE underflow, bits 4 and 5 of the DXC are set to 00, 10, or 11 binary, indicating that the result is exact, inexact and truncated, or inexact and incremented, respectively.

y For IEEE inexact, bit 5 of the DXC is set to zero or one, indicating that the result is inexact and truncated or inexact and incremented, respectively.

BFP Binary-floating-point instructions (Chapter 19).

FPS Floating-point-support instructions (Chapter 9).

HFP Hexadecimal-floating-point instructions (Chapter 18).

*Figure 6-3. Data Exceptions*

## Program-Interruption Conditions

The following is a detailed description of each program-interruption condition.

### Addressing Exception

An addressing exception is recognized when the CPU attempts to reference a main-storage location that is not available in the configuration. A main-storage location is not available in the configuration when the location is not installed, when the storage unit is not in the configuration, or when power is off in the storage unit. An address designating a storage location that is not available in the configuration is referred to as invalid.

The operation is suppressed when the address of the instruction is invalid. Similarly, the operation is suppressed when the address of the target instruction of EXECUTE is invalid. Also, the unit of operation is suppressed when an addressing exception is encountered in accessing a table or table entry. The tables and table entries to which the rule applies are the dispatchable-unit-control table, the primary ASN-second-table entry, and entries in the access list, segment table, page



LzLabs GmbH. Ex. 1008-272

table, linkage table, entry table, ASN first table, ASN second table, authority table, linkage stack, and trace table. Addressing exceptions result in suppression when they are encountered for references to the segment table and page table, in both implicit references for dynamic address translation and references associated with the execution of LOAD REAL ADDRESS and TEST PROTECTION. Similarly, addressing exceptions for accesses to the dispatchable-unit-control table, primary ASN-second-table entry, access list, ASN second table, or authority table result in suppression when they are encountered in access-register translation done either implicitly or as part of LOAD REAL ADDRESS, TEST ACCESS, or TEST PROTECTION. Except for some specific instructions whose execution is suppressed, the operation is terminated for an operand address that can be translated but designates an unavailable location. See Figure 6-4 on page 6-18.

For termination, changes may occur only to result fields. In this context, the term “result field” includes the condition code, registers, and any storage locations that are provided and that are designated to be changed by the instruction. Therefore, if an instruction is due to change only the contents of a field in storage, and every byte of the field is in a location that is not available in the configuration, the operation is suppressed. When part of an operand location is available in the configuration and part is not, storing may be performed in the part that is available in the configuration.

When an addressing exception occurs during the fetching of an instruction or during the fetching of a DAT table entry associated with an instruction fetch, it is unpredictable whether the ILC is 1, 2, or 3. When the exception is associated with fetching the target of EXECUTE, the ILC is 2.

In all cases of addressing exceptions not associated with instruction fetching, the ILC is 1, 2, or 3, indicating the length of the instruction that caused the reference.

An addressing exception is indicated by a program-interruption code of 0005 hex (or 0085 hex if a concurrent PER event is indicated).


LzLabs GmbH. Ex. 1008-273

| Exception | Action on | | | |
|---|---|---|---|---|
| | Table-Entry Fetch[1] | Table-Entry Store[2] | Instruction Fetch | Operand Reference |
| Addressing exception | Suppress | Suppress | Suppress | Suppress for IPTE, LASP, LPSW, MSCH, PLO[6], RP, SCKC, SPT, SPX, SSCH, SSM, STCRW, STNSM, STOSM, TPI, and TPROT<br>Terminate for all others.[4] |
| Protection exception for key-controlled protection | -- | -- | Suppress | Suppress for IPTE, LASP, LPSW, MSCH, PLO[6], RP, SCKC, SPT, SPX, SSCH, SSM, STCRW, STNSM, STOSM, and TPI[5]<br>Terminate for all others.[4] |
| Protection exception for access-list-controlled protection | -- | -- | -- | Suppress |
| Protection exception for page protection | -- | Suppress[3] | -- | Suppress for STCRW, STNSM, STOSM, and TPI[5].<br><br>Terminate for all others.[4] |
| Protection exception for low-address protection | -- | Suppress | -- | Suppress for IPTE, STCRW, STNSM, STOSM, and TPI[5].<br><br>Terminate for all others.[4] |

**Explanation:**

-- Not applicable.

[1] Table entries include segment table, page table, PCF-entry table, linkage table, entry table, ASN first table, ASN second table, authority table, dispatchable-unit-control table, primary ASN-second-table-entry, access list, and linkage stack.

[2] Table entries include linkage stack and trace table.

[3] Page protection applies to the linkage stack but not the trace table.

[4] For termination, changes may occur only to result fields. In this context, "result field" includes condition code, registers, and storage locations, if any, which are designated to be changed by the instruction. However, no change is made to a storage location or a storage key when the reference causes an access exception. Therefore, if an instruction is due to change only the contents of a field in main storage, and every byte of that field would cause an access exception, the result is the same as if the operation had been suppressed. If the suppression-on-protection facility is installed, the action is, for page protection, or may be, for key-controlled protection and low-address protection, suppression (except for the condition code) instead of termination; see "Suppression on Protection" in Chapter 3, “Storage.”

[5] When the effective address of TPI is zero, the store access is to implicit real locations 184-191, and key-controlled protection, page protection, and low-address protection do not apply.

[6] Suppression occurs only for the compare-and-load and compare-and-swap operations.

*Figure 6-4. Summary of Action for Addressing and Protection Exceptions*



LzLabs GmbH. Ex. 1008-274

## AFX-Translation Exception

An AFX-translation exception is recognized when, during ASN translation in the space-switching form of PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, or SET SECONDARY ASN, or during ASN translation in PROGRAM RETURN when the restored SASN does not equal the restored PASN, bit 0 of the ASN-first-table entry used is not zero.

The ASN being translated is stored at real locations 146 and 147, and real locations 144 and 145 are set with zeros.

The operation is nullified.

The instruction-length code is 1 or 2.

The AFX-translation exception is indicated by a program-interruption code of 0020 hex (or 00A0 hex if a concurrent PER event is indicated).

## ALEN-Translation Exception

An ALEN-translation exception is recognized during access-register translation when either:

1. The access register used contains an access-list-entry number that designates an access-list entry which is beyond the end of the access list designated by the effective access-list designation.
2. Bit 0 of the access-list entry is not zero.

The number of the access register is stored in bit positions 4-7 of real location 160, and bits 0-3 of the location are set to zeros.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ALEN-translation exception is indicated by a program-interruption code of 0029 hex (or 00A9 hex if a concurrent PER event is indicated).

## ALE-Sequence Exception

An ALE-sequence exception is recognized during access-register translation when the access register used contains an access-list-entry sequence number (ALESN) which is not equal to the ALESN in the access-list entry that is designated by the access register.

The number of the access register is stored in bit positions 4-7 at real location 160, and bits 0-3 are set to zeros.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ALE-sequence exception is indicated by a program-interruption code of 002A hex (or 00AA hex if a concurrent PER event is indicated).

## ALET-Specification Exception

An ALET-specification exception is recognized during access-register translation when bit positions 0-6 of the access-list-entry token in the access register used do not contain all zeros. However, when access-register 0 is used, except in TEST ACCESS, it is treated as containing all zeros, and this exception is not recognized. TEST ACCESS uses the actual contents of access register 0.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

The ALET-specification exception is indicated by a program-interruption code of 0028 hex (or 00A8 hex if a concurrent PER event is indicated).

## ASN-Translation-Specification Exception

An ASN-translation-specification exception may be recognized during ASN translation in the space-switching form of PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, or SET SECONDARY ASN, during ASN translation in PROGRAM RETURN when the restored SASN does not equal the restored PASN, or during ASN translation in LOAD ADDRESS SPACE PARAMETERS, when either:

1. Bit positions 28-31 or 26-31, depending on the address-space-function control, bit 15 of control register 0, of the valid ASN-first-table entry used do not contain zeros.
2. Bit positions 30, 31, and 60-63 of the valid ASN-second-table entry used do not contain zeros.

An ASN-translation-specification exception also may be recognized during implicit access-register translation and during access-register translation


LzLabs GmbH. Ex. 1008-275

in the execution of LOAD REAL ADDRESS, TEST ACCESS, and TEST PROTECTION when bit positions 30, 31, and 60-63 of the valid ASN-second-table entry used do not contain zeros, provided that it is necessary to examine the authority table that is designated by the ASN-second-table entry. This examination is necessary if the private bit in the access-list entry used is not zero and the access-list-entry authorization index in the access-list entry is not equal to the extended authorization index in control register 8.

Whether an ASN-translation-specification exception is recognized in the above cases may depend on the model or may be unpredictable.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

The ASN-translation-specification exception is indicated by a program-interruption code of 0017 hex (or 0097 hex if a concurrent PER event is indicated).

## ASTE-Sequence Exception

An ASTE-sequence exception is recognized when any of the following is true:

1. During access-register translation, except as in 2, the access-list entry used contains an ASN-second-table-entry sequence number (ASTESN) which is not equal to the ASTESN in the ASN-second-table entry that is designated by the access-list entry. The access-list entry is the one designated by the access register used.
2. During access-register translation of ALET 1 by BRANCH IN SUBSPACE GROUP, the subspace ASTESN (SSASTESN) in the dispatchable-unit control table (DUCT) is not equal to the ASTESN in the subspace ASTE designated by the subspace-ASTE origin (SSASTEO) in the DUCT.
3. During a subspace-replacement operation, the subspace ASTESN (SSASTESN) in the dispatchable-unit control table (DUCT) is not equal to the ASTESN in the subspace ASTE designated by the subspace-ASTE origin (SSASTEO) in the DUCT.

In the first and second cases, the number of the access register is stored in bit positions 4-7 at real location 160, and bits 0-3 are set to zeros. In the third case, all zeros are stored at real location 160.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ASTE-sequence exception is indicated by a program-interruption code of 002C hex (or 00AC hex if a concurrent PER event is indicated).

**Programming Note:** The storing of zeros at real location 160 in the case of an ASTE-sequence exception recognized during a subspace-replacement operation is a unique indication since the use of access register 0 in access-register translation cannot result in the exception.

## ASTE-Validity Exception

An ASTE-validity exception is recognized when any of the following is true:

1. During access-register translation, except as in 2, the access-list entry used designates an ASN-second-table entry in which bit 0 is not zero. The access-list entry is the one designated by the access register used.
2. During access-register translation of ALET 1 by BRANCH IN SUBSPACE GROUP, the subspace-ASTE origin (SSASTEO) in the dispatchable-unit control table designates an ASN-second-table entry in which bit 0 is not zero.
3. During a subspace-replacement operation, the subspace-ASTE origin (SSASTEO) in the dispatchable-unit control table designates an ASN-second-table entry in which bit 0 is not zero.

In the first and second cases, the number of the access register is stored in bit positions 4-7 at real location 160, and bits 0-3 are set to zeros. In the third case, all zeros are stored at real location 160.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ASTE-validity exception is indicated by a program-interruption code of 002B hex (or 00AB hex if a concurrent PER event is indicated).



LzLabs GmbH. Ex. 1008-276

**Programming Note:** The storing of zeros at real location 160 in the case of an ASTE-validity exception recognized during a subspace-replacement operation is a unique indication since the use of access register 0 in access-register translation cannot result in the exception.

## ASX-Translation Exception

An ASX-translation exception is recognized when, during ASN translation in the space-switching form of PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, or SET SECONDARY ASN, or during ASN translation in PROGRAM RETURN when the restored SASN does not equal the restored PASN, bit 0 of the ASN-second-table entry used is not zero.

The ASN being translated is stored at real locations 146 and 147, and real locations 144 and 145 are set with zeros.

The operation is nullified.

The instruction-length code is 1 or 2.

The ASX-translation exception is indicated by a program-interruption code of 0021 hex (or 00A1 hex if a concurrent PER event is indicated).

## Crypto-Operation Exception

A crypto-operation exception is recognized when a crypto-facility instruction is executed while bit 29 of control register 0 is zero on a CPU which has the crypto facility installed and available. The crypto-operation exception is also recognized when a crypto-facility instruction is executed and the crypto facility is not installed or available on this CPU, but the facility can be made available to the program either on this CPU or another CPU in the configuration.

When a crypto-facility instruction is executed and the crypto facility is not installed on any CPU which is or can be placed in the configuration, it depends on the model whether a crypto-operation exception or an operation exception is recognized.

The operation is nullified when the crypto-operation exception is recognized.

The instruction-length code is 2.

The crypto-operation exception is indicated by a program-interruption code of 0119 hex (or 0199 hex if a concurrent PER event is indicated).

## Data Exception

The data-exception conditions are shown in Figure 6-3 on page 6-16. A mask bit may or may not control whether an interruption occurs, as noted for each condition.

When a non-maskable data-exception condition is recognized, a program interruption for a data exception always occurs.

Each of the IEEE exception conditions is controlled by a mask bit in the floating-point-control (FPC) register. The handling of these conditions is described in the section "IEEE Exception Conditions" on page 19-10.

A data exception is recognized for the following cases:

- **Decimal-operand** data exception is recognized when an instruction which operates on decimal operands encounters invalid decimal digit or sign codes or has its operands specified improperly. The operation is suppressed when a sign code is invalid; otherwise, the operation is suppressed on some models and terminated on others. See the section "Decimal-Operand Data Exception" on page 8-4 for details. A decimal-operand data exception is also recognized when COMPRESSION CALL (see SA22-7208) encounters errors in its dictionaries, and, in this case, the operation is terminated. When the basic-floating-point-extensions facility is installed, the decimal-operand data exception is reported with DXC 0.
- **AFP-register** data exception is recognized when the basic-floating-point-extensions facility is installed, bit 13 of control register 0 is zero, and a floating-point-support (FPS) instruction or a hexadecimal-floating-point (HFP) instruction specifies a floating-point register other than 0, 2, 4, or 6. The operation is suppressed and is reported with DXC 1.
- **BFP-instruction** data exception is recognized when bit 13 of control register 0 is zero and a BFP instruction is executed. The operation is suppressed and is reported with DXC 2.


LzLabs GmbH. Ex. 1008-277

- **IEEE-exception-condition** data exceptions are recognized when a BFP instruction encounters an exceptional condition. The operation is suppressed or completed, depending on the type of condition. See the section "IEEE Exception Conditions" on page 19-10 for details.

The instruction-length code is 1, 2, or 3.

The data exception is indicated by a program-interruption code of 0007 hex (or 0087 hex if a concurrent PER event is indicated).

## Decimal-Divide Exception

A decimal-divide exception is recognized when in decimal division the divisor is zero or the quotient exceeds the specified data-field size.

The decimal-divide exception is indicated only if the sign codes of both the divisor and dividend are valid and only if the digit or digits used in establishing the exception are valid.

The operation is suppressed.

The instruction-length code is 2 or 3.

The decimal-divide exception is indicated by a program-interruption code of 000B hex (or 008B hex if a concurrent PER event is indicated).

## Decimal-Overflow Exception

A decimal-overflow exception is recognized when one or more nonzero digits are lost because the destination field in a decimal operation is too short to contain the result.

The interruption may be disallowed by the decimal-overflow mask (PSW bit 21).

The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set.

The instruction-length code is 2 or 3.

The decimal-overflow exception is indicated by a program-interruption code of 000A hex (or 008A hex if a concurrent PER event is indicated).

## Execute Exception

The execute exception is recognized when the target instruction of EXECUTE is another EXECUTE.

The operation is suppressed.

The instruction-length code is 2.

The execute exception is indicated by a program-interruption code of 0003 hex (or 0083 hex if a concurrent PER event is indicated).

## EX-Translation Exception

An EX-translation exception is recognized during PC-number translation in PROGRAM CALL when the entry-table entry designated by the entry-index part of the PC number is beyond the end of the entry table as designated by the linkage-table entry used.

The PC number is stored in bit positions 12-31 of the word at real location 144, and the leftmost 12 bits of the word are set to zeros.

An EX-translation exception is recognized in PROGRAM CALL FAST when bits 304-319 (flags) of the PCF-entry-table entry used are not all zeros.

The PC number is stored in bit positions 12-31 of the word at real location 144, bits 0-10 of the word are set to zeros, and bit 11 of the word is set to one.

The operation is nullified.

The instruction-length code is 2.

The EX-translation exception is indicated by a program-interruption code of 0023 hex (or 00A3 hex if a concurrent PER event is indicated).

## Extended-Authority Exception

An extended-authority exception is recognized during access-register translation when all of the following are true:

1. The private bit in the access-list entry used is one.
2. The access-list-entry authorization index (ALEAX) in the access-list entry is not equal to the extended authorization index (EAX) in control register 8.



LzLabs GmbH. Ex. 1008-278

3. Either of the following is true:

   a. The authority-table entry designated by the EAX is beyond the length of the authority table used. The authority table is the one designated by the ASN-second-table entry that is designated by the access-list entry used.

   b. The secondary-authority bit designated by the EAX is zero.

The access-list entry is the one designated by the access register used.

The number of the access register is stored in bit positions 4-7 at real location 160, and bits 0-3 are set to zeros.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The extended-authority exception is indicated by a program-interruption code of 002D hex (or 00AD hex if a concurrent PER event is indicated).

## Fixed-Point-Divide Exception

A fixed-point-divide exception is recognized when either of the following is true:

1. In signed or unsigned binary division, the divisor is zero, or the quotient cannot be expressed as a 32-bit signed or unsigned, respectively, binary integer.
2. The result of CONVERT TO BINARY cannot be expressed as a 32-bit signed binary integer.

In the case of division, the operation is suppressed. The execution of CONVERT TO BINARY is completed by ignoring the leftmost bits that cannot be placed in the register.

The instruction-length code is 1 or 2.

The fixed-point-divide exception is indicated by a program-interruption code of 0009 hex (or 0089 hex if a concurrent PER event is indicated).

## Fixed-Point-Overflow Exception

A fixed-point-overflow exception is recognized when an overflow occurs during signed binary arithmetic or signed left-shift operations.

The interruption may be disallowed by the fixed-point-overflow mask (PSW bit 20).

The operation is completed. The result is obtained by ignoring the overflow information, and condition code 3 is set.

The instruction-length code is 1 or 2.

The fixed-point-overflow exception is indicated by a program-interruption code of XX08 hex (or XX88 hex if a concurrent PER event is indicated), where XX is the exception-extension code.

## HFP-Divide Exception

An HFP-divide exception is recognized when in HFP division the divisor has a zero fraction.

The operation is suppressed, except that the operation is inhibited when the exception is recognized by the vector facility.

The instruction-length code is 1 or 2.

The HFP-divide exception is indicated by a program-interruption code of XX0F hex (or XX8F hex if a concurrent PER event is indicated), where XX is the exception-extension code.

## HFP-Exponent-Overflow Exception

An HFP-exponent-overflow exception is recognized when the result characteristic of an HFP operation exceeds 127 and the result fraction is not zero.

The operation is completed. The fraction is normalized, and the sign and fraction of the result remain correct. The result characteristic is made 128 smaller than the correct characteristic.

The instruction-length code is 1, 2, or 3.

The HFP-exponent-overflow exception is indicated by a program-interruption code of XX0C hex (or XX8C hex if a concurrent PER event is indicated), where XX is the exception-extension code.


LzLabs GmbH. Ex. 1008-279

## HFP-Exponent-Underflow Exception

An HFP-exponent-underflow exception is recognized when the result characteristic of an HFP operation is less than zero and the result fraction is not zero. For an extended-format HFP result, HFP-exponent underflow is indicated only when the high-order characteristic underflows.

The interruption may be disallowed by the HFP-exponent-underflow mask (PSW bit 22).

The operation is completed. The HFP-exponent-underflow mask also affects the result of the operation. When the mask bit is zero, the sign, characteristic, and fraction are set to zero, making the result a true zero. When the mask bit is one, the fraction is normalized, the characteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct.

The instruction-length code is 1, 2, or 3.

The HFP-exponent-underflow exception is indicated by a program-interruption code of XX0D hex (or XX8D hex if a concurrent PER event is indicated), where XX is the exception-extension code.

## HFP-Significance Exception

An HFP-significance exception is recognized when the result fraction in HFP addition or subtraction is zero.

The interruption may be disallowed by the HFP-significance mask (PSW bit 23).

The operation is completed. The HFP-significance mask also affects the result of the operation. When the mask bit is zero, the operation is completed by replacing the result with a true zero. When the mask bit is one, the operation is completed without further change to the characteristic of the result.

The instruction-length code is 1 or 2.

The HFP-significance exception is indicated by a program-interruption code of XX0E hex (or XX8E hex if a concurrent PER event is indicated), where XX is the exception-extension code.

## HFP-Square-Root Exception

An HFP-square-root exception is recognized when the second operand of an HFP SQUARE ROOT instruction is less than zero.

The operation is suppressed, except that the operation is inhibited when the exception is recognized by the vector facility.

The instruction-length code is 2 or 3.

The HFP-square-root exception is indicated by a program-interruption code of 001D hex (or 009D hex if a concurrent PER event is indicated).

## LX-Translation Exception

An LX-translation exception is recognized during PC-number translation in PROGRAM CALL when either:

1. The linkage-table entry designated by the linkage-index part of the PC number is beyond the end of the linkage table as designated by the linkage-table designation used.
2. Bit 0 of the linkage-table entry is not zero.

The PC number is stored in bit positions 12-31 of the word at real location 144, and the leftmost 12 bits of the word are set to zeros.

The operation is nullified.

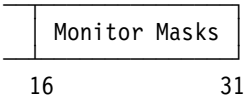
The instruction-length code is 2.

The LX-translation exception is indicated by a program-interruption code of 0022 hex (or 00A2 hex if a concurrent PER event is indicated).

## Monitor Event

A monitor event is recognized when MONITOR CALL is executed and the monitor-mask bit in control register 8 corresponding to the class specified by instruction bits 12-15 is one. The information in control register 8 has the following format:

Control Register 8




The monitor-mask bits, bits 16-31 of control register 8, correspond to monitor classes 0-15, respectively. Any number of monitor-mask bits may be on at a time; together they specify the



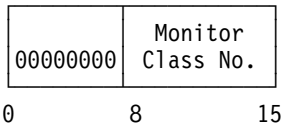
LzLabs GmbH. Ex. 1008-280

classes of monitor events that are monitored at that time. The mask bits are initialized to zeros.

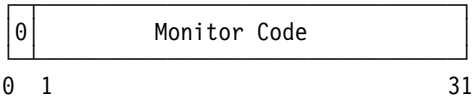
When MONITOR CALL is executed and the corresponding monitor-mask bit is one, a program interruption for monitor event occurs.

Additional information is stored at real locations 148-149 and 156-159. The format of the information stored at these locations is as follows:

Real Locations 148-149




Real Locations 156-159




The contents of bit positions 8-15 of the MONITOR CALL instruction are stored at real location 149 and constitute the monitor-class number. Zeros are stored at real location 148. The effective address specified by the $B_1$ and $D_1$ fields of the instruction forms the monitor code, which is stored in the word at real location 156. The value of the address is under control of the addressing mode, bit 32 of the current PSW; in the 24-bit addressing mode, bits 0-7 of the address are zeros, while in the 31-bit addressing mode, bit 0 is zero.

The operation is completed.

The instruction-length code is 2.

The monitor event is indicated by a program-interruption code of 0040 hex (or 00C0 hex if a concurrent PER event is indicated).

## Operand Exception

An operand exception is recognized when any of the following is true:

1. Execution of CLEAR SUBCHANNEL, HALT SUBCHANNEL, MODIFY SUBCHANNEL, RESUME SUBCHANNEL, START SUBCHANNEL, STORE SUBCHANNEL, or TEST SUBCHANNEL is attempted, and bits 0-15 of general register 1 do not contain 0001 hex.
2. Execution of MODIFY SUBCHANNEL is attempted, and bits 0-1 and 5-7 of word 1 and bits 0-30 (or 0-31 if the concurrent-sense facility is not installed) of word 6 of the SCHIB operand are not all zeros.
3. Execution of MODIFY SUBCHANNEL is attempted, and bits 9 and 10 of word 1 of the SCHIB operand are both ones.
4. Execution of RESET CHANNEL PATH is attempted, and bits 0-23 of general register 1 are not all zeros.
5. Execution of SET ADDRESS LIMIT is attempted, and bits 0 and 16-31 of general register 1 are not all zeros.
6. Execution of SET CHANNEL MONITOR is attempted, bit 30 of general register 1 is one, and bits 0 and 27-31 of general register 2 are not all zeros.
7. Execution of SET CHANNEL MONITOR is attempted, and bits 4-29 of general register 1 are not all zeros.
8. On some models, execution of START SUBCHANNEL is attempted, and bits 5-7, 13-15, and 25-31 of word 1 and bit 0 of word 2 of the ORB operand are not all zeros.
9. On some models, execution of START SUBCHANNEL is attempted, the incorrect-length-indication suppression facility is not installed, and bit 24 of word 1 of the ORB is one.

The operation is suppressed.

The instruction-length code is 2.

The operand exception is indicated by a program-interruption code of 0015 hex (or 0095 hex if a concurrent PER event is indicated).

## Operation Exception

An operation exception is recognized when the CPU attempts to execute an instruction with an invalid operation code. The operation code may be unassigned, or the instruction with that operation code may not be installed on the CPU.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.


LzLabs GmbH. Ex. 1008-281

The operation exception is indicated by a program-interruption code of 0001 hex (or 0081 hex if a concurrent PER event is indicated).

**Programming Notes:**

1. Some models may offer instructions not described in this publication, such as those provided for assists or as part of special or custom features. Consequently, operation codes not described in this publication do not necessarily cause an operation exception to be recognized. Furthermore, these instructions may cause modes of operation to be set up or may otherwise alter the machine so as to affect the execution of subsequent instructions. To avoid causing such an operation, an instruction with an operation code not described in this publication should be executed only when the specific function associated with the operation code is desired.
2. The operation code 00, with a two-byte instruction format, currently is not assigned. It is improbable that this operation code will ever be assigned.

## Page-Translation Exception

Except for the operands of MOVE PAGE, a page-translation exception is recognized when either:

1. The page-table entry indicated by the page-index portion of a virtual address is outside the page table.
2. The page-invalid bit is one.

The exception is recognized as part of the execution of an instruction that needs the page-table entry in the translation of an instruction or operand address, except for the operand address in LOAD REAL ADDRESS and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code.

For the operands of MOVE PAGE, a page-translation exception is recognized when any of the following is true:

1. The page-table entry indicated by the page-index portion of a virtual address is outside the page table.
2. The page-invalid bit is one, and the operand is also invalid in expanded storage. If this is true for both operands, the exception is recognized for the source operand (the second operand).
3. For both operands, the page-invalid bit is one, and the operand is valid in expanded storage. The exception is recognized for the destination operand (the first operand).
4. The page-invalid bit is one for the destination operand, that operand is valid in expanded storage, the source operand is valid in main storage, and the destination-reference-intention bit, bit 22 of general register 0, is one. The exception is recognized for the destination operand.
5. The page-invalid bit is one, and the operand is valid in expanded storage, but either:
   a. The translation path is locked.
   b. The translation path is unlocked, but the expanded-storage block causes an expanded-storage data error.

   The case when for both operands the page-invalid bit is one and the operand is valid in expanded storage is recognized before this case.

When the page-translation-exception condition exists for any of the above reasons other than that the page-table entry is outside the page table, and the condition-code-option bit, bit 23 of general register 0, is one, MOVE PAGE sets a nonzero condition code instead of recognizing the exception. Move-page facility 1 requires bit 23 of general register 0 to be one.

When an interruption occurs, information about the virtual address causing the exception is stored at real locations 144-147 and conditionally at real location 160. See “Assigned Storage Locations” on page 3-43 for a detailed description of this information.

The unit of operation is nullified, except that when the exception is caused by an expanded-storage data error occurring when MOVE PAGE moves data between main storage and expanded storage, the contents of the first-operand location are unpredictable.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during a reference to the target of EXECUTE, the ILC is 2.



LzLabs GmbH. Ex. 1008-282

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The page-translation exception is indicated by a program-interruption code of 0011 hex (or 0091 hex if a concurrent PER event is indicated).

## PC-Translation-Specification Exception

A PC-translation-specification exception is recognized during PC-number translation in PROGRAM CALL when bit position 32 of the entry-table entry used is zero and bit positions 33-39 are not all zeros.

The operation is suppressed.

The instruction-length code is 2.

The PC-translation-specification exception is indicated by a program-interruption code of 001F hex (or 009F hex if a concurrent PER event is indicated).

## PER Event

A PER event is recognized when the CPU is enabled for PER and one or more of these events occur.

The PER mask, bit 1 of the PSW, controls whether the CPU is enabled for PER. When the PER mask is zero, PER events are not recognized. When the bit is one, PER events are recognized, subject to the PER-event-mask bits in control register 9.

The unit of operation is completed, unless another condition has caused the unit of operation to be inhibited, nullified, suppressed, or terminated.

Information identifying the event is stored at real locations 150-155 and conditionally at real location 161.

The instruction-length code is 0, 1, 2, or 3. Code 0 is set only if a specification exception is indicated concurrently.

The PER event is indicated by setting bit 8 of the program-interruption code to one.

See “Program-Event Recording” on page 4-14 for a detailed description of the PER event and the associated interruption information.

## Primary-Authority Exception

A primary-authority exception is recognized during ASN authorization in PROGRAM TRANSFER with space switching (PT-ss) when either:

1. The authority-table entry indicated by the authorization index in control register 4 is beyond the end of the authority table used. The authority table is the one designated by the ASN-second-table entry for the ASN used.
2. The primary-authority bit indicated by the authorization index is zero.

The ASN used is stored at real locations 146-147, and real locations 144-145 are set to zeros.

The operation is nullified.

The instruction-length code is 2.

The primary-authority exception is indicated by a program-interruption code of 0024 hex (or 00A4 hex if a concurrent PER event is indicated).

## Privileged-Operation Exception

A privileged-operation exception is recognized when any of the following is true:

1. Execution of a privileged instruction is attempted in the problem state.
2. The value of the rightmost bit of the general register designated by the $R_2$ field of the PROGRAM TRANSFER instruction is zero and would cause the PSW problem-state bit to change from the problem state (one) to the supervisor state (zero).
3. In the problem state, the key value specified by the second operand of the SET PSW KEY FROM ADDRESS instruction corresponds to a zero PSW-key-mask bit in control register 3.
4. In the problem state, the key value specified by the rightmost byte of the register designated by the $R_3$ field of the MOVE WITH KEY instruction corresponds to a zero PSW-key-mask bit in control register 3.
5. In the problem state, the key value specified by the rightmost byte of the register designated by the $R_3$ field for the instruction MOVE TO PRIMARY, MOVE TO SECONDARY, or


LzLabs GmbH. Ex. 1008-283

MOVE WITH KEY corresponds to a zero PSW-key-mask bit in control register 3.

6. In the problem state, any of the instructions

   - EXTRACT PRIMARY ASN
   - EXTRACT SECONDARY ASN
   - INSERT ADDRESS SPACE CONTROL
   - INSERT PSW KEY
   - INSERT VIRTUAL STORAGE KEY

   is encountered, and the extraction-authority control, bit 4 of control register 0, is zero.

7. In the problem state, the result of ANDing the authorization key mask (AKM) with the PSW-key mask in control register 3 during PROGRAM CALL produces a result of zero.

8. In the problem state, bits 20-23 of the second-operand address of the SET ADDRESS SPACE CONTROL or SET ADDRESS SPACE CONTROL FAST instruction have the value 0011 binary.

9. In the problem state, the key value specified by the rightmost byte of general register 1 for the instruction MOVE WITH SOURCE KEY or MOVE WITH DESTINATION KEY corresponds to a zero PSW-key-mask bit in control register 3.

10. In the problem state, the key value specified by the rightmost byte of the register designated by the $R_1$ field for the instruction BRANCH AND SET AUTHORITY corresponds to a zero PSW-key-mask bit in control register 3.

11. In the problem state, bits 16 and 17 of the PSW field in the second operand of RESUME PROGRAM have the value 11 binary.

The operation is suppressed.

The instruction-length code is 2 or 3.

The privileged-operation exception is indicated by a program-interruption code of 0002 hex (or 0082 hex if a concurrent PER event is indicated).

## Protection Exception

A protection exception is recognized when any of the following is true:

1. *Key-Controlled Protection*: The CPU attempts to access a storage location that is protected against the type of reference, and the access key does not match the storage key.

2. *Access-List-Controlled Protection*: The CPU attempts to store, in the access-register mode, by means of an access-list entry which has the fetch-only bit set to one.

3. *Low-Address Protection*: The CPU attempts a store that is subject to low-address protection, the effective address is in the range 0-511, and the low-address protection control, bit 3 of control register 0, is one.

4. *Page Protection*: The CPU attempts to store, with DAT on, into a page which has the page-protection bit set to one.

The operation is suppressed when the location of the instruction is protected against fetching. Similarly, the operation is suppressed when the location of the target instruction of EXECUTE is protected against fetching.

For access-list-controlled protection, the operation is suppressed. For the other three types of protection, except in the case of some specific instructions whose execution is suppressed, the operation is terminated when a protection exception is encountered during a reference to an operand location. See Figure 6-4 on page 6-18. However, if the suppression-on-protection facility is installed, the operation may be suppressed (except for the condition code) as described in "Suppression on Protection" on page 3-12.

For termination, changes may occur only to result fields. In this context, the term "result field" includes condition code, registers, and storage locations, if any, which are due to be changed by the instruction. However, no change is made to a storage location when a reference to that location causes a protection exception. Therefore, if an instruction is due to change only the contents of a field in storage, and every byte of that field would cause a protection exception, the operation is suppressed. When termination occurs on fetching, the protected information is not loaded into an addressable register nor moved to another storage location.

If the suppression-on-protection facility is installed, information about the address causing the exception is stored at real locations 144-147 and conditionally at real location 160. See "Suppression on Protection" on page 3-12.



LzLabs GmbH. Ex. 1008-284

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2.

For a protected operand location, the instruction-length code (ILC) is 1, 2, or 3, indicating the length of the instruction that caused the reference.

The protection exception is indicated by a program-interruption code of 0004 hex (or 0084 hex if a concurrent PER event is indicated).

## Secondary-Authority Exception

A secondary-authority exception is recognized during ASN authorization in SET SECONDARY ASN with space switching, or during ASN authorization in PROGRAM RETURN when the restored SASN does not equal the restored PASN, when either:

1. The authority-table entry indicated by the authorization index in control register 4 is beyond the end of the authority table used. The authority table is the one designated by the ASN-second-table entry for the ASN used. For PROGRAM RETURN, the ASN is the SASN being restored from the linkage-stack state entry used.
2. The secondary-authority bit indicated by the authorization index is zero.

The ASN used is stored at real locations 146-147, and real locations 144-145 are set to zeros.

The operation is nullified.

The instruction-length code is 1 or 2.

The secondary-authority exception is indicated by a program-interruption code of 0025 hex (or 00A5 hex if a concurrent PER event is indicated).

## Segment-Translation Exception

A segment-translation exception is recognized when either:

1. The segment-table entry indicated by the segment-index portion of a virtual address is outside the segment table.
2. The segment-invalid bit is one.

The exception is recognized as part of the execution of an instruction that needs the segment-table entry in the translation of an instruction or operand address, except for the operand address in LOAD REAL ADDRESS and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code.

When an interruption occurs, information about the virtual address causing the exception is stored at real locations 144-147 and conditionally at real location 160. See “Assigned Storage Locations” on page 3-43 for a detailed description of this information.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The segment-translation exception is indicated by a program-interruption code of 0010 hex (or 0090 hex if a concurrent PER event is indicated).

## Space-Switch Event

A space-switch event is recognized at the completion of the operation in each of the following cases:

1. The space-switching form of PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, or PROGRAM TRANSFER is executed and any of the following is true:
   a. The primary space-switch-event-control bit, bit 0 of control register 1, is one before the operation.
   b. The primary space-switch-event-control bit is one after the operation.
   c. A PER event is indicated.
2. RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST is executed, the CPU is in the home-space mode either before or after the operation, but not both before and after the operation, and any of the following is true:
   a. The primary space-switch-event-control bit, bit 0 of control register 1, is one.


LzLabs GmbH. Ex. 1008-285

b. The home space-switch-event-control bit, bit 0 of control register 13, is one.

c. A PER event is indicated.

For PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, and PROGRAM TRANSFER, and for a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes the translation mode to the home-space mode, the old PASN, which is in the right half of control register 4 before the operation, is stored at real locations 146-147, and the old primary space-switch-event-control bit is placed in bit position 0 and zeros are placed in bit positions 1-15 at real locations 144-145.

For a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes the translation mode away from the home-space mode, zeros are stored at real locations 146-147, and the home space-switch-event-control bit is placed in bit position 0 and zeros are placed in bit positions 1-15 at real locations 144-145.

For a PROGRAM RETURN instruction that introduces a PSW-format error, it is unpredictable whether the instruction-length code is 0 or 1, or 0 or 2 if EXECUTE was used.

The operation is completed.

The instruction-length code is 0, 1, or 2.

The space-switch event is indicated by a program-interruption code of 001C hex (or 009C hex if a concurrent PER event is indicated).

**Programming Notes:**

1. The space-switch event permits the control program to gain control whenever a program enters or leaves a particular address space. The primary space-switch-event-control bit is loaded into control register 1, along with the remaining bits of the primary segment-table designation, whenever control register 1 is loaded.
2. The space-switch event may be useful in obtaining programmed authorization checking, in causing additional trace information to be recorded, or in enabling or disabling the CPU for PER or tracing.
3. Bit 64 of the ASN-second-table entry (ASTE) is loaded into bit position 0 of control register 1 as part of the PC-ss, PR-ss, and PT-ss operations. If bit 64 of the ASTE for a particular address space is set to one, then a space-switch event is recognized when a program enters or leaves the address space by means of any of PC-ss, PR-ss, or PT-ss. Bit 224 of the PCF-entry-table entry provides the same results for PCF-ss.
4. The occurrence of a space-switch event at the completion of a PC-ss, PCF-ss, PR-ss, or PT-ss operation when any PER event is indicated, or at the completion of execution of a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes to or from the home-space mode when any PER event is indicated, permits the control program to determine the address space from which the instruction causing the PER event was fetched.

## Special-Operation Exception

A special-operation exception is recognized when any of the following is true:

1. Execution of SET SYSTEM MASK is attempted in the supervisor state, and the SSM-suppression control, bit 1 of control register 0, is one.
2. Execution of any of the following instructions is attempted with DAT off:
   - EXTRACT PRIMARY ASN
   - EXTRACT SECONDARY ASN
   - INSERT ADDRESS SPACE CONTROL
   - INSERT VIRTUAL STORAGE KEY
   - SET ADDRESS SPACE CONTROL
   - SET SECONDARY ASN
3. Execution of MOVE TO PRIMARY or MOVE TO SECONDARY is attempted, and the CPU is not in the primary-space or secondary-space mode.
4. Execution of basic PROGRAM CALL or PROGRAM TRANSFER is attempted, and the CPU is not in the primary-space mode.
5. Execution of BRANCH AND STACK, stacking PROGRAM CALL, PROGRAM CALL FAST, PROGRAM RETURN, or TRAP is attempted, and the CPU is not in the primary-space or access-register mode.



LzLabs GmbH. Ex. 1008-286

6. Execution of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE or MODIFY STACKED STATE is attempted, and the CPU is not in the primary-space, access-register, or home-space mode.
7. Execution of LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), or SET SECONDARY ASN (SSAR-cp or SSAR-ss) is attempted, or execution of a PROGRAM RETURN instruction requiring PASN or SASN translation is attempted, and the ASN-translation control, bit 12 of control register 14, is zero.
8. Execution of PROGRAM CALL or PROGRAM TRANSFER is attempted, and the subsystem-linkage control, bit 0 of the linkage-table designation in control register 5 or the primary ASN-second-table entry, is zero.
9. Execution of SET ADDRESS SPACE CONTROL, MOVE TO PRIMARY, or MOVE TO SECONDARY is attempted, and the secondary-space control, bit 5 of control register 0, is zero. The exception may be recognized for this reason when execution of SET ADDRESS SPACE CONTROL FAST is attempted.
10. Execution of BRANCH AND SET AUTHORITY, BRANCH AND STACK, BRANCH IN SUBSPACE GROUP, EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, PROGRAM RETURN, or TEST ACCESS is attempted, or execution of a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that is to set the access-register mode is attempted, and the address-space-function control, bit 15 of control register 0, is zero.
11. Execution of BRANCH IN SUBSPACE GROUP is attempted, and any of the following is true:
    a. The current primary address space is not in a subspace group associated with the current dispatchable unit, that is, the primary-ASTE origin (PASTEO) in control register 5 does not equal the base-ASTE origin (BASTEO) in the dispatchable-unit control table (DUCT).
    b. The access-list-entry token (ALET) in access register $R_2$ is ALET 1, but a subspace has not previously been entered by the dispatchable unit by means of BRANCH IN SUBSPACE GROUP, that is, the subspace-ASTE origin (SSASTEO) in the DUCT is all zeros.
    c. The ALET used is other than ALET 0 and ALET 1, and the destination ASTE (DASTE) does not specify the base space or a subspace of the subspace group, that is, the DASTE origin (DASTEO) obtained from an access-list entry does not equal the BASTEO in the DUCT, and either the subspace-group bit (G) in the segment-table designation in the DASTE is zero or the base-space bit (B) in the DASTE is one.
12. Execution of BRANCH AND SET AUTHORITY is attempted, and the $R_2$ field is zero in the base-authority state or nonzero in the reduced-authority state.
13. Execution of TRAP is attempted, and the TRAP-enabled bit, bit 31 in bytes 44-47 of the dispatchable-unit control table, is zero.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

The special-operation exception is indicated by a program-interruption code of 0013 hex (or 0093 hex if a concurrent PER event is indicated).

## Specification Exception

A specification exception is recognized when any of the following is true:

1. A one is introduced into an unassigned bit position of the PSW (that is, any of bit positions 0, 2-4, or 24-31). This is handled as an early PSW specification exception.
2. A zero is introduced into bit position 12 of the PSW. This is handled as an early PSW specification exception.
3. A zero is introduced into bit position 32 of the PSW, but bits 33-39 are not all zeros. This is handled as an early PSW specification exception.
4. The PSW contains an odd instruction address.


LzLabs GmbH. Ex. 1008-287

5. An operand address does not designate an integral boundary in an instruction requiring such integral-boundary designation.
6. An odd-numbered general register is designated by an R field of an instruction that requires an even-numbered register designation.
7. A floating-point register other than 0, 2, 4, or 6 is designated for a short or long operand, or a floating-point register other than 0 or 4 is designated for an extended operand, when the basic-floating-point-extensions facility is not installed.
8. A floating-point register other than 0, 1, 4, 5, 8, 9, 12, or 13 is designated for an extended operand when the basic-floating-point-extensions facility is installed.
9. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.
10. The length of the first-operand field is less than or equal to the length of the second-operand field in decimal multiplication or division.
11. Bit positions 8-11 of MONITOR CALL do not contain zeros.
12. Bits 20 and 21 of the second-operand address of SET ADDRESS SPACE CONTROL or SET ADDRESS SPACE CONTROL FAST are not both zeros.
13. The addressing-mode bit in the general register designated by the $R_2$ field of PROGRAM TRANSFER is zero, but the leftmost seven bits of the instruction address in the same register are not all zeros.
14. Execution of COMPARE AND FORM CODEWORD is attempted, and general registers 1, 2, and 3 do not initially contain even values.
15. Execution of UPDATE TREE is attempted, and the initial contents of general registers 4 and 5 are not a multiple of 8.
16. Execution of EXTRACT STACKED STATE is attempted, and the code in bit positions 56-63 of general register $R_2$ is greater than 3.
17. Execution of MOVE PAGE (facility 1) is attempted, and bit positions 16-23 of general register 0 do not contain 00000001 binary.
18. Execution of MOVE PAGE (facility 2) is attempted, and bit positions 16-19 of general register 0 do not contain zeros or bits 20 and 21 of the register are both one.
19. Execution of COMPARE LOGICAL STRING, MOVE STRING, or SEARCH STRING is attempted, and bits 0-23 of general register 0 are not all zeros.
20. Execution of EXECUTE is attempted, and the target address is odd.
21. Execution of RESUME PROGRAM is attempted, and bits 32-63 of the PSW field in the second operand are not valid for placement in the current PSW. Either bit 32 is zero and bits 33-39 are not all zeros or bit 63 is one.
22. Execution of SET ADDRESSING MODE (SAM24) is attempted, and bits 1-7 of the unupdated instruction address in the PSW, bits 33-39 of the PSW, are not all zeros.
23. Execution of CIPHER MESSAGE, CIPHER MESSAGE WITH CHAINING, COMPUTE INTERMEDIATE MESSAGE DIGEST, COMPUTE LAST MESSAGE DIGEST, or COMPUTE MESSAGE AUTHENTICATION CODE is attempted, and the function code in bits 25-31 of general register 0 contain an unassigned or uninstalled function code.
24. Execution of CIPHER MESSAGE or CIPHER MESSAGE WITH CHAINING is attempted, and the $R_1$ or $R_2$ field designates an odd-numbered register or general register 0.
25. Execution of COMPUTE INTERMEDIATE MESSAGE DIGEST, COMPUTE LAST MESSAGE DIGEST, or COMPUTE MESSAGE AUTHENTICATION CODE is attempted, and the $R_2$ field designates an odd-numbered register or general register 0.
26. Execution of CIPHER MESSAGE, CIPHER MESSAGE WITH CHAINING, COMPUTE INTERMEDIATE MESSAGE DIGEST or COMPUTE MESSAGE AUTHENTICATION CODE is attempted, and the second operand length is not a multiple of the data block size of the designated function. This specification-exception condition does not apply to the query functions.

The execution of the instruction identified by the old PSW is suppressed. However, for early PSW



LzLabs GmbH. Ex. 1008-288

specification exceptions (causes 1-3), the operation that introduces the new PSW is completed, but an interruption occurs immediately thereafter.

Except as noted below, the instruction-length code (ILC) is 1, 2, or 3, indicating the length of the instruction causing the exception.

When the instruction address is odd (cause 4), it is unpredictable whether the ILC is 1, 2, or 3.

When the exception is recognized because of an early PSW specification exception (causes 1-3) and the exception has been introduced by LOAD PSW, PROGRAM CALL FAST, PROGRAM RETURN, or an interruption, the ILC is 0. When the exception is introduced by SET ADDRESSING MODE (SAM24), the ILC is 1, or it is 2 if SET ADDRESSING MODE was the target of EXECUTE. When the exception is introduced by SET SYSTEM MASK or by STORE THEN OR SYSTEM MASK, the ILC is 2.

The specification exception is indicated by a program-interruption code of 0006 hex (or 0086 hex if a concurrent PER event is indicated).

**Programming Note:** See "Exceptions Associated with the PSW" on page 6-9 for a definition of when the exceptions associated with the PSW are recognized.

## Stack-Empty Exception

A stack-empty exception is recognized during the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN when the current linkage-stack entry is a header entry and the backward stack-entry validity bit in the header entry is zero.

The operation is nullified.

The instruction-length code is 1 or 2.

The stack-empty exception is indicated by a program-interruption code of 0031 hex (or 00B1 hex if a concurrent PER event is indicated).

## Stack-Full Exception

A stack-full exception is recognized during the stacking process in BRANCH AND STACK, stacking PROGRAM CALL, or PROGRAM CALL FAST when there is not enough remaining free space in the current linkage-stack section and the forward-section validity bit in the trailer entry of the section is zero.

The operation is nullified.

The instruction-length code is 2.

The stack-full exception is indicated by a program-interruption code of 0030 hex (or 00B0 hex if a concurrent PER event is indicated).

## Stack-Operation Exception

A stack-operation exception is recognized during the unstacking process in PROGRAM RETURN when the unstack-suppression bit is one in any linkage-stack state entry or header entry encountered during the process.

The operation is nullified.

The instruction-length code is 1 or 2.

The stack-operation exception is indicated by a program-interruption code of 0034 hex (or 00B4 hex if a concurrent PER event is indicated).

## Stack-Specification Exception

A stack-specification exception is recognized in each of the following cases:

1. During the stacking process in BRANCH AND STACK, stacking PROGRAM CALL, or PROGRAM CALL FAST when there is not enough remaining free space in the current linkage-stack section and either of the following is true:
   a. The remaining-free-space value used to locate the trailer entry of the current section is not a multiple of 8.
   b. There is not enough remaining free space in the next section.
2. During the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN when the current linkage-stack entry is a header entry in


LzLabs GmbH. Ex. 1008-289

which the backward stack-entry address designates another header entry.

The operation is nullified.

The instruction-length code is 1 or 2.

The stack-specification exception is indicated by a program-interruption code of 0032 hex (or 00B2 hex if a concurrent PER event is indicated).

## Stack-Type Exception

A stack-type exception is recognized during the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN in each of the following cases:

1. The current linkage-stack entry is not a header entry or a state entry.
2. When the current linkage-stack entry is a header entry, the preceding entry, designated by the backward stack-entry address in the header entry, is not a header entry or a state entry. (A stack-specification exception is recognized if the preceding entry is a header entry.)

The operation is nullified.

The instruction-length code is 1 or 2.

The stack-type exception is indicated by a program-interruption code of 0033 hex (or 00B3 hex if a concurrent PER event is indicated).

## Trace-Table Exception

A trace-table exception is recognized when the CPU attempts to store a trace-table entry which would reach or cross the next 4K-byte block boundary. For the purpose of recognizing this exception in the TRACE instruction, the explicit trace entry is treated as being 76 bytes long.

The operation is nullified.

The instruction-length code is 1, 2, or 3, indicating the length of the instruction causing the exception.

The trace-table exception is indicated by a program-interruption code of 0016 hex (or 0096 hex if a concurrent PER event is indicated).

## Translation-Specification Exception

A translation-specification exception is recognized when translation of a virtual address is attempted and any of the following is true:

1. Bit positions 8-12 of control register 0 do not contain the code 10110.
2. The segment-table entry used for the translation is valid, and bit position 0 in the entry does not contain zero.
3. The page-table entry used for the translation is valid, and bit positions 0, 20, and 23 in the entry do not contain zeros.
4. The private-space control, bit 23, in the segment-table designation used for the translation is one, the segment-table entry used for the translation is valid, and the common-segment bit, bit 27, in the segment-table entry is one.

Any of reasons 2-4 is referred to by saying that the DAT-table entry has a format error.

The exception is recognized only as part of the execution of an instruction using address translation, that is, when DAT is on and a logical address, instruction address, or virtual address must be translated, or when LOAD REAL ADDRESS or INVALIDATE PAGE TABLE ENTRY is executed. Cause 1 is recognized on any translation attempt; causes 2, 3, and 4 are recognized only for table entries that are actually used.

The unit of operation is suppressed.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The translation-specification exception is indicated by a program-interruption code of 0012 hex (or 0092 hex if a concurrent PER event is indicated).

**Programming Note:** When a translation-specification exception is recognized in the process of translating an instruction address, the operation is suppressed. In this case, the instruction-length code (ILC) is needed to derive



LzLabs GmbH. Ex. 1008-290

the address of the instruction, as the instruction address in the old PSW has been incremented by the amount indicated by the ILC. In the case of segment-translation and page-translation exceptions, the operation is nullified, the instruction address in the old PSW identifies the instruction, and the ILC may be arbitrarily set to 1, 2, or 3.

### Unnormalized-Operand Exception

An unnormalized-operand exception is recognized when, in a vector floating-point divide or multiply operation, a source-operand element has a nonzero fraction with a leftmost hexadecimal digit of zero. For more details, see the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207.

The unit of operation is inhibited.

The instruction-length code is 2.

The unnormalized-operand exception is indicated by a program-interruption code of XX1E hex (or XX9E hex if a concurrent PER event is indicated), where XX is the exception-extension code.

### Vector-Operation Exception

A vector-operation exception is recognized when a vector-facility instruction is executed while bit 14 of control register 0 is zero on a CPU which has the vector facility installed and available. The vector-operation exception is also recognized when a vector-facility instruction is executed and the vector facility is not installed or available on this CPU, but the facility can be made available to the program either on this CPU or another CPU in the configuration.

When a vector-facility instruction is executed, and the vector facility is not installed on any CPU which is or can be placed in the configuration, it depends on the model whether a vector-operation exception or an operation exception is recognized.

The operation is nullified when the vector-operation exception is recognized.

The instruction-length code is 2 or 3.

The vector-operation exception is indicated by a program-interruption code of 0019 hex (or 0099 hex if a concurrent PER event is indicated).

## Collective Program-Interruption Names

For the sake of convenience, certain program exceptions are grouped together under a single collective name. These collective names are used when it is necessary to refer to the complete set of exceptions, such as in instruction definitions. Four collective names are used:

- Access exceptions
- ASN-translation exceptions
- Subspace-replacement exceptions
- Trace exceptions

The individual exceptions and their priorities are listed in “Multiple Program-Interruption Conditions” on page 6-38.

## Recognition of Access Exceptions

Figure 6-5 on page 6-36 summarizes the conditions that can cause access exceptions and the action taken when they are encountered.


LzLabs GmbH. Ex. 1008-291

| Condition | Translation for Virtual Address of LRA | | Translation for TAR and TPROT, and Access for Logical Address of TPROT[1] | | Translation and Access for Any Other Address | |
|---|---|---|---|---|---|---|
| | Indi-cation | Action | Indi-cation | Action | Indi-cation | Action |
| Access register[2] | | | | | | |
| Bits 0-6 not all zeros | cc3 | Complete | cc3 | Complete | AS | Suppress |
| Effective access-list designation[2] | | | | | | |
| Designation protected against fetching | - | - | - | - | - | - |
| Invalid address of designation | A | Suppress | A | Suppress | A | Suppress |
| Access-list entry[2] | | | | | | |
| Access-list-length violation | cc3 | Complete | cc3 | Complete | AT | Nullify |
| Entry protected against fetching | - | - | - | - | - | - |
| Invalid address of entry | A | Suppress | A | Suppress | A | Suppress |
| I bit on | cc3 | Complete | cc3 | Complete | AT | Nullify |
| Sequence number in access register not equal to sequence number in entry | cc3 | Complete | cc3 | Complete | ALQ | Nullify |
| ASN-second-table entry[2] | | | | | | |
| Entry protected against fetching | - | - | - | - | - | - |
| Invalid address of entry | A | Suppress | A | Suppress | A | Suppress |
| I bit on | cc3 | Complete | cc3 | Complete | AV | Nullify |
| Sequence number in access-list entry not equal to sequence number in entry | cc3 | Complete | cc3 | Complete | ASQ | Nullify |
| Bits 30, 31, and 60-63 not all zeros[3] | ATS | Suppress | ATS | Suppress | ATS | Suppress |
| Authority-table entry[2] [4] | | | | | | |
| Authority-table-length violation | cc3 | Complete | cc3 | Complete | EA | Nullify |
| Entry protected against fetching | - | - | - | - | - | - |
| Invalid address of entry | A | Suppress | A | Suppress | A | Suppress |
| Secondary-authority bit not one | cc3 | Complete | cc3 | Complete | EA | Nullify |
| Control-register-0 contents[5] | | | | | | |
| Invalid encoding of bits 8-12 | TS | Suppress | -[6] | -[6] | TS | Suppress |
| Segment-table entry | | | | | | |
| Segment-table-length violation | cc3 | Complete | cc3 | Complete | ST | Nullify |
| Entry protected against fetching | - | - | - | - | - | - |
| Invalid address of entry | A | Suppress | A | Suppress | A | Suppress |
| I bit on | cc1 | Complete | cc3 | Complete | ST | Nullify |
| One in a bit position which is checked for zero[7] | TS | Suppress | TS | Suppress | TS | Suppress |
| Page-table entry | | | | | | |
| Page-table-length violation | cc3 | Complete | cc3 | Complete | PT | Nullify |
| Entry protected against fetching | - | - | - | - | - | - |
| Invalid address of entry | A | Suppress | A | Suppress | A | Suppress |
| I bit on | cc2 | Complete | cc3 | Complete | PT | Nullify |
| One in a bit position which is checked for zero[7] | TS | Suppress | TS | Suppress | TS | Suppress |
| Access for instruction fetch | | | | | | |
| Location protected | - | - | - | - | P | Suppress |
| Invalid address | - | - | - | - | A | Suppress |
| Access for operands | | | | | | |
| Location protected | - | - | cc set[8] | Complete | P | Term.* |
| Invalid address | - | - | A | Suppress | A | Term.* |

*Figure 6-5 (Part 1 of 2). Handling of Access Exceptions*



LzLabs GmbH. Ex. 1008-292

**Explanation:**

| | |
|---|---|
| - | The condition does not apply. |
| * | Action is to terminate except where otherwise specified in this publication. For access-list-controlled protection, the action is always to suppress. |
| 1 | TAR does not have a logical address. The rows "Control-register-0 contents" through "Access for operands" apply only to TPROT, not to TAR. |
| 2 | Exceptions related to an access register, effective access-list designation, access-list entry, ASN-second-table entry, or authority-table entry are recognized only in the access-register mode, except that for LOAD REAL ADDRESS they are recognized when PSW bits 16 and 17 are 01 binary, and for TEST ACCESS they are recognized regardless of the translation mode. |
| 3 | An ASN-translation-specification exception is recognized only if it is necessary to access the authority table. |
| 4 | Authority table is not accessed and secondary-authority bit is not checked if the private bit in the access-list entry is zero or the access-list-entry authorization index in the access-list entry is equal to the extended authorization index in control register 8. |
| 5 | A translation-specification exception for an invalid code in control register 0, bit positions 8-12, is recognized as part of the execution of the instruction using address translation; when DAT is on, it is recognized during translation of the instruction address, and, when DAT is off, it is only recognized during execution of INVALIDATE PAGE TABLE ENTRY or for translation of the operand address of LOAD REAL ADDRESS. |
| 6 | A translation-specification exception cannot occur for the logical address of TEST PROTECTION because this exception would have been recognized during the instruction fetch for the instruction. |
| 7 | A translation-specification exception for a format error in a table entry is recognized only when the execution of an instruction requires the entry for translation of an address. |
| 8 | The condition code is set as follows:<br>0 Operand location not protected.<br>1 Fetches permitted, but stores not permitted.<br>2 Neither fetches nor stores permitted. |
| A | Addressing exception. |
| ALQ | ALE-sequence exception. |
| AS | ALET-specification exception. |
| ASQ | ASTE-sequence exception. |
| AT | ALEN-translation exception. |
| ATS | ASN-translation-specification exception. |
| AV | ASTE-validity exception. |
| cc1 | Condition code 1 set. |
| cc2 | Condition code 2 set. |
| cc3 | Condition code 3 set. |
| EA | Extended-authority exception. |
| P | Protection exception. |
| PT | Page-translation exception. |
| ST | Segment-translation exception. |
| TS | Translation-specification exception. |

*Figure 6-5 (Part 2 of 2). Handling of Access Exceptions*

Any access exception is recognized as part of the execution of the instruction with which the exception is associated. An access exception is not recognized when the CPU attempts to prefetch from an unavailable location or detects some other access-exception condition, but a branch instruction or an interruption changes the instruction sequence such that the instruction is not executed.

Every instruction can cause an access exception to be recognized because of instruction fetch. Additionally, access exceptions associated with instruction execution may occur because of an access to an operand in storage.

An access exception due to fetching an instruction is indicated when the first instruction halfword cannot be fetched without encountering the exception. When the first halfword of the instruction has no access exceptions, access exceptions may be indicated for additional halfwords according to the instruction length specified by the first two bits of the instruction; however, when the operation can be performed without accessing the second or third halfwords of the instruction, it is unpredictable whether the access exception is indicated for the unused part. Since the indication of access exceptions for instruction fetch is common to all instructions, it is not covered in the individual instruction definitions.


LzLabs GmbH. Ex. 1008-293

Except where otherwise indicated in the individual instruction description, the following rules apply for exceptions associated with an access to an operand location. For a fetch-type operand, access exceptions are necessarily indicated only for that portion of the operand which is required for completing the operation. It is unpredictable whether access exceptions are indicated for those portions of a fetch-type operand which are not required for completing the operation. For a store-type operand, access exceptions are recognized for the entire operand even if the operation could be completed without the use of the inaccessible part of the operand. In situations where the value of a store-type operand is defined to be unpredictable, it is unpredictable whether an access exception is indicated.

Whenever an access to an operand location can cause an access exception to be recognized, the word "access" is included in the list of program exceptions in the description of the instruction. This entry also indicates which operand can cause the exception to be recognized and whether the exception is recognized on a fetch or store access to that operand location. Access exceptions are recognized only for the portion of the operand as defined for each particular instruction.

## Multiple Program-Interruption Conditions

Except for PER events, only one program-interruption condition is indicated with a program interruption. The existence of one condition, however, does not preclude the existence of other conditions. When more than one program-interruption condition exists, only the condition having the highest priority is identified in the interruption code.

With two conditions of the same priority, it is unpredictable which is indicated. In particular, the priority of access exceptions associated with the two parts of an operand that crosses a page or protection boundary is unpredictable and is not necessarily related to the sequence specified for the access of bytes within the operand.

The type of ending which occurs (nullification, suppression, or termination) is that which is defined for the type of exception that is indicated in the interruption code. However, if a condition is indicated which permits termination, and another condition also exists which would cause either nullification or suppression, then the unit of operation is suppressed.

Figure 6-6 on page 6-39 lists the priorities of all program-interruption conditions other than PER events and exceptions associated with some of the more complex control instructions. All exceptions associated with references to storage for a particular instruction halfword or a particular operand byte are grouped as a single entry called "access." Figure 6-7 on page 6-42 lists the priority of access exceptions for a single access. Thus, the second figure specifies which of several exceptions, encountered either in the access of a particular portion of an instruction or in any particular access associated with an operand, has highest priority, and the first figure specifies the priority of this condition in relation to other conditions detected in the operation. Similarly, the priorities for exceptions occurring as part of ASN translation and tracing are covered in Figure 6-8 on page 6-45 and Figure 6-10 on page 6-45, respectively.

For some instructions, the priority is shown in the individual instruction description.

The relative priorities of any two conditions listed in the figure can be found by comparing the priority numbers, as found in the figure, from left to right until a mismatch is found. If the first inequality is between numeric characters, either the two conditions are mutually exclusive or, if both can occur, the condition with the smaller number is indicated. If the first inequality is between alphabetic characters, then the two conditions are not exclusive, and it is unpredictable which is indicated when both occur.

To understand the use of the table, consider an example involving the instruction ADD DECIMAL, which is a six-byte instruction. Assume that the first four bytes of the instruction can be accessed but that the instruction crosses a boundary so that an addressing exception exists for the last two bytes. Additionally, assume that the first operand addressed by the instruction contains invalid decimal digits and is in a location that can be fetched from, but not stored into, because of key-controlled protection. The three exceptions which could result from attempted execution of the ADD DECIMAL are:



LzLabs GmbH. Ex. 1008-294

| Priority Number | Exception |
|---|---|
| 7.B | Access exceptions for third instruction halfword. |
| 8.B | Access exceptions (operand 1). |
| 8.D | Data exception. |

Since the first inequality (7≠8) is between numeric characters, the addressing exception would be indicated. If, however, the entire ADD DECIMAL instruction can be fetched, and only the second two exceptions listed above exist, then the inequality (B≠D) is between alphabetic characters, and it is unpredictable whether the protection exception or the data exception would be indicated.

| | |
|---|---|
| 1. | Specification exception due to any PSW error of the type that causes an immediate interruption.[1] |
| 2. | Specification exception due to an odd instruction address in the PSW. |
| 3. | Access exceptions for first halfword of EXECUTE.[2] |
| 4. | Access exceptions for second halfword of EXECUTE.[2] |
| 5. | Specification exception due to target instruction of EXECUTE not being specified on halfword boundary.[2] |
| 6. | Access exceptions for first instruction halfword. |
| 7.A | Access exceptions for second instruction halfword.[3] |
| 7.B | Access exceptions for third instruction halfword.[3] |
| 7.C.1 | Vector-operation exception. |
| 7.C.2 | Operation exception. |
| 7.C.3 | Privileged-operation exception for privileged instructions. |
| 7.C.4 | Execute exception. |
| 7.C.5 | Special-operation exception. |
| 8.A | Specification exception due to conditions other than those included in 1, 2, and 5 above. |
| 8.B[4] | Access exceptions for an access to an operand in storage.[5] |
| 8.C[4] | Access exceptions for any other access to an operand in storage.[5] |
| 8.D | Data exception.[6] |
| 8.E | Decimal-divide exception.[7] |
| 8.F | Trace exceptions. |
| 9. | Events other than PER events, exceptions which result in completion, and the following exceptions: fixed-point divide, HFP divide, operand, HFP square root, and unnormalized operand. Either these exceptions and events are mutually exclusive or their priority is specified in the corresponding definitions. |

*Figure 6-6 (Part 1 of 2). Priority of Program-Interruption Conditions*


LzLabs GmbH. Ex. 1008-295

**Explanation:**

Numbers indicate priority, with "1" being the highest priority; letters indicate no priority.

[1] PSW errors which cause an immediate interruption may be introduced by a new PSW loaded as a result of an interruption or by the instructions LOAD PSW, PROGRAM RETURN, SET SYSTEM MASK, and STORE THEN OR SYSTEM MASK. The priority shown in the chart is for a PSW error introduced by an interruption and may also be considered as the priority for a PSW error introduced by the previous instruction. The error is introduced only if the instruction encounters no other exceptions. The resulting interruption has a higher priority than any interruption caused by the instruction which would have been executed next; it has lower priority, however, than any interruption caused by the instruction which introduced the erroneous PSW.

[2] Priorities 3, 4, and 5 are for the EXECUTE instruction, and priorities starting with 6 are for the target instruction. When no EXECUTE is encountered, priorities 3, 4, and 5 do not apply.

[3] Separate accesses may occur for each halfword of an instruction. The second instruction halfword is accessed only if bits 0-1 of the instruction are not both zeros. The third instruction halfword is accessed only if bits 0-1 of of the instruction are both ones. Access exceptions for one of these halfwords are not necessarily recognized if the instruction can be completed without use of the contents of the halfword or if an exception of lower priority can be determined without the use of the halfword.

[4] As in instruction fetching, separate accesses may occur for each portion of an operand. Each of these accesses, and also accesses for different operands, are of equal priority, and the two entries 8.B and 8.C are listed to represent the relative priorities of exceptions associated with any two of these accesses. Access exceptions for INSERT STORAGE KEY EXTENDED, INSERT VIRTUAL STORAGE KEY, INVALIDATE PAGE TABLE ENTRY, LOAD REAL ADDRESS, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY EXTENDED, and TEST PROTECTION are also included in 8.B.

[5] For MOVE LONG, MOVE LONG EXTENDED, COMPARE LOGICAL LONG, and COMPARE LOGICAL LONG EXTENDED, an access exception for a particular operand can be indicated only if the R field for that operand designates an even-numbered register.

[6] A decimal-operand data exception can be indicated only if the sign, digit, or digits responsible for the exception were fetched without encountering an access exception.

[7] The exception can be indicated only if the digits used in establishing the exception, and also the signs, were fetched without encountering an access exception, only if the signs are valid, and only if the digits used in establishing the exception are valid.

*Figure 6-6 (Part 2 of 2). Priority of Program-Interruption Conditions*

## Access Exceptions

The access exceptions consist of those exceptions which can be encountered while using an absolute, instruction, logical, real, or virtual address to access storage. Thus, in the access-register mode, the exceptions are:

1. ALET specification
2. ALEN translation
3. ALE sequence
4. ASTE validity
5. ASTE sequence
6. ASN-translation specification
7. Extended authority
8. Addressing (the ART tables)
9. Translation specification
10. Segment translation



LzLabs GmbH. Ex. 1008-296

11. Page translation
12. Addressing (the DAT tables)
13. Addressing (the operand or instruction)
14. Protection (key-controlled, access-list-controlled, page, and low-address)

With DAT on but in other than the access-register mode, exceptions 9-14 in the above list, except for access-list-controlled protection, can be encountered.

With DAT off, the exceptions are:

1. Addressing (the operand or instruction)
2. Protection (key-controlled and low-address)

Additionally, the instructions LOAD REAL ADDRESS and INVALIDATE PAGE TABLE ENTRY can encounter a translation-specification exception and addressing exceptions for the ART and DAT tables even with DAT off.


LzLabs GmbH. Ex. 1008-297

| | |
|---|---|
| A. | Protection exception (low-address protection) due to a store-type operand reference with an effective address in the range 0-511. Not recognized if DAT is on and the segment-table designation to be used in the translation cannot be obtained because of another exception. |
| B.1.A.1 | ALET-specification exception due to bits 0-6 of access register not being all zeros.[1] |
| B.1.A.2 | Addressing exception for access to effective access-list designation.[2] |
| B.1.A.3 | ALEN-translation exception due to access-list entry being outside the list.[1] |
| B.1.A.4 | Addressing exception for access to access-list entry.[2] |
| B.1.A.5 | ALEN-translation exception due to I bit in access-list entry having the value one.[1] |
| B.1.A.6 | ALE-sequence exception due to access-list-entry sequence number (ALESN) in access register not being equal to ALESN in access-list entry.[1] |
| B.1.A.7 | Addressing exception for access to ASN-second-table entry.[2] |
| B.1.A.8 | ASTE-validity exception due to I bit in ASN-second-table entry having the value one.[1] |
| B.1.A.9 | ASTE-sequence exception due to ASN-second-table-entry sequence number (ASTESN) in access-list entry not being equal to ASTESN in ASN-second-table entry.[1] |
| B.1.A.10 | ASN-translation-specification exception due to a one in bit positions 30, 31, or 60-63 of ASN-second-table entry (optional and only if authority-table access is required).[2]<br><br>**Note:** Exceptions B.1.A.11 through B.1.A.13 are recognized only when the private bit in the access-list entry is one and the access-list extended-authorization index (ALEAX) in the entry is not equal to the extended-authorization index in control register 8. |
| B.1.A.11 | Extended-authority exception due to authority-table entry being outside table.[1] |
| B.1.A.12 | Addressing exception for access to authority-table entry.[2] |

*Figure 6-7 (Part 1 of 4). Priority of Access Exceptions*



LzLabs GmbH. Ex. 1008-298

| | |
|---|---|
| B.1.A.13 | Extended-authority exception due to (1) private bit in access-list entry not being zero, (2) access-list-entry authorization index in access-list entry not being equal to extended authorization index in control register 8, and (3) secondary-authority bit selected by extended authorization index not being one.[1] |
| B.1.B | Translation-specification exception due to invalid encoding of bits 8-12 of control register 0.[3] |
| B.2.A | Protection exception (access-list-controlled protection) due to store-type operand reference to a virtual address which is protected against stores.[1] |
| B.2.B.1 | Segment-translation exception due to segment-table entry being outside table.[4] |
| B.2.B.2 | Addressing exception for access to segment-table entry.[5] |
| B.2.B.3 | Segment-translation exception due to I bit in segment-table entry having the value one.[4] |
| B.2.B.4 | Translation-specification exception due to invalid ones in segment-table entry (bit 0, and common-segment bit if private-space bit in segment-table designation is one).[5] |
| B.2.B.5 | Page-translation exception due to page-table entry being outside table.[4] |
| B.2.B.6 | Addressing exception for access to page-table entry.[3] |
| B.2.B.7 | Page-translation exception due to I bit in page-table entry having the value one (not recognized for operand of MOVE PAGE).[4] |
| B.2.B.8.A | Translation-specification exception due to invalid ones in page-table entry (bits 0, 20, and 23).[5] |
| B.3.A | Protection exception (page protection) due to a store-type operand reference to a virtual address which is protected against stores.[6] |
| B.3.B | Addressing exception for access to instruction or operand. |
| B.4. | Protection exception (key-controlled protection) due to attempt to access a protected instruction or operand location. |

*Figure 6-7 (Part 2 of 4). Priority of Access Exceptions*


LzLabs GmbH. Ex. 1008-299

| | |
|---|---|
| | **Note:** The following access exceptions are recognized only by MOVE PAGE. |
| B.2.B.8.B | Page-translation exception due to I bit in page-table entry having the value one and the operand not being valid in expanded storage.[6] If this is true for both operands, the exception is recognized for the second operand.[7] |
| B.2.B.9.A.1 | Page-translation exception due to both operands being valid in expanded storage. The exception is recognized for the first operand.[8] |
| B.2.B.9.A.2 | Page-translation exception due to the translation path to the operand being locked.[8] |
| B.2.B.9.B | Page-translation exception due to the operand being the first operand, the second operand being valid in main storage, and the destination-reference-intention bit in general register 0 being one (facility 2 only).[8] |
| B.2.B.9.C | Protection exception (page protection or key-controlled protection) due to the operand being protected. |
| B.3.C | Page-translation exception due to the accesses to the operand resulting in an expanded-storage data error.[7] |

**Explanation:**

[1] Not applicable when not in the access-register mode; not applicable for execution of TEST ACCESS and for translation of operand address of LOAD REAL ADDRESS and TEST PROTECTION.

[2] Not applicable when not in the access-register mode, except applicable for execution of TEST ACCESS and, when PSW bits 16 and 17 are 01 binary, for translation of operand address of LOAD REAL ADDRESS.

[3] Not applicable when DAT is off, except for execution of INVALIDATE PAGE TABLE ENTRY and for translation of operand address of LOAD REAL ADDRESS.

[4] Not applicable when DAT is off; not applicable to operand addresses for LOAD REAL ADDRESS and TEST PROTECTION.

[5] Not applicable when DAT is off except for translation of operand address for LOAD REAL ADDRESS.

[6] Not applicable when DAT is off.

*Figure 6-7 (Part 3 of 4). Priority of Access Exceptions*



LzLabs GmbH. Ex. 1008-300

**Explanation** (Continued):

7 With move-page facility 1, or with move-page facility 2 when the condition-code-option bit is one, the exception is not recognized. Instead, condition code 1 is set if the condition is true for only the first operand, or condition code 2 is set if the condition is true for the second operand or both operands.

8 With move-page facility 1, or with move-page facility 2 when the condition-code-option bit is one, the exception is not recognized. Instead, condition code 1 is set.

*Figure 6-7 (Part 4 of 4). Priority of Access Exceptions*

## ASN-Translation Exceptions

The ASN-translation exceptions are those exceptions which are common to the process of translating an ASN in the instructions PROGRAM RETURN, PROGRAM TRANSFER, and SET SECONDARY ASN. The exceptions and the priority in which they are detected are shown in Figure 6-8.

| | |
|---|---|
| 1. | Addressing exception for access to ASN-first-table entry. |
| 2. | AFX-translation exception due to I bit (bit 0) in ASN-first-table entry being one. |
| 3. | ASN-translation-specification exception due to invalid ones (bits 28-31) in first-table entry. |
| 4. | Addressing exception for access to ASN-second-table entry. |
| 5. | ASX-translation exception due to I bit (bit 0) in ASN-second-table entry being one. |
| 6. | ASN-translation-specification exception due to invalid ones (bits 30, 31, 60-63) in ASN-second-table entry (optional). |

*Figure 6-8. Priority of ASN-Translation Exceptions*

## Subspace-Replacement Exceptions

The subspace-replacement exceptions are those exceptions which can be recognized during a subspace-replacement operation in PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, or SET SECONDARY ASN. The exceptions can be recognized only if the subspace-group facility is installed and the address-space-function control, bit 15 of control register 0, is one. The exceptions and their priority are shown in Figure 6-9.

| | |
|---|---|
| 1. | Addressing exception for access to dispatchable-unit control table. |
| 2. | Addressing exception for access to subspace ASN-second-table entry. |
| 3. | ASTE-validity exception due to bit 0 being one in subspace ASN-second-table entry. |
| 4. | ASTE-sequence exception due to subspace ASN-second-table-entry sequence number in dispatchable-unit control table not being equal to ASN-second-table-entry sequence number in subspace ASN-second-table entry. |

*Figure 6-9. Priority of Subspace-Replacement Exceptions*

## Trace Exceptions

The trace exceptions are those exceptions which can be encountered while forming a trace-table entry. The exceptions and their priority are shown in Figure 6-10.

| | |
|---|---|
| A. | Protection exception (low-address protection) due to entry address being in the range 0-511. |
| B.1 | Trace-table exception due to new entry reaching or crossing next 4K-byte boundary. |
| B.2 | Addressing exception for access to trace-table entry. |

*Figure 6-10. Priority of Trace Exceptions*



LzLabs GmbH. Ex. 1008-301

## Restart Interruption

The restart interruption provides a means for the operator or another CPU to invoke the execution of a specified program. The CPU cannot be disabled for this interruption.

A restart interruption causes the old PSW to be stored at real location 8 and a new PSW, designating the start of the program to be executed, to be fetched from real location 0. The instruction-length code and interruption code are not stored.

If the CPU is in the operating state, the exchange of the PSWs occurs at the completion of the current unit of operation and after all other pending interruption conditions for which the CPU is enabled have been honored. If the CPU is in the stopped state, the CPU enters the operating state and exchanges the PSWs without first honoring any other pending interruptions.

The restart interruption is initiated by activating the restart key. The operation can also be initiated at the addressed CPU by executing a SIGNAL PROCESSOR instruction which specifies the restart order.

When the rate control is set to the instruction-step position, it is unpredictable whether restart causes a unit of operation or additional interruptions to be performed after the PSWs have been exchanged.

**Programming Note:** To perform a restart when the CPU is in the check-stop state, the CPU has to be reset. Resetting with loss of the least amount of information can be accomplished by means of the system-reset-normal key, which does not clear the contents of program-addressable registers, including the control registers, but causes the channel subsystem to be reset. The CPU-reset SIGNAL PROCESSOR order can be used to clear the CPU without affecting the channel subsystem.

## Supervisor-Call Interruption

The supervisor-call interruption occurs when the instruction SUPERVISOR CALL is executed. The CPU cannot be disabled for the interruption, and the interruption occurs immediately upon the execution of the instruction.

The supervisor-call interruption causes the old PSW to be stored at real location 32 and a new PSW to be fetched from real location 96.

The contents of bit positions 8-15 of the SUPERVISOR CALL instruction are placed in the rightmost byte of the interruption code. The leftmost byte of the interruption code is set to zero. The instruction-length code is 1, unless the instruction was executed by means of EXECUTE, in which case the code is 2.

The interruption code is placed at real locations 138-139; the instruction-length code is placed in bit positions 5 and 6 of the byte at real location 137, with the other bits set to zeros; and zeros are stored at real location 136.

## Priority of Interruptions

During the execution of an instruction, several interruption-causing events may occur simultaneously. The instruction may give rise to a program interruption, a request for an external interruption may be received, equipment malfunctioning may be detected, an I/O-interruption request may be made, and the restart key may be activated. Instead of the program interruption, a supervisor-call interruption might occur; or both can occur if PER is active. Simultaneous interruption requests are honored in a predetermined order.

An exigent machine-check condition has the highest priority. When it occurs, the current operation is terminated or nullified. Program and supervisor-call interruptions that would have occurred as a result of the current operation may be eliminated. Any pending repressible machine-check conditions may be indicated with the exigent machine-check interruption. Every reasonable attempt is made to limit the side effects of an exigent machine-check condition, and requests for external, I/O, and restart interruptions normally remain unaffected.

In the absence of an exigent machine-check condition, interruption requests existing concurrently at the end of a unit of operation are honored, in descending order of priority, as follows:

- Supervisor call
- Program
- Repressible machine check
- External



LzLabs GmbH. Ex. 1008-302

- Input/output
- Restart

The processing of multiple simultaneous interruption requests consists in storing the old PSW and fetching the new PSW belonging to the interruption first honored. This new PSW is subsequently stored without the execution of any instructions, and the new PSW associated with the next interruption is fetched. Storing and fetching of PSWs continues until no more interruptions are to be serviced. The priority is reevaluated after each new PSW is loaded. Each evaluation takes into consideration any additional interruptions which may have become pending. Additionally, external and I/O interruptions, as well as machine-check interruptions due to repressible conditions, occur only if the current PSW at the instant of evaluation indicates that the CPU is interruptible for the cause.

Instruction execution is resumed using the last-fetched PSW. The order of executing interruption subroutines is, therefore, the reverse of the order in which the PSWs are fetched.

If the new PSW for a program interruption does not specify the wait state and has an odd instruction address, or causes an access exception to be recognized, another program interruption occurs. Since this second interruption introduces the same unacceptable PSW, a string of interruptions is established. These program exceptions are recognized as part of the execution of the following instruction, and the string may be broken by an external, I/O, machine-check, or restart interruption or by the stop function.

If the new PSW for a program interruption contains a zero in bit position 12 or contains a one in an unassigned bit position or if the leftmost seven bits of the instruction address are not zeros when bit 32 indicates 24-bit addressing, another program interruption occurs. This condition is of higher priority than restart, I/O, external, or repressible machine-check conditions, or the stop function, and CPU reset has to be used to break the string of interruptions.

A string of interruptions for other interruption classes can also exist if the new PSW allows the interruption which has just occurred. These include machine-check interruptions, external interruptions, and I/O interruptions due to PCI conditions generated because of CCWs which form a loop. Furthermore, a string of interruptions involving more than one interruption class can exist. For example, assume that the CPU timer is negative and the CPU-timer subclass mask is one. If the external new PSW has a one in an unassigned bit position, and the program new PSW is enabled for external interruptions, then a string of interruptions occurs, alternating between external and program. Even more complex strings of interruptions are possible. As long as more interruptions must be serviced, the string of interruptions cannot be broken by employing the stop function; CPU reset is required.

Similarly, CPU reset has to be invoked to terminate the condition that exists when an interruption is attempted with a prefix value designating a storage location that is not available to the CPU.

Interruptions for all requests for which the CPU is enabled occur before the CPU is placed in the stopped state. When the CPU is in the stopped state, restart has the highest priority.

**Programming Note:** The order in which concurrent interruption requests are honored can be changed to some extent by masking.


LzLabs GmbH. Ex. 1008-303

LzLabs GmbH. Ex. 1008-304

# Chapter 7. General Instructions


Data Format . . . . . 7-2
Binary-Integer Representation . . . . . 7-2
Binary Arithmetic . . . . . 7-3
  Signed Binary Arithmetic . . . . . 7-3
    Addition and Subtraction . . . . . 7-3
    Fixed-Point Overflow . . . . . 7-4
  Unsigned Binary Arithmetic . . . . . 7-4
Signed and Logical Comparison . . . . . 7-4
Instructions . . . . . 7-5
  ADD . . . . . 7-12
  ADD HALFWORD . . . . . 7-12
  ADD HALFWORD IMMEDIATE . . . . . 7-12
  ADD LOGICAL . . . . . 7-13
  ADD LOGICAL WITH CARRY . . . . . 7-13
  AND . . . . . 7-13
  BRANCH AND LINK . . . . . 7-14
  BRANCH AND SAVE . . . . . 7-15
  BRANCH AND SAVE AND SET MODE . 7-16
  BRANCH AND SET MODE . . . . . 7-16
  BRANCH ON CONDITION . . . . . 7-17
  BRANCH ON COUNT . . . . . 7-18
  BRANCH ON INDEX HIGH . . . . . 7-18
  BRANCH ON INDEX LOW OR EQUAL . . 7-18
  BRANCH RELATIVE AND SAVE . . . . . 7-19
  BRANCH RELATIVE AND SAVE LONG . 7-19
  BRANCH RELATIVE ON CONDITION . . 7-20
  BRANCH RELATIVE ON CONDITION LONG . . . . . 7-20
  BRANCH RELATIVE ON COUNT . . . . . 7-21
  BRANCH RELATIVE ON INDEX HIGH . . 7-21
  BRANCH RELATIVE ON INDEX LOW OR EQUAL . . . . . 7-21
  CHECKSUM . . . . . 7-22
  CIPHER MESSAGE (KM) . . . . . 7-26
  CIPHER MESSAGE WITH CHAINING (KMC) . . . . . 7-26
  COMPARE . . . . . 7-35
  COMPARE AND FORM CODEWORD . . 7-36
  COMPARE AND SWAP . . . . . 7-40
  COMPARE DOUBLE AND SWAP . . . . . 7-40
  COMPARE HALFWORD . . . . . 7-42
  COMPARE HALFWORD IMMEDIATE . . 7-42
  COMPARE LOGICAL . . . . . 7-42
  COMPARE LOGICAL CHARACTERS UNDER MASK . . . . . 7-43
  COMPARE LOGICAL LONG . . . . . 7-43
  COMPARE LOGICAL LONG EXTENDED 7-45
  COMPARE LOGICAL LONG UNICODE . 7-47
  COMPARE LOGICAL STRING . . . . . 7-50
  COMPARE UNTIL SUBSTRING EQUAL . 7-51
  COMPUTE INTERMEDIATE MESSAGE DIGEST (KIMD) . . . . . 7-55
  COMPUTE LAST MESSAGE DIGEST (KLMD) . . . . . 7-55
  COMPUTE MESSAGE AUTHENTICATION CODE (KMAC) . . 7-61
  CONVERT TO BINARY . . . . . 7-66
  CONVERT TO DECIMAL . . . . . 7-67
  CONVERT UNICODE TO UTF-8 . . . . . 7-67
  CONVERT UTF-8 TO UNICODE . . . . . 7-70
  COPY ACCESS . . . . . 7-72
  DIVIDE . . . . . 7-73
  DIVIDE LOGICAL . . . . . 7-73
  EXCLUSIVE OR . . . . . 7-74
  EXECUTE . . . . . 7-74
  EXTRACT ACCESS . . . . . 7-75
  EXTRACT PSW . . . . . 7-76
  INSERT CHARACTER . . . . . 7-76
  INSERT CHARACTERS UNDER MASK . 7-76
  INSERT PROGRAM MASK . . . . . 7-77
  LOAD . . . . . 7-77
  LOAD ACCESS MULTIPLE . . . . . 7-77
  LOAD ADDRESS . . . . . 7-78
  LOAD ADDRESS EXTENDED . . . . . 7-78
  LOAD ADDRESS RELATIVE LONG . . . 7-79
  LOAD AND TEST . . . . . 7-79
  LOAD COMPLEMENT . . . . . 7-79
  LOAD HALFWORD . . . . . 7-80
  LOAD HALFWORD IMMEDIATE . . . . . 7-80
  LOAD MULTIPLE . . . . . 7-80
  LOAD NEGATIVE . . . . . 7-80
  LOAD POSITIVE . . . . . 7-81
  LOAD REVERSED . . . . . 7-81
  MONITOR CALL . . . . . 7-82
  MOVE . . . . . 7-83
  MOVE INVERSE . . . . . 7-83
  MOVE LONG . . . . . 7-83
  MOVE LONG EXTENDED . . . . . 7-87
  MOVE LONG UNICODE . . . . . 7-90
  MOVE NUMERICS . . . . . 7-93
  MOVE PAGE (Facility 1) . . . . . 7-93
  MOVE STRING . . . . . 7-95
  MOVE WITH OFFSET . . . . . 7-97
  MOVE ZONES . . . . . 7-97
  MULTIPLY . . . . . 7-98
  MULTIPLY HALFWORD . . . . . 7-98
  MULTIPLY HALFWORD IMMEDIATE . . . 7-98
  MULTIPLY LOGICAL . . . . . 7-99
  MULTIPLY SINGLE . . . . . 7-99
  OR . . . . . 7-100


© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-305

PACK . . . . . 7-100
PACK ASCII . . . . . 7-101
PACK UNICODE . . . . . 7-102
PERFORM LOCKED OPERATION . . . 7-103
ROTATE LEFT SINGLE LOGICAL . . . 7-116
SEARCH STRING . . . . . 7-116
SET ACCESS . . . . . 7-117
SET ADDRESSING MODE . . . . . 7-117
SET PROGRAM MASK . . . . . 7-118
SHIFT LEFT DOUBLE . . . . . 7-118
SHIFT LEFT DOUBLE LOGICAL . . . . 7-119
SHIFT LEFT SINGLE . . . . . 7-119
SHIFT LEFT SINGLE LOGICAL . . . . . 7-120
SHIFT RIGHT DOUBLE . . . . . 7-120
SHIFT RIGHT DOUBLE LOGICAL . . . 7-121
SHIFT RIGHT SINGLE . . . . . 7-121
SHIFT RIGHT SINGLE LOGICAL . . . . 7-121
STORE . . . . . 7-122
STORE ACCESS MULTIPLE . . . . . 7-122
STORE CHARACTER . . . . . 7-122
STORE CHARACTERS UNDER MASK 7-122
STORE CLOCK . . . . . 7-123
STORE CLOCK EXTENDED . . . . . 7-124
STORE HALFWORD . . . . . 7-126
STORE MULTIPLE . . . . . 7-126
STORE REVERSED . . . . . 7-126
SUBTRACT . . . . . 7-127
SUBTRACT HALFWORD . . . . . 7-127
SUBTRACT LOGICAL . . . . . 7-127
SUBTRACT LOGICAL WITH BORROW 7-128
SUPERVISOR CALL . . . . . 7-129
TEST ADDRESSING MODE . . . . . 7-129
TEST AND SET . . . . . 7-129
TEST UNDER MASK . . . . . 7-130
TEST UNDER MASK HIGH . . . . . 7-130
TEST UNDER MASK LOW . . . . . 7-130
TRANSLATE . . . . . 7-131
TRANSLATE AND TEST . . . . . 7-132
TRANSLATE EXTENDED . . . . . 7-132
TRANSLATE ONE TO ONE . . . . . 7-134
TRANSLATE ONE TO TWO . . . . . 7-135
TRANSLATE TWO TO ONE . . . . . 7-135
TRANSLATE TWO TO TWO . . . . . 7-135
UNPACK . . . . . 7-139
UNPACK ASCII . . . . . 7-139
UNPACK UNICODE . . . . . 7-140
UPDATE TREE . . . . . 7-141

This chapter includes all the unprivileged instructions described in this publication other than the decimal and floating-point instructions.

## Data Format

The general instructions treat data as being of four types: signed binary integers, unsigned binary integers, unstructured logical data, and decimal data. Data is treated as decimal by the conversion, packing, and unpacking instructions. Decimal data is described in Chapter 8, “Decimal Instructions.”

The general instructions manipulate data which resides in general registers or in storage or is introduced from the instruction stream. Some general instructions operate on data which resides in the PSW or the TOD clock.

In a storage-and-storage operation the operand fields may be defined in such a way that they overlap. The effect of this overlap depends upon the operation. When the operands remain unchanged, as in COMPARE or TRANSLATE AND TEST, overlapping does not affect the execution of the operation. For instructions such as MOVE and TRANSLATE, one operand is replaced by new data, and the execution of the operation may be affected by the amount of overlap and the manner in which data is fetched or stored. For purposes of evaluating the effect of overlapped operands, data is considered to be handled one eight-bit byte at a time. Special rules apply to the operands of MOVE LONG and MOVE INVERSE. See “Interlocks within a Single Instruction” on page 5-80 for how overlap is detected in the access-register mode.

## Binary-Integer Representation

Binary integers are treated as signed or unsigned.

In an unsigned binary integer, all bits are used to express the absolute value of the number. When two unsigned binary integers of different lengths are added, the shorter number is considered to be extended on the left with zeros.

In some operations, the result is achieved by the use of the one's complement of the number. The one's complement of a number is obtained by inverting each bit of the number, including the sign.



LzLabs GmbH. Ex. 1008-306

For signed binary integers, the leftmost bit represents the sign, which is followed by the numeric field. Positive numbers are represented in true binary notation with the sign bit set to zero. When the value is zero, all bits are zeros, including the sign bit. Negative numbers are represented in two's-complement binary notation with a one in the sign-bit position.

Specifically, a negative number is represented by the two's complement of the positive number of the same absolute value. The two's complement of a number is obtained by forming the one's complement of the number, adding a value of one in the rightmost bit position, allowing a carry into the sign position, and ignoring any carry out of the sign position.

This number representation can be considered the rightmost portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number are zeros. When the number is negative, these bits are ones. Therefore, when a signed operand must be extended with bits on the left, the extension is achieved by setting these bits equal to the sign bit of the operand.

The notation for signed binary integers does not include a negative zero. It has a number range in which, for a given length, the set of negative nonzero numbers is one larger than the set of positive nonzero numbers. The maximum positive number consists of a sign bit of zero followed by all ones, whereas the maximum negative number (the negative number with the greatest absolute value) consists of a sign bit of one followed by all zeros.

A signed binary integer of either sign, except for zero and the maximum negative number, can be changed to a number of the same magnitude but opposite sign by forming its two's complement. Forming the two's complement of a number is equivalent to subtracting the number from zero. The two's complement of zero is zero.

The two's complement of the maximum negative number cannot be represented in the same number of bits. When an operation, such as LOAD COMPLEMENT, attempts to produce the two's complement of the maximum negative number, the result is the maximum negative number, and a fixed-point-overflow exception is recognized. An overflow does not result, however, when the maximum negative number is complemented as an intermediate result but the final result is within the representable range. An example of this case is a subtraction of the maximum negative number from -1. The product of two maximum negative numbers of a given length is representable as a positive number of double that length.

In discussions of signed binary integers in this publication, a signed binary integer includes the sign bit. Thus, the expression “32-bit signed binary integer” denotes an integer with 31 numeric bits and a sign bit, and the expression “64-bit signed binary integer” denotes an integer with 63 numeric bits and a sign bit.

In an arithmetic operation, a carry out of the numeric field of a signed binary integer is carried into the sign bit. However, in algebraic left-shifting, the sign bit does not change even if significant numeric bits are shifted out.

**Programming Notes:**

1. An alternate way of forming the two's complement of a signed binary integer is to invert all bits to the left of the rightmost one bit, leaving the rightmost one bit and all zero bits to the right of it unchanged.
2. The numeric bits of a signed binary integer may be considered to represent a positive value, with the sign representing a value of either zero or the maximum negative number.

# Binary Arithmetic

## Signed Binary Arithmetic

### Addition and Subtraction

Addition of signed binary integers is performed by adding all bits of each operand, including the sign bits. When one of the operands is shorter, the shorter operand is considered to be extended on the left to the length of the longer operand by propagating the sign-bit value.

Subtraction is performed by adding the one's complement of the second operand and a value of one to the first operand.



LzLabs GmbH. Ex. 1008-307

### Fixed-Point Overflow

A fixed-point-overflow condition exists for signed binary addition or subtraction when the carry out of the sign-bit position and the carry out of the left-most numeric bit position disagree. Detection of an overflow does not affect the result produced by the addition. In mathematical terms, signed addition and subtraction produce a fixed-point overflow when the result is outside the range of representation for signed binary integers. Specifically, for ADD and SUBTRACT, which operate on 32-bit signed binary integers, there is an overflow when the proper result would be greater than or equal to $+2^{31}$ or less than $-2^{31}$. The actual result placed in the general register after an overflow differs from the proper result by $2^{32}$. A fixed-point overflow causes a program interruption if allowed by the program mask.

The instructions SHIFT LEFT SINGLE and SHIFT LEFT DOUBLE produce an overflow when the result is outside the range of representation for signed binary integers. The actual result differs from that for addition and subtraction in that the sign of the result remains the same as the original sign.

## Unsigned Binary Arithmetic

Addition of unsigned binary integers is performed by adding all bits of each operand. When one of the operands is shorter, the shorter operand is considered to be extended on the left with zeros. Unsigned binary arithmetic is used in address arithmetic for adding the X, B, and D fields. (See “Address Generation” on page 5-7.) It is also used to obtain the addresses of the function bytes in TRANSLATE and TRANSLATE AND TEST. Furthermore, unsigned binary arithmetic is used on 32-bit unsigned binary integers by ADD LOGICAL and SUBTRACT LOGICAL. Given the same two operands, ADD and ADD LOGICAL produce the same 32-bit result. The instructions differ only in the interpretation of this result. ADD interprets the result as a signed binary integer and inspects it for sign, magnitude, and overflow to set the condition code accordingly. ADD LOGICAL interprets the result as an unsigned binary integer and sets the condition code according to whether the result is zero and whether there was a carry out of bit position 0. Such a carry is not considered an overflow, and no program interruption for overflow can occur for ADD LOGICAL.

SUBTRACT LOGICAL differs from ADD LOGICAL in that the one's complement of the second operand and a value of one are added to the first operand.

**Programming Notes:**

1. Logical addition and subtraction may be used to perform arithmetic on multiple-precision binary-integer operands. Thus, for multiple-precision addition, ADD LOGICAL can be used to add the corresponding parts of the operands beginning with the lowest-order parts. If the condition code indicates a carry, a value of one should be added to the sum of the next-higher-order parts. If the multiple-precision operands are signed, ADD should be used on the highest-order parts. The condition code then indicates any overflow or the proper sign and magnitude of the entire result; an overflow is also indicated by a program interruption for fixed-point overflow if allowed by the program mask. If the multiple-precision operands are unsigned, ADD LOGICAL should be used throughout.
2. Another use for ADD LOGICAL is to increment values representing binary counters, which are allowed to wrap around from all ones to all zeros without indicating overflow.

## Signed and Logical Comparison

Comparison operations determine whether two operands are equal or not and, for most operations, which of two unequal operands is the greater (high). Signed-binary-comparison operations are provided which treat the operands as signed binary integers, and logical-comparison operations are provided which treat the operands as unsigned binary integers or as unstructured data.

COMPARE and COMPARE HALFWORD are signed-binary-comparison operations. These instructions are equivalent to SUBTRACT and SUBTRACT HALFWORD without replacing either operand, the resulting difference being used only to set the condition code. The operations permit comparison of numbers of opposite sign which differ by $2^{31}$ or more. Thus, unlike SUBTRACT, COMPARE cannot cause overflow.



LzLabs GmbH. Ex. 1008-308

Logical comparison of two operands is performed byte by byte, in a left-to-right sequence. The operands are equal when all their bytes are equal. When the operands are unequal, the comparison result is determined by a left-to-right comparison of corresponding bit positions in the first unequal pair of bytes: the zero bit in the first unequal pair of bits indicates the low operand, and the one bit the high operand. Since the remaining bit and byte positions do not change the comparison, it is not necessary to continue comparing unequal operands beyond the first unequal bit pair.

## Instructions

The general instructions and their mnemonics, formats, and operation codes are listed in Figure 7-1 on page 7-7. The figure also indicates when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

A detailed definition of instruction formats, operand designation and length, and address generation is contained in “Instructions” on page 5-2. Exceptions to the general rules stated in that section are explicitly identified in the individual instruction descriptions.

**Note:** In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designations for the assembler language are shown with each instruction. For LOAD AND TEST, for example, LTR is the mnemonic and $R_1$,$R_2$ the operand designation.

**Programming Notes:**

1. Bimodal addressing affects the general instructions in the manner in which logical storage addresses are handled and in that, for the following instructions, the leftmost byte of the results in registers may be handled differently depending on whether the addressing mode is the 24-bit or the 31-bit mode:

   - BRANCH AND LINK (BAL, BALR)
   - BRANCH AND SAVE (BAS, BASR)
   - BRANCH AND SAVE AND SET MODE
   - BRANCH AND SET MODE
   - BRANCH RELATIVE AND SAVE
   - BRANCH RELATIVE AND SAVE LONG
   - CHECKSUM
   - COMPARE LOGICAL LONG
   - COMPARE LOGICAL LONG EXTENDED
   - COMPARE LOGICAL LONG UNICODE
   - COMPARE LOGICAL STRING
   - COMPARE UNTIL SUBSTRING EQUAL
   - CONVERT UNICODE TO UTF-8
   - CONVERT UTF-8 TO UNICODE
   - LOAD ADDRESS
   - LOAD ADDRESS EXTENDED
   - LOAD ADDRESS RELATIVE LONG
   - MOVE LONG
   - MOVE LONG EXTENDED
   - MOVE LONG UNICODE
   - MOVE STRING
   - SEARCH STRING
   - TRANSLATE AND TEST
   - TRANSLATE EXTENDED
   - TRANSLATE ONE TO ONE
   - TRANSLATE ONE TO TWO
   - TRANSLATE TWO TO ONE
   - TRANSLATE TWO TO TWO

   Otherwise, the general instructions are executed the same way in both the 24-bit and 31-bit addressing modes.

2. The following additional general instructions are available in ESA/370 and ESA/390 as compared to 370-XA:

   - COMPARE UNTIL SUBSTRING EQUAL
   - COPY ACCESS
   - EXTRACT ACCESS
   - LOAD ACCESS MULTIPLE
   - LOAD ADDRESS EXTENDED
   - MOVE PAGE
   - SET ACCESS
   - STORE ACCESS MULTIPLE

3. The MOVE INVERSE instruction, which is optional in 370-XA and ESA/390 and was described as being optional in ESA/390, is described in the eighth edition of this publication as being basic in ESA/390 (it is on all ESA/390 models).

4. The following additional general instructions are available when the string-instruction facility is installed:

   - COMPARE LOGICAL STRING
   - MOVE STRING
   - SEARCH STRING

5. The following additional general instructions are available when the compare-and-move-extended facility is installed:

   - COMPARE LOGICAL LONG EXTENDED



LzLabs GmbH. Ex. 1008-309

- MOVE LONG EXTENDED

6. The following additional general instructions are available when the immediate-and-relative-instruction facility is installed:
   - ADD HALFWORD IMMEDIATE
   - BRANCH RELATIVE AND SAVE
   - BRANCH RELATIVE ON CONDITION
   - BRANCH RELATIVE ON COUNT
   - BRANCH RELATIVE ON INDEX HIGH
   - BRANCH RELATIVE ON INDEX LOW OR EQUAL
   - COMPARE HALFWORD IMMEDIATE
   - LOAD HALFWORD IMMEDIATE
   - MULTIPLY SINGLE
   - MULTIPLY HALFWORD IMMEDIATE
   - TEST UNDER MASK HIGH
   - TEST UNDER MASK LOW
7. The general instruction CHECKSUM is available when the checksum facility is installed.
8. The general instruction PERFORM LOCKED OPERATION is available when the perform-locked-operation facility is installed.
9. The general instruction STORE CLOCK EXTENDED is available when the extended-TOD-clock facility is installed.
10. The general instructions CONVERT UNICODE TO UTF-8, CONVERT UTF-8 TO UNICODE, and TRANSLATE EXTENDED are available when the extended-translation facility 1 is installed.
11. The following additional general instructions are available in the ESA/390 architectural mode when the z/Architecture architectural mode is installed:
    - ADD LOGICAL WITH CARRY (ALC, ALCR)
    - BRANCH RELATIVE AND SAVE LONG
    - BRANCH RELATIVE ON CONDITION LONG
    - DIVIDE LOGICAL (DL, DLR)
    - EXTRACT PSW
    - LOAD ADDRESS RELATIVE LONG
    - LOAD REVERSED (LRV, LRVH, LRVR)
    - MULTIPLY LOGICAL (ML, MLR)
    - ROTATE LEFT SINGLE LOGICAL
    - SET ADDRESSING MODE (SAM24, SAM31)
    - STORE REVERSED (STRV, STRVH)
    - SUBTRACT LOGICAL WITH BORROW (SLB, SLBR)
    - TEST ADDRESSING MODE
12. The following additional general instructions are available when the extended-translation facility 2 is installed:
    - COMPARE LOGICAL LONG UNICODE
    - MOVE LONG UNICODE
    - PACK ASCII
    - PACK UNICODE
    - TRANSLATE ONE TO ONE
    - TRANSLATE ONE TO TWO
    - TRANSLATE TWO TO ONE
    - TRANSLATE TWO TO TWO
    - UNPACK ASCII
    - UNPACK UNICODE
13. The following additional general instructions are available when the message-security assist is installed:
    - CIPHER MESSAGE
    - CIPHER MESSAGE WITH CHAINING
    - COMPUTE INTERMEDIATE MESSAGE DIGEST
    - COMPUTE LAST MESSAGE DIGEST
    - COMPUTE MESSAGE AUTHENTICATION CODE



LzLabs GmbH. Ex. 1008-310

| Name | Mne-monic | Characteristics | | | | | | | Op Code |
|---|---|---|---|---|---|---|---|---|---|
| ADD | AR | RR C | | | IF | | R | | 1A |
| ADD | A | RX C | | A | IF | | R | $B_2$ | 5A |
| ADD HALFWORD | AH | RX C | | A | IF | | R | $B_2$ | 4A |
| ADD HALFWORD IMMEDIATE | AHI | RI C | IR | | IF | | R | | A7A |
| ADD LOGICAL | ALR | RR C | | | | | R | | 1E |
| ADD LOGICAL | AL | RX C | | A | | | R | $B_2$ | 5E |
| ADD LOGICAL WITH CARRY | ALCR | RRE C | N3 | | | | R | | B998 |
| ADD LOGICAL WITH CARRY | ALC | RXE C | N3 | A | | | R | $B_2$ | E398 |
| AND | NR | RR C | | | | | R | | 14 |
| AND | N | RX C | | A | | | R | $B_2$ | 54 |
| AND (character) | NC | SS C | | A | | | ST | $B_1$ $B_2$ | D4 |
| AND (immediate) | NI | SI C | | A | | | ST | $B_1$ | 94 |
| BRANCH AND LINK | BALR | RR | | | T | | B R | | 05 |
| BRANCH AND LINK | BAL | RX | | | | | B R | | 45 |
| BRANCH AND SAVE | BASR | RR | | | T | | B R | | 0D |
| BRANCH AND SAVE | BAS | RX | | | | | B R | | 4D |
| BRANCH AND SAVE AND SET MODE | BASSM | RR | | | T | | B R | | 0C |
| BRANCH AND SET MODE | BSM | RR | | | | | B R | | 0B |
| BRANCH ON CONDITION | BCR | RR | | | ¢[1] | | B | | 07 |
| BRANCH ON CONDITION | BC | RX | | | | | B | | 47 |
| BRANCH ON COUNT | BCTR | RR | | | | | B R | | 06 |
| BRANCH ON COUNT | BCT | RX | | | | | B R | | 46 |
| BRANCH ON INDEX HIGH | BXH | RS | | | | | B R | | 86 |
| BRANCH ON INDEX LOW OR EQUAL | BXLE | RS | | | | | B R | | 87 |
| BRANCH RELATIVE AND SAVE | BRAS | RI | IR | | | | B R | | A75 |
| BRANCH RELATIVE AND SAVE LONG | BRASL | RIL | N3 | | | | B R | | C05 |
| BRANCH RELATIVE ON CONDITION | BRC | RI | IR | | | | B | | A74 |
| BRANCH RELATIVE ON CONDITION LONG | BRCL | RIL | N3 | | | | B | | C04 |
| BRANCH RELATIVE ON COUNT | BRCT | RI | IR | | | | B R | | A76 |
| BRANCH RELATIVE ON INDEX HIGH | BRXH | RSI | IR | | | | B R | | 84 |
| BRANCH RELATIVE ON INDEX LOW OR EQ. | BRXLE | RSI | IR | | | | B R | | 85 |
| CHECKSUM | CKSM | RRE C | CK | A SP | | | R | $R_2$ | B241 |
| CIPHER MESSAGE | KM | RRE C | MS | A SP | | GM I1 | ST | $R_1$ $R_2$ | B92E |
| CIPHER MESSAGE WITH CHAINING | KMC | RRE C | MS | A SP | | GM I1 | ST | $R_1$ $R_2$ | B92F |
| COMPARE | CR | RR C | | | | | | | 19 |
| COMPARE | C | RX C | | A | | | | $B_2$ | 59 |
| COMPARE AND FORM CODEWORD | CFC | S C | | A SP | II | GM | R | I1 | B21A |
| COMPARE AND SWAP | CS | RS C | | A SP | $ | | R ST | $B_2$ | BA |
| COMPARE DOUBLE AND SWAP | CDS | RS C | | A SP | $ | | R ST | $B_2$ | BB |
| COMPARE HALFWORD | CH | RX C | | A | | | | $B_2$ | 49 |

*Figure 7-1 (Part 1 of 7). Summary of General Instructions*



LzLabs GmbH. Ex. 1008-311

| Name | Mne-monic | Characteristics | | | | | | Op Code |
|---|---|---|---|---|---|---|---|---|
| COMPARE HALFWORD IMMEDIATE | CHI | RI C IR | | | | | | A7E |
| COMPARE LOGICAL | CLR | RR C | | | | | | 15 |
| COMPARE LOGICAL | CL | RX C | A | | | | $B_2$ | 55 |
| COMPARE LOGICAL (character) | CLC | SS C | A | | | | $B_1$ $B_2$ | D5 |
| COMPARE LOGICAL (immediate) | CLI | SI C | A | | | | $B_1$ | 95 |
| COMPARE LOGICAL CHARS. UNDER MASK | CLM | RS C | A | | | | $B_2$ | BD |
| COMPARE LOGICAL LONG | CLCL | RR C | A SP | II | | R | $R_1$ $R_2$ | 0F |
| COMPARE LOGICAL LONG EXTENDED | CLCLE | RS C CM | A SP | | | R | $R_1$ $R_3$ | A9 |
| COMPARE LOGICAL LONG UNICODE | CLCLU | RSE C E2 | A SP | | | R | $R_1$ $R_2$ | EB8F |
| COMPARE LOGICAL STRING | CLST | RRE C SR | A SP | | G0 | R | $R_1$ $R_2$ | B25D |
| COMPARE UNTIL SUBSTRING EQUAL | CUSE | RRE C | A SP | II | GM | | $R_1$ $R_2$ | B257 |
| COMPUTE INTERMEDIATE MESSAGE DIGEST | KIMD | RRE C MS | A SP | | GM I1 | ST | $R_2$ | B93E |
| COMPUTE LAST MESSAGE DIGEST | KLMD | RRE C MS | A SP | | GM I1 | ST | $R_2$ | B93F |
| COMPUTE MESSAGE AUTHENTICATION CODE | KMAC | RRE C MS | A SP | | GM I1 | ST | $R_2$ | B91E |
| CONVERT TO BINARY | CVB | RX | A | Dd | IK | R | $B_2$ | 4F |
| CONVERT TO DECIMAL | CVD | RX | A | | | ST | $B_2$ | 4E |
| CONVERT UNICODE TO UTF-8 | CUUTF | RRE C ET | A SP | | | R ST | $R_1$ $R_2$ | B2A6 |
| CONVERT UTF-8 TO UNICODE | CUTFU | RRE C ET | A SP | | | R ST | $R_1$ $R_2$ | B2A7 |
| COPY ACCESS | CPYA | RRE | | | | | $U_1$ $U_2$ | B24D |
| DIVIDE | DR | RR | SP | | IK | R | | 1D |
| DIVIDE | D | RX | A SP | | IK | R | $B_2$ | 5D |
| DIVIDE LOGICAL | DLR | RRE N3 | SP | | IK | R | | B997 |
| DIVIDE LOGICAL | DL | RXE N3 | A SP | | IK | R | $B_2$ | E397 |
| EXCLUSIVE OR | XR | RR C | | | | R | | 17 |
| EXCLUSIVE OR | X | RX C | A | | | R | $B_2$ | 57 |
| EXCLUSIVE OR (character) | XC | SS C | A | | | ST | $B_1$ $B_2$ | D7 |
| EXCLUSIVE OR (immediate) | XI | SI C | A | | | ST | $B_1$ | 97 |
| EXECUTE | EX | RX | AI SP | | EX | | | 44 |
| EXTRACT ACCESS | EAR | RRE | | | | R | $U_2$ | B24F |
| EXTRACT PSW | EPSW | RRE N3 | | | | R | | B98D |
| INSERT CHARACTER | IC | RX | A | | | R | $B_2$ | 43 |
| INSERT CHARACTERS UNDER MASK | ICM | RS C | A | | | R | $B_2$ | BF |
| INSERT PROGRAM MASK | IPM | RRE | | | | R | | B222 |
| LOAD | LR | RR | | | | R | | 18 |
| LOAD | L | RX | A | | | R | $B_2$ | 58 |
| LOAD ACCESS MULTIPLE | LAM | RS | A SP | | | | UB | 9A |
| LOAD ADDRESS | LA | RX | | | | R | | 41 |
| LOAD ADDRESS EXTENDED | LAE | RX | | | | R | $U_1$ BP | 51 |
| LOAD ADDRESS RELATIVE LONG | LARL | RIL N3 | | | | R | | C00 |
| LOAD AND TEST | LTR | RR C | | | | R | | 12 |

*Figure 7-1 (Part 2 of 7). Summary of General Instructions*



LzLabs GmbH. Ex. 1008-312

| Name | Mne-monic | Characteristics | | | | | Op Code |
|---|---|---|---|---|---|---|---|
| LOAD COMPLEMENT | LCR | RR C | | IF | R | | 13 |
| LOAD HALFWORD | LH | RX | A | | R | $B_2$ | 48 |
| LOAD HALFWORD IMMEDIATE | LHI | RI IR | | | R | | A78 |
| LOAD MULTIPLE | LM | RS | A | | R | $B_2$ | 98 |
| LOAD NEGATIVE | LNR | RR C | | | R | | 11 |
| LOAD POSITIVE | LPR | RR C | | IF | R | | 10 |
| LOAD REVERSED | LRVR | RRE N3 | | | R | | B91F |
| LOAD REVERSED | LRVH | RXE N3 | A | | R | $B_2$ | E31F |
| LOAD REVERSED | LRV | RXE N3 | A | | R | $B_2$ | E31E |
| MONITOR CALL | MC | SI | SP | MO | | | AF |
| MOVE (character) | MVC | SS | A | | ST | $B_1$ $B_2$ | D2 |
| MOVE (immediate) | MVI | SI | A | | ST | $B_1$ | 92 |
| MOVE INVERSE | MVCIN | SS | A | | ST | $B_1$ $B_2$ | E8 |
| MOVE LONG | MVCL | RR C | A SP | II | R ST | $R_1$ $R_2$ | 0E |
| MOVE LONG EXTENDED | MVCLE | RS C CM | A SP | | R ST | $R_1$ $R_3$ | A8 |
| MOVE LONG UNICODE | MVCLU | RSE C E2 | A SP | | R ST | $R_1$ $R_2$ | EB8E |
| MOVE NUMERICS | MVN | SS | A | | ST | $B_1$ $B_2$ | D1 |
| MOVE PAGE (facility 1) | MVPG | RRE C M1 | A[1] SP | G0 | ST | $R_1$ $R_2$ | B254 |
| MOVE STRING | MVST | RRE C SR | A SP | G0 | R ST | $R_1$ $R_2$ | B255 |
| MOVE WITH OFFSET | MVO | SS | A | | ST | $B_1$ $B_2$ | F1 |
| MOVE ZONES | MVZ | SS | A | | ST | $B_1$ $B_2$ | D3 |
| MULTIPLY | MR | RR | SP | | R | | 1C |
| MULTIPLY | M | RX | A SP | | R | $B_2$ | 5C |
| MULTIPLY HALFWORD | MH | RX | A | | R | $B_2$ | 4C |
| MULTIPLY HALFWORD IMMEDIATE | MHI | RI IR | | | R | | A7C |
| MULTIPLY LOGICAL | MLR | RRE N3 | SP | | R | | B996 |
| MULTIPLY LOGICAL | ML | RXE N3 | A SP | | R | $B_2$ | E396 |
| MULTIPLY SINGLE | MSR | RRE IR | | | R | | B252 |
| MULTIPLY SINGLE | MS | RX IR | A | | R | $B_2$ | 71 |
| OR | OR | RR C | | | R | | 16 |
| OR | O | RX C | A | | R | $B_2$ | 56 |
| OR (character) | OC | SS C | A | | ST | $B_1$ $B_2$ | D6 |
| OR (immediate) | OI | SI C | A | | ST | $B_1$ | 96 |
| PACK | PACK | SS | A | | ST | $B_1$ $B_2$ | F2 |
| PACK ASCII | PKA | SS E2 | A SP | | ST | $B_1$ $B_2$ | E9 |
| PACK UNICODE | PKU | SS E2 | A SP | | ST | $B_1$ $B_2$ | E1 |
| PERFORM LOCKED OPERATION | PLO | SS C PL | A SP | $ GM | R ST | FC | EE |
| ROTATE LEFT SINGLE LOGICAL | RLL | RSE N3 | | | R | | EB1D |
| SEARCH STRING | SRST | RRE C SR | A SP | G0 | R | $R_2$ | B25E |
| SET PROGRAM MASK | SPM | RR L | | | | | 04 |

*Figure 7-1 (Part 3 of 7). Summary of General Instructions*



LzLabs GmbH. Ex. 1008-313

| Name | Mne-monic | Characteristics | | | | | Op Code |
|---|---|---|---|---|---|---|---|
| SHIFT LEFT DOUBLE | SLDA | RS C | SP | IF | R | | 8F |
| SHIFT LEFT DOUBLE LOGICAL | SLDL | RS | SP | | R | | 8D |
| SET ACCESS | SAR | RRE | | | | $U_1$ | B24E |
| SET ADDRESSING MODE | SAM24 | E N3 | SP | T | | | 010C |
| SET ADDRESSING MODE | SAM31 | E N3 | SP | T | | | 010D |
| SHIFT LEFT SINGLE | SLA | RS C | | IF | R | | 8B |
| SHIFT LEFT SINGLE LOGICAL | SLL | RS | | | R | | 89 |
| SHIFT RIGHT DOUBLE | SRDA | RS C | SP | | R | | 8E |
| SHIFT RIGHT DOUBLE LOGICAL | SRDL | RS | SP | | R | | 8C |
| SHIFT RIGHT SINGLE | SRA | RS C | | | R | | 8A |
| SHIFT RIGHT SINGLE LOGICAL | SRL | RS | | | R | | 88 |
| STORE | ST | RX | A | | ST | $B_2$ | 50 |
| STORE ACCESS MULTIPLE | STAM | RS | A SP | | ST | UB | 9B |
| STORE CHARACTER | STC | RX | A | | ST | $B_2$ | 42 |
| STORE CHARACTERS UNDER MASK | STCM | RS | A | | ST | $B_2$ | BE |
| STORE CLOCK | STCK | S C | A | $ | ST | $B_2$ | B205 |
| STORE CLOCK EXTENDED | STCKE | S C EK | A | $ | ST | $B_2$ | B278 |
| STORE HALFWORD | STH | RX | A | | ST | $B_2$ | 40 |
| STORE MULTIPLE | STM | RS | A | | ST | $B_2$ | 90 |
| STORE REVERSED | STRVH | RXE N3 | A | | ST | $B_2$ | E33F |
| STORE REVERSED | STRV | RXE N3 | A | | ST | $B_2$ | E33E |
| SUBTRACT | SR | RR C | | IF | R | | 1B |
| SUBTRACT | S | RX C | A | IF | R | $B_2$ | 5B |
| SUBTRACT HALFWORD | SH | RX C | A | IF | R | $B_2$ | 4B |
| SUBTRACT LOGICAL | SLR | RR C | | | R | | 1F |
| SUBTRACT LOGICAL | SL | RX C | A | | R | $B_2$ | 5F |
| SUBTRACT LOGICAL WITH BORROW | SLBR | RRE C N3 | | | R | | B999 |
| SUBTRACT LOGICAL WITH BORROW | SLB | RXE C N3 | A | | R | $B_2$ | E399 |
| SUPERVISOR CALL | SVC | RR | | ¢ | | | 0A |
| TEST ADDRESSING MODE | TAM | E C N3 | | | | | 010B |
| TEST AND SET | TS | S C | A | $ | ST | $B_2$ | 93 |
| TEST UNDER MASK | TM | SI C | A | | | $B_1$ | 91 |
| TEST UNDER MASK HIGH | TMH | RI C IR | | | | | A70 |
| TEST UNDER MASK LOW | TML | RI C IR | | | | | A71 |
| TRANSLATE | TR | SS | A | | ST | $B_1$ $B_2$ | DC |
| TRANSLATE AND TEST | TRT | SS C | A | GM | R | $B_1$ $B_2$ | DD |
| TRANSLATE EXTENDED | TRE | RRE C ET | A SP | | R ST | $R_1$ $R_2$ | B2A5 |
| TRANSLATE ONE TO ONE | TROO | RRE C E2 | A SP | GM | R ST | RM $R_2$ | B993 |
| TRANSLATE ONE TO TWO | TROT | RRE C E2 | A SP | GM | R ST | RM $R_2$ | B992 |
| TRANSLATE TWO TO ONE | TRTO | RRE C E2 | A SP | GM | R ST | RM $R_2$ | B991 |
| TRANSLATE TWO TO TWO | TRTT | RRE C E2 | A SP | GM | R ST | RM $R_2$ | B990 |
| UNPACK | UNPK | SS | A | | ST | $B_1$ $B_2$ | F3 |
| UNPACK ASCII | UNPKA | SS C E2 | A SP | | ST | $B_1$ $B_2$ | EA |
| UNPACK UNICODE | UNPKU | SS C E2 | A SP | | ST | $B_1$ $B_2$ | E2 |
| UPDATE TREE | UPT | E C | A SP | II GM | R ST | I4 | 0102 |

*Figure 7-1 (Part 4 of 7). Summary of General Instructions*

*Figure 7-1 (Part 5 of 7). Summary of General Instructions*



LzLabs GmbH. Ex. 1008-314

**Explanation:**

| | |
|---|---|
| ¢ | Causes serialization and checkpoint synchronization. |
| ¢1 | Causes serialization and checkpoint synchronization when the $M_1$ and $R_2$ fields contain all ones and all zeros, respectively. |
| $ | Causes serialization. |
| A | Access exceptions for logical addresses. |
| A1 | Access exceptions; not all access exceptions may occur; see instruction description for details. |
| AI | Access exceptions for instruction address. |
| B | PER branch event. |
| $B_1$ | $B_1$ field designates an access register in the access-register mode. |
| $B_2$ | $B_2$ field designates an access register in the access-register mode. |
| BP | $B_2$ field designates an access register when PSW bits 16 and 17 have the value 01. |
| C | Condition code is set. |
| CK | Checksum facility. |
| CM | Compare-and-move-extended facility. |
| Dd | Decimal-operand data exception. |
| E | E instruction format. |
| E2 | Extended-translation facility 2. |
| EK | Extended-TOD-clock facility. |
| ET | Extended-translation facility 1. |
| EX | Execute exception. |
| FC | Designation of access registers depends on the function code of the instruction. |
| G0 | Instruction execution includes the implied use of general register 0. |
| GM | Instruction execution includes the implied use of multiple general registers:<br>General registers 0 and 1 for TRANSLATE ONE TO ONE, TRANSLATE ONE TO TWO, TRANSLATE TWO TO ONE, and TRANSLATE TWO TO TWO.<br>General registers 1 and 2 for TRANSLATE AND TEST.<br>General registers 1, 2, and 3 for COMPARE AND FORM CODEWORD.<br>General registers 0 and 1 for COMPARE UNTIL SUBSTRING EQUAL and PERFORM LOCKED OPERATION.<br>General registers 0-5 for UPDATE TREE. |
| IF | Fixed-point-overflow exception. |
| II | Interruptible instruction. |
| IK | Fixed-point-divide exception. |
| IR | Immediate-and-relative-instruction facility. |
| I1 | Access register 1 is implicitly designated in the access-register mode. |
| I4 | Access register 4 is implicitly designated in the access-register mode. |
| L | New condition code is loaded. |
| MO | Monitor event. |
| MS | Message-security assist. |
| M1 | Move-page facility 1, which is a subset of move-page facility 2. |
| N3 | Instruction is added to ESA/390 from z/Architecture. |
| PL | Perform-locked-operation facility. |
| R | PER general-register-alteration event. |
| $R_1$ | $R_1$ field designates an access register in the access-register mode. |
| $R_2$ | $R_2$ field designates an access register in the access-register mode. |

*Figure 7-1 (Part 6 of 7). Summary of General Instructions*



LzLabs GmbH. Ex. 1008-315

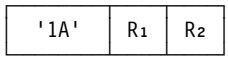
**Explanation** (Continued):

| | |
|---|---|
| $R_3$ | $R_3$ field designates an access register in the access-register mode. |
| RI | RI instruction format. |
| RM | $R_1$ field designates an access register in the access-register mode, and access-register 1 also is used in the access-register mode. |
| RR | RR instruction format. |
| RRE | RRE instruction format. |
| RS | RS instruction format. |
| RSE | RSE instruction format. |
| RSI | RSI instruction format. |
| RX | RX instruction format. |
| S | S instruction format. |
| SI | SI instruction format. |
| SP | Specification exception. |
| SR | String-instruction facility. |
| SS | SS instruction format. |
| ST | PER storage-alteration event. |
| T | Trace exceptions (includes trace table, addressing, and low-address protection). |
| $U_1$ | $R_1$ field designates an access register unconditionally. |
| $U_2$ | $R_2$ field designates an access register unconditionally. |
| UB | $R_1$ and $R_3$ fields designate access registers unconditionally, and $B_2$ field designates an access register in the access-register mode. |

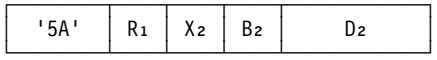
*Figure 7-1 (Part 7 of 7). Summary of General Instructions*

## ADD

AR $R_1,R_2$ [RR]




A $R_1,D_2(X_2,B_2)$ [RX]




The second operand is added to the first operand, and the sum is placed at the first-operand location. The operands and the sum are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

***Resulting Condition Code:***

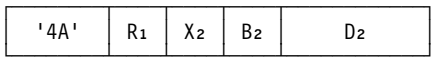
0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

***Program Exceptions:***

- Access (fetch, operand 2 of A only)
- Fixed-point overflow

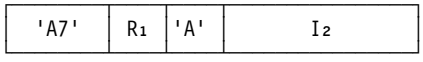
## ADD HALFWORD

AH $R_1,D_2(X_2,B_2)$ [RX]




## ADD HALFWORD IMMEDIATE

AHI $R_1,I_2$ [RI]




The second operand is added to the first operand, and the sum is placed at the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand and the sum are treated as 32-bit signed binary integers.



LzLabs GmbH. Ex. 1008-316

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

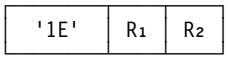
***Program Exceptions:***

- Access (fetch, operand 2 of AH only)
- Fixed-point overflow
- Operation (AHI if the immediate-and-relative-instruction facility is not installed)

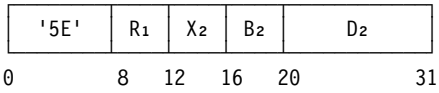
**Programming Note:** An example of the use of the ADD HALFWORD instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

## ADD LOGICAL

ALR R1,R2 [RR]




AL R1,D2(X2,B2) [RX]




The second operand is added to the first operand, and the sum is placed at the first-operand location. The operands and the sum are treated as 32-bit unsigned binary integers.

***Resulting Condition Code:***

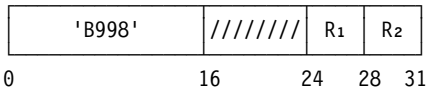
0 Result zero; no carry
1 Result not zero; no carry
2 Result zero; carry
3 Result not zero; carry

***Program Exceptions:***

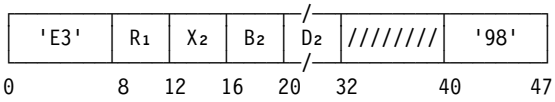
- Access (fetch, operand 2 of AL only)

## ADD LOGICAL WITH CARRY

ALCR R1,R2 [RRE]




ALC R1,D2(X2,B2) [RXE]




The second operand and the carry are added to the first operand, and the sum is placed at the first-operand location. The operands, the carry, and the sum are treated as 32-bit unsigned binary integers.

***Resulting Condition Code:***

0 Result zero; no carry
1 Result not zero; no carry
2 Result zero; carry
3 Result not zero; carry

***Program Exceptions:***

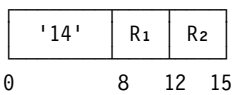
- Access (fetch, operand 2 of ALC only)
- Operation (if z/Architecture is not installed)

**Programming Notes:**

1. A carry is represented by a one value of bit 18 of the current PSW. Bit 18 is the leftmost bit of the two-bit condition code in the PSW. Bit 18 is set to one by an execution of an ADD LOGICAL or ADD LOGICAL WITH CARRY instruction that produces a carry out of bit position 0 of the result.
2. ADD and ADD LOGICAL may provide better performance than ADD LOGICAL WITH CARRY, depending on the model.

## AND

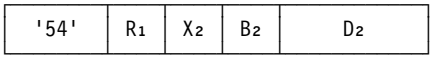
NR R1,R2 [RR]



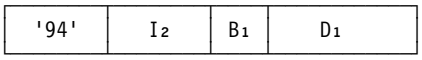


LzLabs GmbH. Ex. 1008-317

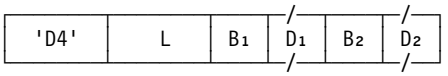
```
N      R1,D2(X2,B2)      [RX]
```




```
NI     D1(B1),I2         [SI]
```




```
NC     D1(L,B1),D2(B2)          [SS]
```




The AND of the first and second operands is placed at the first-operand location.

The connective AND is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the corresponding bit positions in both operands contain ones; otherwise, the result bit is set to zero.

For AND (NC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

For AND (NI), the first operand is one byte in length, and only one byte is stored.

***Resulting Condition Code:***

0 Result zero
1 Result not zero
2 --
3 --

***Program Exceptions:***

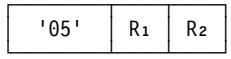
- Access (fetch, operand 2, N and NC; fetch and store, operand 1, NI and NC)

**Programming Notes:**

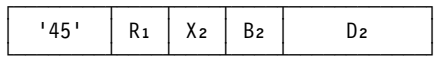
1. An example of the use of the AND instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The AND instruction may be used to set a bit to zero.
3. Accesses to the first operand of AND (NI) and AND (NC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, the instruction AND cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in "Multiprogramming and Multiprocessing Examples" on page A-43.

## BRANCH AND LINK

```
BALR   R1,R2      [RR]
```



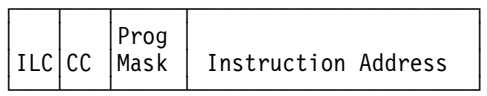

```
BAL    R1,D2(X2,B2)      [RX]
```




Information from the current PSW, including the updated instruction address, is loaded as link information at the first-operand location. Subsequently, the instruction address is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of general register $R_2$ are used to generate the branch address; however, when the $R_2$ field is zero, the operation is performed without branching. The branch address is computed before general register $R_1$ is changed.

The link information in the 24-bit addressing mode consists of the instruction-length code (ILC), the condition code (CC), the program-mask bits, and the rightmost 24 bits of the updated instruction address, arranged in the following format:

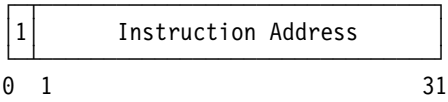



The instruction-length code is 1 or 2.



LzLabs GmbH. Ex. 1008-318

The link information in the 31-bit addressing mode consists of the right half of the PSW, that is, the addressing-mode bit (always a one) and a 31-bit updated instruction address, arranged in the following format:




***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

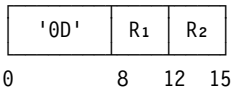
- Trace ($R_2$ field nonzero, BALR only)

**Programming Notes:**

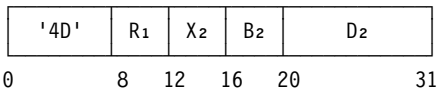
1. An example of the use of the BRANCH AND LINK instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When the $R_2$ field in the RR format is zero, the link information is loaded without branching.
3. The BRANCH AND LINK instruction (BAL and BALR) is provided for compatibility purposes. It is recommended that, where possible, the BRANCH AND SAVE instruction (BAS and BASR) or BRANCH RELATIVE AND SAVE be used and BRANCH AND LINK avoided, since the latter places nonzero information in bit positions 0-7 of the link register in the 24-bit addressing mode, which may lead to problems. Additionally, BRANCH AND LINK may be slower than BRANCH AND SAVE and BRANCH RELATIVE AND SAVE because the latter instructions always save the right half of the PSW, and BRANCH AND LINK, which does not, may require additional time to test the addressing mode, and even more time, if the 24-bit addressing mode is in effect, to construct the ILC, condition code, and program mask to be placed in the leftmost byte of the link register.
4. The condition-code and program-mask information, which is provided in the leftmost byte of the link information only in the 24-bit addressing mode, can be obtained in both the 24-bit and 31-bit addressing modes by means of the INSERT PROGRAM MASK instruction.

## BRANCH AND SAVE

BASR $R_1,R_2$ [RR]




BAS $R_1,D_2(X_2,B_2)$ [RX]




Bits 32-63 of the current PSW, including the updated instruction address, are saved as link information at the first-operand location. Subsequently, the instruction address is replaced by the branch address.

In the 24-bit addressing mode, the link information consists of a 24-bit instruction address with eight zeros appended on the left. In the 31-bit addressing mode, the link information consists of a 31-bit address with a one appended on the left.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of general register $R_2$ are used to generate the branch address; however, when the $R_2$ field is zero, the operation is performed without branching. The branch address is computed before general register $R_1$ is changed.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Trace ($R_2$ field nonzero, BASR only)

**Programming Notes:**

1. An example of the use of the BRANCH AND SAVE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The BRANCH AND SAVE instruction (BAS and BASR) is intended to be used for linkage to programs known to be in the same addressing mode as the caller. This instruction should be used in place of the BRANCH AND LINK instruction (BAL and BALR). See the programming note on page 5-14 at the end of "Subroutine Linkage without the Linkage Stack" for a detailed discussion of this


LzLabs GmbH. Ex. 1008-319

and other linkage instructions. See also the programming note under BRANCH AND LINK for a discussion of the advantages of the BRANCH AND SAVE instruction.

## BRANCH AND SAVE AND SET MODE

```
BASSM  R1,R2      [RR]
```




Bits 32-63 of the current PSW, including the updated instruction address, are saved as link information at the first-operand location. Subsequently, the addressing mode and instruction address in the current PSW are replaced from the second operand. The action associated with the second operand is not performed if the $R_2$ field is zero.

In the 24-bit addressing mode, the link information consists of a 24-bit instruction address with eight zeros appended on the left. In the 31-bit addressing mode, the link information consists of a 31-bit address with a one appended on the left.

The contents of general register $R_2$ specify the new addressing mode and designate the branch address; however, when the $R_2$ field is zero, the operation is performed without branching and without setting the addressing mode.

When the contents of general register $R_2$ are used, bit 0 of the register specifies the new addressing mode and replaces bit 32 of the current PSW, and the branch address is generated from the contents of the register under the control of the new addressing mode. The new value for the PSW is computed before general register $R_1$ is changed.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Trace ($R_2$ field nonzero)

**Programming Notes:**

1. An example of the use of the BRANCH AND SAVE AND SET MODE instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. BRANCH AND SAVE AND SET MODE is intended to be the principal calling instruction to subroutines which may operate in a different addressing mode from that of the caller. See the programming note on page 5-14 at the end of “Subroutine Linkage without the Linkage Stack” for a detailed discussion of this and other linkage instructions.

## BRANCH AND SET MODE

```
BSM    R1,R2      [RR]
```




Bit 32 of the current PSW, the addressing mode, is inserted into the first operand. Subsequently, the addressing mode and instruction address in the current PSW are replaced from the second operand. The action associated with an operand is not performed if the associated R field is zero.

The value of bit 32 of the PSW is placed in bit position 0 of general register $R_1$, and bits 1-31 of the register remain unchanged; however, when the $R_1$ field is zero, the bit is not inserted, and the contents of general register 0 are not changed.

The contents of general register $R_2$ specify the new addressing mode and designate the branch address; however, when the $R_2$ field is zero, the operation is performed without branching and without setting the addressing mode.

When the contents of general register $R_2$ are used, bit 0 of the register specifies the new addressing mode and replaces bit 32 of the current PSW, and the branch address is generated from the contents of the register under the control of the new addressing mode. The new value for the PSW is computed before general register $R_1$ is changed.

***Condition Code:*** The code remains unchanged.



LzLabs GmbH. Ex. 1008-320

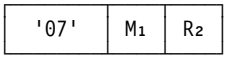
***Program Exceptions:*** None.

**Programming Notes:**

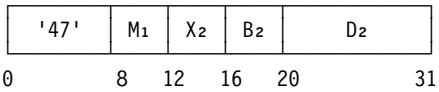
1. An example of the use of the BRANCH AND SET MODE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. BRANCH AND SET MODE with an $R_1$ field of zero is intended to be the standard return instruction. BRANCH AND SET MODE with a nonzero $R_1$ field is intended to be used in a "glue module" to connect old 24-bit programs and new programs which may exploit bimodal addressing. See the programming note at the end of "Subroutine Linkage without the Linkage Stack" in Chapter 5, "Program Execution" for a detailed discussion of this and other linkage instructions.

## BRANCH ON CONDITION

BCR $M_1,R_2$ [RR]




BC $M_1,D_2(X_2,B_2)$ [RX]




The instruction address in the current PSW is replaced by the branch address if the condition code has one of the values specified by $M_1$; otherwise, normal instruction sequencing proceeds with the updated instruction address.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of general register $R_2$ are used to generate the branch address; however, when the $R_2$ field is zero, the operation is performed without branching.

The $M_1$ field is used as a four-bit mask. The four condition codes (0, 1, 2, and 3) correspond, left to right, with the four bits of the mask, as follows:

| Condition Code | Instruction Bit No. of Mask | Mask Position Value |
|---|---|---|
| 0 | 8 | 8 |
| 1 | 9 | 4 |
| 2 | 10 | 2 |
| 3 | 11 | 1 |

The current condition code is used to select the corresponding mask bit. If the mask bit selected by the condition code is one, the branch is successful. If the mask bit selected is zero, normal instruction sequencing proceeds with the next sequential instruction.

When the $M_1$ and $R_2$ fields of BRANCH ON CONDITION (BCR) are all ones and all zeros, respectively, a serialization and checkpoint-synchronization function is performed.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:*** None.

**Programming Notes:**

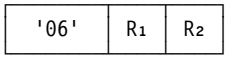
1. An example of the use of the BRANCH ON CONDITION instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When a branch is to depend on more than one condition, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made when the condition code is 0 or 1.
3. When all four mask bits are zeros or when the $R_2$ field in the RR format contains zero, the branch instruction is equivalent to a no-operation. When all four mask bits are ones, that is, the mask value is 15, the branch is unconditional unless the $R_2$ field in the RR format is zero.
4. Execution of BCR 15,0 (that is, an instruction with a value of 07F0 hex) may result in significant performance degradation. To ensure optimum performance, the program should avoid use of BCR 15,0 except in cases when the serialization or checkpoint-synchronization function is actually required.
5. Note that the relation between the RR and RX formats in branch-address specification is not


LzLabs GmbH. Ex. 1008-321

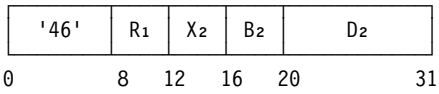
the same as in operand-address specification. For branch instructions in the RX format, the branch address is the address specified by $X_2$, $B_2$, and $D_2$; in the RR format, the branch address is contained in the register designated by $R_2$. For operands, the address specified by $X_2$, $B_2$, and $D_2$ is the operand address, but the register designated by $R_2$ contains the operand, not the operand address.

## BRANCH ON COUNT

```
BCTR   R1,R2      [RR]
```




```
BCT    R1,D2(X2,B2)      [RX]
```




A one is subtracted from the first operand, and the result is placed at the first-operand location. The first operand and result are treated as 32-bit binary integers, with overflow ignored. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address in the current PSW is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of general register $R_2$ are used to generate the branch address; however, when the $R_2$ field is zero, the operation is performed without branching. The branch address is computed before general register $R_1$ is changed.

***Condition Code:*** The code remains unchanged.

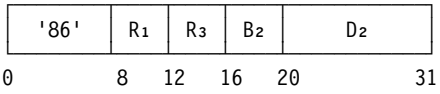
***Program Exceptions:*** None.

**Programming Notes:**

1. An example of the use of the BRANCH ON COUNT instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. The first operand and result can be considered as either signed or unsigned binary integers since the result of a binary subtraction is the same in both cases.
3. An initial count of one results in zero, and no branching takes place; an initial count of zero results in -1 and causes branching to be performed; an initial count of -1 results in -2 and causes branching to be performed; and so on. In a loop, branching takes place each time the instruction is executed until the result is again zero. Note that, because of the number range, an initial count of $-2^{31}$ results in a positive value of $2^{31} - 1$.
4. Counting is performed without branching when the $R_2$ field in the RR format contains zero.

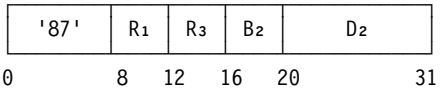
## BRANCH ON INDEX HIGH

```
BXH    R1,R3,D2(B2)      [RS]
```




## BRANCH ON INDEX LOW OR EQUAL

```
BXLE   R1,R3,D2(B2)      [RS]
```




An increment is added to the first operand, and the sum is compared with a compare value. The result of the comparison determines whether branching occurs. Subsequently, the sum is placed at the first-operand location. The second-operand address is used as a branch address. The $R_3$ field designates registers containing the increment and the compare value.

For BRANCH ON INDEX HIGH, when the sum is high, the instruction address in the current PSW is replaced by the branch address. When the sum is low or equal, normal instruction sequencing proceeds with the updated instruction address.

For BRANCH ON INDEX LOW OR EQUAL, when the sum is low or equal, the instruction address in the current PSW is replaced by the branch address. When the sum is high, normal instruc-



LzLabs GmbH. Ex. 1008-322

tion sequencing proceeds with the updated instruction address.

When the $R_3$ field is even, it designates a pair of registers; the contents of the even and odd registers of the pair are used as the increment and the compare value, respectively. When the $R_3$ field is odd, it designates a single register, the contents of which are used as both the increment and the compare value.

For purposes of the addition and comparison, all operands and results are treated as 32-bit signed binary integers. Overflow caused by the addition is ignored.

The original contents of the compare-value register are used as the compare value even when that register is also specified to be the first-operand location. The branch address is computed before general register $R_1$ is changed.

The sum is placed at the first-operand location, regardless of whether the branch is taken.

***Condition Code:*** The code remains unchanged.

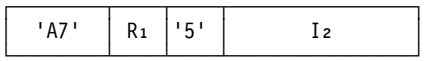
***Program Exceptions:*** None.

**Programming Notes:**

1. Several examples of the use of the BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL instructions are given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. The word “index” in the names of these instructions indicates that one of the major purposes is the incrementing and testing of an index value. The increment, being a signed binary integer, may be used to increase or decrease the value in general register $R_1$ by an arbitrary amount.
3. Care must be taken in the 31-bit addressing mode when a data area in storage is at the rightmost end of an address space and a BRANCH ON INDEX HIGH or BRANCH ON INDEX LOW OR EQUAL instruction is used to step upward through the data. Since the addition and comparison operations performed during the execution of these instructions treat the operands as 32-bit signed binary integers, the value following $2^{31} - 1$ is not $2^{31}$, which cannot be represented in that format, but $-2^{31}$. The instruction does not provide an indication of such overflow. Consequently, some common looping techniques based on the use of these instructions do not work when a data area ends at address $2^{31} - 1$. This problem is illustrated in a BRANCH ON INDEX LOW OR EQUAL example in Appendix A, “Number Representation and Instruction-Use Examples.”

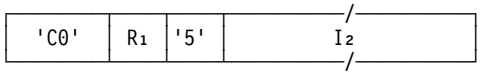
## BRANCH RELATIVE AND SAVE

```
BRAS     R1,I2     [RI]
```




## BRANCH RELATIVE AND SAVE LONG

```
BRASL    R1,I2     [RIL]
```




Bits 32-63 of the current PSW, including the updated instruction address, are saved as link information at the first-operand location. Subsequently, the instruction address is replaced by the branch address.

In the 24-bit addressing mode, the link information consists of a 24-bit instruction address with eight zeros appended on the left. In the 31-bit addressing mode, the link information consists of a 31-bit address with a one appended on the left.

The contents of the $I_2$ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Operation (if the immediate-and-relative-instruction facility is not installed, BRAS only; if z/Architecture is not installed, BRASL only)

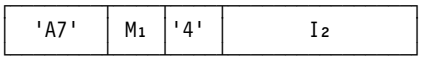

LzLabs GmbH. Ex. 1008-323

**Programming Notes:**

1. The operation is the same as that of the BRANCH AND SAVE (BAS) instruction except for the means of specifying the branch address. An example of the use of BRANCH AND SAVE is given in Appendix A.
2. The BRANCH RELATIVE AND SAVE and BRANCH RELATIVE AND SAVE LONG instructions, like the BRANCH AND SAVE instruction, are intended to be used for linkage to programs known to be in the same addressing mode as the caller. These instructions should be used in place of the BRANCH AND LINK instruction (BAL and BALR). See the programming note on page 5-14 at the end of "Subroutine Linkage without the Linkage Stack" for a detailed discussion of these and other linkage instructions. See also the programming note under BRANCH AND LINK for a discussion of the advantages of the BRANCH RELATIVE AND SAVE, BRANCH RELATIVE AND SAVE LONG, and BRANCH AND SAVE instructions.
3. When the instruction is the target of EXECUTE, the branch is relative to the target address; see "Branch-Address Generation" on page 5-9.

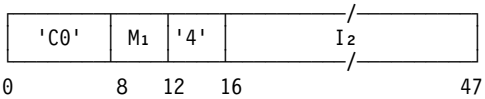
## BRANCH RELATIVE ON CONDITION

```
BRC     M1,I2     [RI]
```




## BRANCH RELATIVE ON CONDITION LONG

```
BRCL    M1,I2     [RIL]
```




The instruction address in the current PSW is replaced by the branch address if the condition code has one of the values specified by $M_1$; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The contents of the $I_2$ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

The $M_1$ field is used as a four-bit mask. The four condition codes (0, 1, 2, and 3) correspond, left to right, with the four bits of the mask, as follows:

| Condition Code | Instruction Bit No. of Mask | Mask Position Value |
|---|---|---|
| 0 | 8 | 8 |
| 1 | 9 | 4 |
| 2 | 10 | 2 |
| 3 | 11 | 1 |

The current condition code is used to select the corresponding mask bit. If the mask bit selected by the condition code is one, the branch is successful. If the mask bit selected is zero, normal instruction sequencing proceeds with the next sequential instruction.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Operation (if the immediate-and-relative-instruction facility is not installed, BRC only; if z/Architecture is not installed, BRCL only)

**Programming Notes:**

1. The operation is the same as that of the BRANCH ON CONDITION instruction except for the means of specifying the branch address. An example of the use of BRANCH ON CONDITION is given in Appendix A.
2. When a branch is to depend on more than one condition, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made when the condition code is 0 or 1.
3. When all four mask bits are zeros, the branch instruction is equivalent to a no-operation. When all four mask bits are ones, that is, the mask value is 15, the branch is unconditional.
4. When the instruction is the target of EXECUTE, the branch is relative to the target

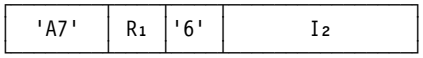


LzLabs GmbH. Ex. 1008-324

address; see “Branch-Address Generation” on page 5-9.

## BRANCH RELATIVE ON COUNT

```
BRCT     R1,I2     [RI]
```




A one is subtracted from the first operand, and the result is placed at the first-operand location. The first operand and result are treated as 32-bit binary integers, with overflow ignored. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address in the current PSW is replaced by the branch address.

The contents of the $I_2$ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

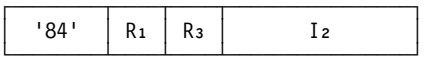
- Operation (if the immediate-and-relative-instruction facility is not installed)

**Programming Notes:**

1. The operation is the same as that of the BRANCH ON COUNT instruction except for the means of specifying the branch address. An example of the use of BRANCH ON COUNT is given in Appendix A.
2. The first operand and result can be considered as either signed or unsigned binary integers since the result of a binary subtraction is the same in both cases.
3. An initial count of one results in zero, and no branching takes place; an initial count of zero results in -1 and causes branching to be executed; an initial count of -1 results in -2 and causes branching to be executed; and so on. In a loop, branching takes place each time the instruction is executed until the result is again zero. Note that, because of the number range, an initial count of $-2^{31}$ results in a positive value of $2^{31} - 1$.
4. When the instruction is the target of EXECUTE, the branch is relative to the target address; see “Branch-Address Generation” on page 5-9.

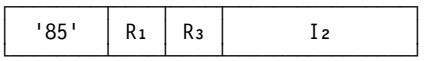
## BRANCH RELATIVE ON INDEX HIGH

```
BRXH     R1,R3,I2     [RSI]
```




## BRANCH RELATIVE ON INDEX LOW OR EQUAL

```
BRXLE    R1,R3,I2     [RSI]
```




An increment is added to the first operand, and the sum is compared with a compare value. The result of the comparison determines whether branching occurs. Subsequently, the sum is placed at the first-operand location. The $R_3$ field designates registers containing the increment and the compare value.

The contents of the $I_2$ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

For BRANCH RELATIVE ON INDEX HIGH, when the sum is high, the instruction address in the current PSW is replaced by the branch address. When the sum is low or equal, normal instruction sequencing proceeds with the updated instruction address.

For BRANCH RELATIVE ON INDEX LOW OR EQUAL, when the sum is low or equal, the instruction address in the current PSW is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address.

When the $R_3$ field is even, it designates a pair of registers; the contents of the even and odd registers of the pair are used as the increment and the


LzLabs GmbH. Ex. 1008-325

compare value, respectively. When the $R_3$ field is odd, it designates a single register, the contents of which are used as both the increment and the compare value.

For purposes of the addition and comparison, all operands and results are treated as 32-bit signed binary integers. Overflow caused by the addition is ignored.

The original contents of the compare-value register are used as the compare value even when that register is also specified to be the first-operand location.

The sum is placed at the first-operand location, regardless of whether the branch is taken.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

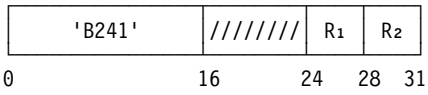
- Operation (if the immediate-and-relative-instruction facility is not installed)

**Programming Notes:**

1. The operations are the same as those of the BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL instructions except for the means of specifying the branch address. Several examples of the use of BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL are given in Appendix A.
2. The word "index" in the names of these instructions indicates that one of the major purposes is the incrementing and testing of an index value. The increment, being a signed binary integer, may be used to increase or decrease the value in general register $R_1$ by an arbitrary amount.
3. Care must be taken in the 31-bit addressing mode when a data area in storage is at the rightmost end of an address space and a BRANCH RELATIVE ON INDEX HIGH or BRANCH RELATIVE ON INDEX LOW OR EQUAL instruction is used to step upward through the data. Since the addition and comparison operations performed during the execution of these instructions treat the operands as 32-bit signed binary integers, the value following $2^{31} - 1$ is not $2^{31}$, which cannot be represented in that format, but $-2^{31}$. The instruction does not provide an indication of such overflow. Consequently, some common looping techniques based on the use of these instructions do not work when a data area ends at address $2^{31} - 1$. This problem is illustrated in a BRANCH ON INDEX LOW OR EQUAL example in Appendix A.
4. When the instruction is the target of EXECUTE, the branch is relative to the target address; see "Branch-Address Generation" on page 5-9.

## CHECKSUM

```
CKSM     R1,R2     [RRE]
```




Successive four-byte elements of the second operand are added to the first operand in general register $R_1$ to form a 32-bit checksum in the register. The first operand and the four-byte elements are treated as 32-bit unsigned binary integers. After each addition of an element, a carry out of bit position 0 of the first operand is added to bit position 31 of the first operand. If the second operand is not a multiple of four bytes, its last one, two, or three bytes are treated as appended on the right with the number of all-zeros bytes needed to form a four-byte element. The four-byte elements are added to the first operand until either the entire second operand or a CPU-determined amount of the second operand has been processed. The result is indicated in the condition code.

The $R_2$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the second operand is specified by the contents of the $R_2$ general register. The number of bytes in the second-operand location is specified by the 32-bit unsigned binary integer in the $R_2 + 1$ general register.

The handling of the address in general register $R_2$ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit posi-



LzLabs GmbH. Ex. 1008-326

tions 8-31 of general register $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general register $R_2$ constitute the address, and the contents of bit position 0 are ignored.

The addition of second-operand four-byte elements to the first operand proceeds left to right, four-byte element by four-byte element, and ends as soon as (1) the entire second operand has been processed or (2) a lesser CPU-determined amount of the second operand has been processed. In either case, the result in general register $R_1$ is a 32-bit checksum for the part of the second operand that has been processed. When the second operand is not a multiple of four bytes, the final second-operand bytes in excess of a multiple of four are conceptually appended on the right with an appropriate number of all-zeros bytes to form the final four-byte element.

If the operation ends because the entire second operand has been processed, the condition code is set to 0. If the operation ends because a lesser CPU-determined amount of the second operand has been processed, the condition code is set to 3. When the operation is to end with a setting of condition code 3, any carry out of bit position 0 of the first operand is added to bit position 31 of the first operand before the operation ends.

At the completion of the operation, the operand-length field in the $R_2 + 1$ register is decremented by the number of actual second-operand bytes added to the first operand (not including any conceptually appended all-zeros bytes), and the address in the $R_2$ register is incremented by the same number. Thus, the $R_2 + 1$ register contains a zero value if the condition code is set to 0, or it contains a nonzero value if the condition code is set to 3.

When condition code 3 is set, the general registers used by the instruction have been set so that the remainder of the second operand can be processed by simply branching back to reexecute the instruction.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed. The minimum amount is four bytes or the number of bytes specified in the $R_2 + 1$ general register, whichever is smaller.

At the completion of the operation, the leftmost bits which are not part of the address in general register $R_2$ may be set to zeros or may remain unchanged, even when the initial length in register $R_2 + 1$ is zero.

When the $R_1$ register is the same register as the $R_2$ or $R_2 + 1$ register, the results are unpredictable.

Access exceptions for the portion of the second operand to the right of the last byte processed may or may not be recognized. For a second operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

Access exceptions are not recognized if the $R_2$ field is odd. When the length of the second operand is zero, no access exceptions are recognized.

***Resulting Condition Code:***

0 Entire second operand processed
1 --
2 --
3 CPU-determined amount of second operand processed

***Program Exceptions:***

- Access (fetch, operand 2)
- Operation (if the checksum facility is not installed)
- Specification

**Programming Notes:**

1. The initial contents of the $R_1$ general register contribute to the 32-bit checksum. The program normally should set those contents to all zeros before issuing the CHECKSUM instruction.
2. A 16-bit checksum is used in, for example, the TCP/IP application. The following program can be executed after the CHECKSUM instruction to produce in general register $R_2$ a 16-bit checksum from the 32-bit checksum in general register $R_1$. The program is annotated to show the contents of the $R_2$ and $R_2 + 1$ registers after the execution of each


LzLabs GmbH. Ex. 1008-327

instruction. The contents of the $R_1$ register are represented as A,B, meaning the value A in bit positions 0-15 and the value B in bit positions 16-31. The value C is a carry from A + B. Note that register $R_2$ + 1 is known to contain all zeros when CHECKSUM has set condition code 0.

| Program | | Contents of R2 | Contents of R2+1 |
|---|---|---|---|
| LR | R2,R1 | A,B | 0,0 |
| SRDL | R2,16 | 0,A | B,0 |
| ALR | R2,R2+1 | B,A | B,0 |
| ALR | R2,R1 | A+B+C,A+B | B,0 |
| SRL | R2,16 | 0,A+B+C | B,0 |

3. The CHECKSUM instruction may be used in computing hash values as illustrated in the following programming example. The variable KEY contains a string to be mapped into a slot in a hash table. The variable SIZE is a prime number designating the size of the hash table. The value of SIZE is determined by (a) the number of strings to be hashed into the table divided by the acceptable number of hash collisions, and (b) a value that is not too close to a power of two. Following the DIVIDE (D) instruction, the remainder in register 0 represents the resulting hash value.

```
        SR    1,1      Zero accumulator
        LA    2,KEY    Point to string
        LA    3,L'KEY  Load string length
LOOP    CKSM  1,2      Compute checksum
        BNZ   LOOP     Repeat if not done
        SR    0,0      Zero for divide
        D     0,SIZE   Compute hash value
        ...
KEY     DS    CL64     String to be hashed
SIZE    DS    F        Size of hash table
```

4. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
5. The storage-operand references of CHECKSUM may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)
6. Figure 7-2 on page 7-25 contains a summary of the operation.



LzLabs GmbH. Ex. 1008-328

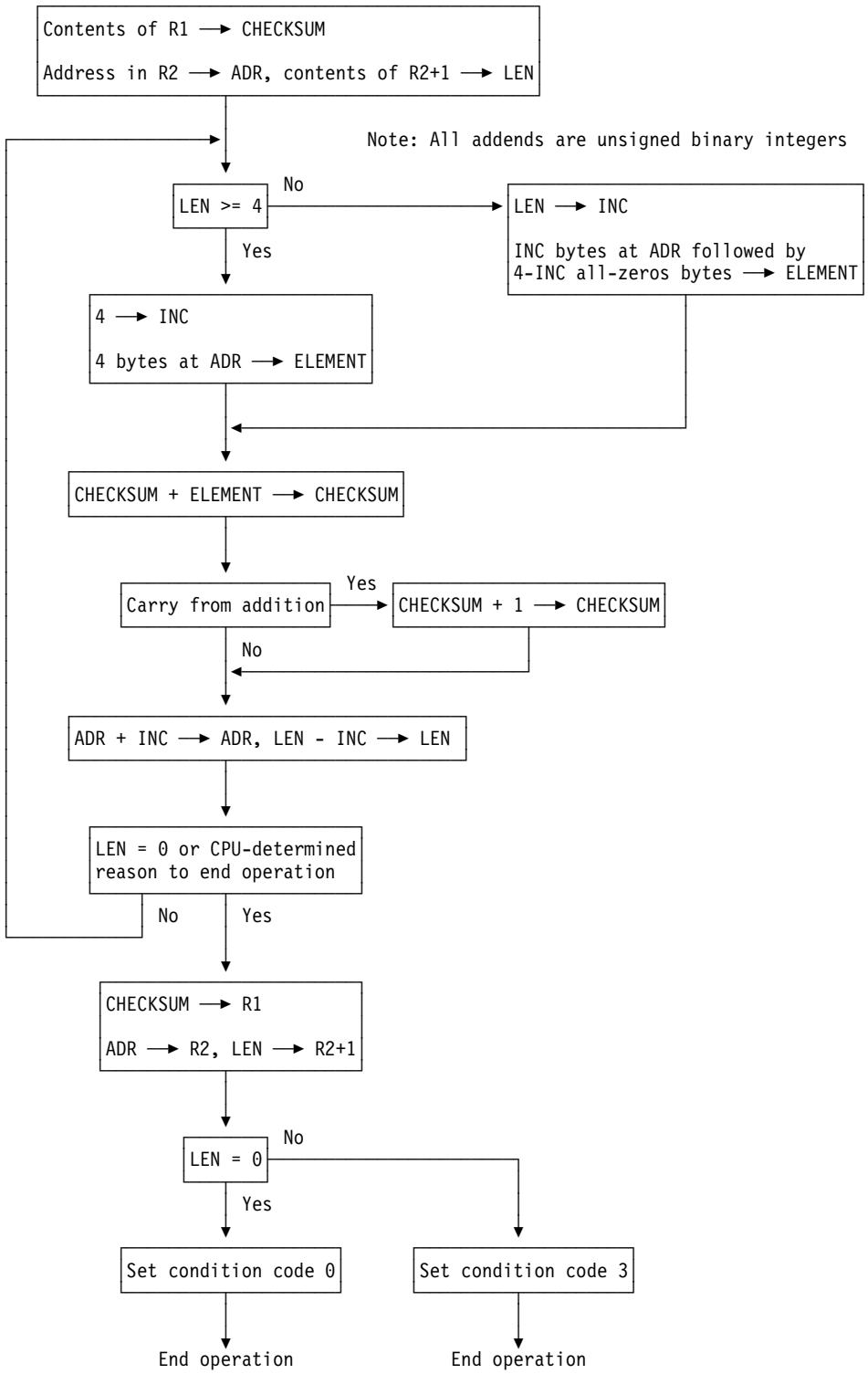


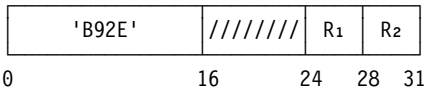

*Figure 7-2. Execution of CHECKSUM*



LzLabs GmbH. Ex. 1008-329

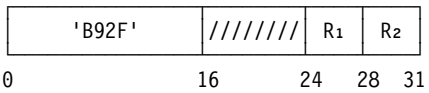
## CIPHER MESSAGE (KM)

```
KM       R1,R2      [RRE]
```




## CIPHER MESSAGE WITH CHAINING (KMC)

```
KMC      R1,R2      [RRE]
```




A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction are ignored.

Bit positions 25-31 of general register 0 contain the function code. Figures 7-3 and 7-4 show the assigned function codes for CIPHER MESSAGE and CIPHER MESSAGE WITH CHAINING, respectively. All other function codes are unassigned. For cipher functions, bit 24 is the modifier bit which specifies whether an encryption or a decryption operation is to be performed. The modifier bit is ignored for all other functions. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 8-31 of general register 1 constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general register 1 constitute the address, and the content of bit position 0 is ignored.

The function codes for CIPHER MESSAGE are as follows.

*Figure 7-3. Function Codes for CIPHER MESSAGE*

| Code | Function | Parm. Block Size (bytes) | Data Block Size (bytes) |
|---|---|---|---|
| 0 | KM-Query | 16 | — |
| 1 | KM-DEA | 8 | 8 |
| 2 | KM-TDEA-128 | 16 | 8 |
| 3 | KM-TDEA-192 | 24 | 8 |

**Explanation:**

— Not applicable

The function codes for CIPHER MESSAGE WITH CHAINING are as follows.

*Figure 7-4. Function Codes for CIPHER MESSAGE WITH CHAINING*

| Code | Function | Parm. Block Size (bytes) | Data Block Size (bytes) |
|---|---|---|---|
| 0 | KMC-Query | 16 | — |
| 1 | KMC-DEA | 16 | 8 |
| 2 | KMC-TDEA-128 | 24 | 8 |
| 3 | KMC-TDEA-192 | 32 | 8 |

**Explanation:**

— Not applicable

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers $R_1$, $R_2$, and $R_1 + 1$ are ignored for the query function.

For all other functions, the second operand is ciphered as specified by the function code using a cryptographic key in the parameter block, and the result is placed in the first-operand location. For CIPHER MESSAGE WITH CHAINING, ciphering also uses an initial chaining value in the parameter block, and the chaining value is updated as part of the operation.



LzLabs GmbH. Ex. 1008-330

The $R_1$ field designates a general register and must designate an even-numbered register; otherwise, a specification exception is recognized.

The $R_2$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first and second operands is specified by the contents of the $R_1$ and $R_2$ general registers, respectively. The number of bytes in the second-operand location is specified in general register $R_2 + 1$. The first operand is the same length as the second operand.

As part of the operation, the addresses in general registers $R_1$ and $R_2$ are incremented by the number of bytes processed, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the addresses and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the addresses of the first and second operands, respectively, and the contents of bit positions 0-7 are ignored; bits 8-31 of the updated addresses replace the corresponding bits in general registers $R_1$ and $R_2$, carries out of bit position 8 of the updated address are ignored, and the contents of bit positions 0-7 of general registers $R_1$ and $R_2$ are set to zeros. In the 31-bit addressing mode, the contents of bit positions 1-31 of general registers $R_1$ and $R_2$ constitute the addresses of the first and second operands, respectively, and the content of bit position 0 is ignored; bits 1-31 of the updated addresses replace the corresponding bits in general registers $R_1$ and $R_2$, carries out of bit position 1 of the updated address are ignored, and the content of bit position 0 of general registers $R_1$ and $R_2$ is set to zero.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 0-31 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which specifies the number of bytes in the first and second operands; and the updated value replaces the contents of bit positions 0-31 of general register $R_2 + 1$.

Figure 7-5 shows the contents of the general registers just described.

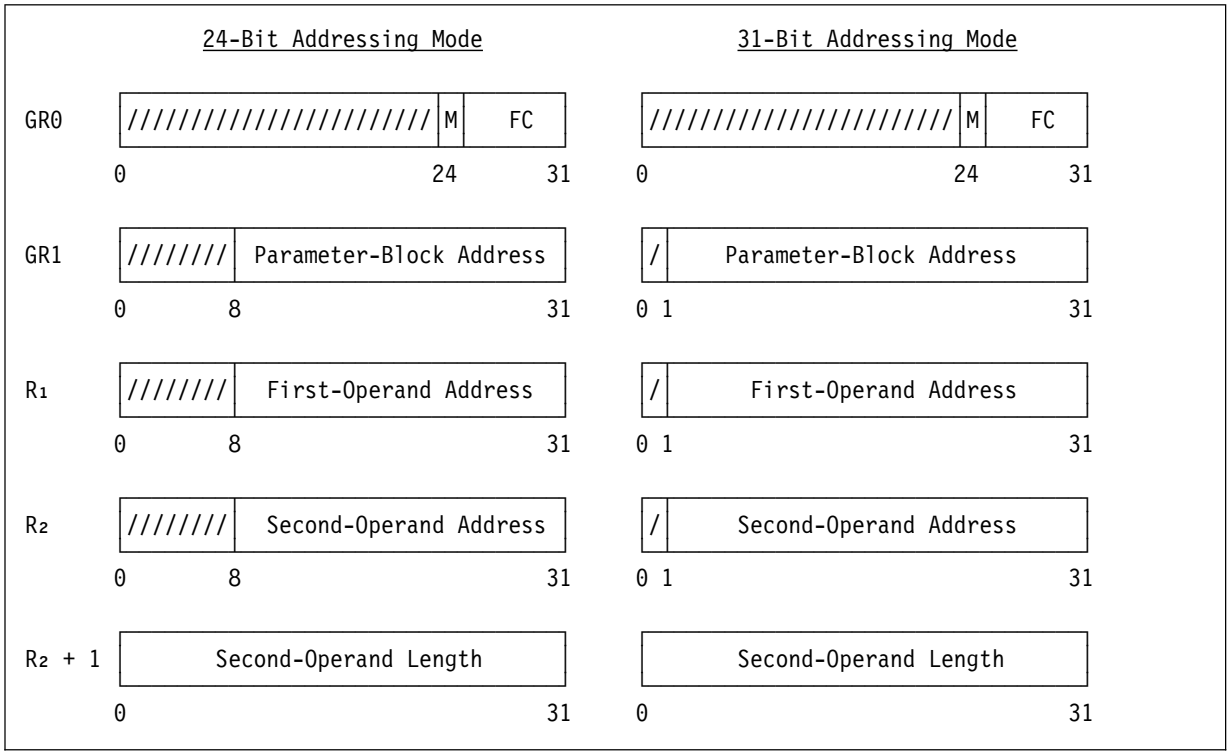



*Figure 7-5. General Register Assignment for KM and KMC*



LzLabs GmbH. Ex. 1008-331

In the access-register mode, access registers 1, $R_1$, and $R_2$ specify the address spaces containing the parameter block, first, and second operands, respectively.

The result is obtained as if processing starts at the left end of both the first and second operands and proceeds to the right, block by block. The operation is ended when the number of bytes in the second operand as specified in general register $R_2 + 1$ have been processed and placed at the first-operand location (called normal completion) or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually nonzero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless reoccurrence of this no-progress case.

The results in the first-operand location and the chaining-value field are unpredictable if any of the following situations occur:

1. The cryptographic-key field overlaps any portion of the first operand.
2. The chaining-value field overlaps any portion of the first operand or the second operand.
3. The first and second operands overlap destructively. Operands are said to overlap destructively when the first-operand location would be used as a source after data would have been moved into it, assuming processing to be performed from left to right and one byte at a time.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in $R_2 + 1$ is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in $R_2 + 1$ is nonzero.

When a storage-alteration PER event is recognized, fewer than 4K additional bytes are stored into the first-operand locations before the event is reported.

When the second-operand length is initially zero, the parameter block, first, and second operands are not accessed, general registers $R_1$, $R_2$, and $R_2 + 1$ are not changed, and condition code 0 is set.

When the contents of the $R_1$ and $R_2$ fields are the same, the contents of the designated registers are incremented only by the number of bytes processed, not by twice the number of bytes processed.

As observed by other CPUs and channel programs, references to the parameter block and storage operands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

In certain unusual situations, instruction execution may complete by setting condition code 3 without updating the registers and chaining value to reflect the last unit of the first and second operands processed. The size of the unit processed in this case depends on the situation and the model, but is limited such that the portion of the first and second operands which have been processed and not reported do not overlap in storage. In all cases, change bits are set and PER storage-alteration events are reported, when applicable, for all first-operand locations processed.

Access exceptions may be reported for a larger portion of an operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of an operand nor for locations more than 4K bytes beyond the current location being processed.

**Symbols Used in Function Descriptions**

The following symbols are used in the subsequent description of the CIPHER MESSAGE and CIPHER MESSAGE WITH CHAINING functions. For data-encryption-algorithm (DEA) functions, the DEA-key-parity bit in each byte of the DEA key is ignored, and the operation proceeds normally, regardless of the DEA-key parity of the key. Further description of the data-encryption algorithm may be found in *Data Encryption Algorithm*, ANSI-X3.92.1981, American National Standard for Information Systems.



LzLabs GmbH. Ex. 1008-332

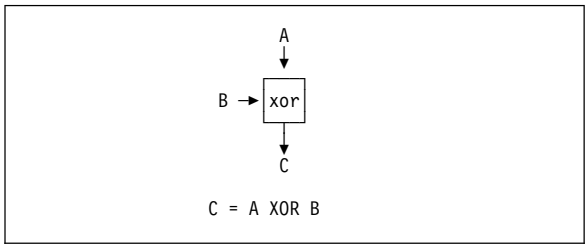



*Figure 7-6. Symbol For Bit-Wise Exclusive Or*

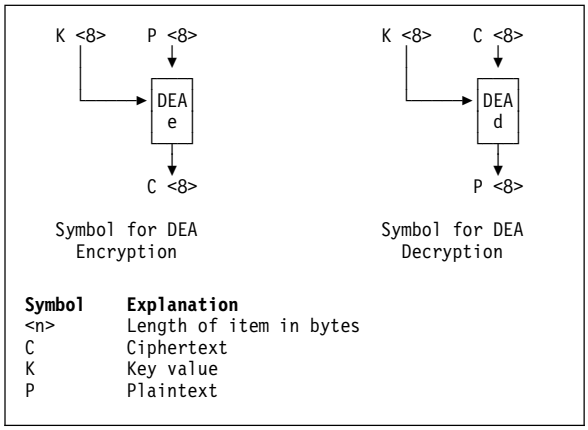



*Figure 7-7. Symbols for DEA Encryption and Decryption*

### KM-Query (KM Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-5 on page 7-27.

The parameter block used for the function has the following format:

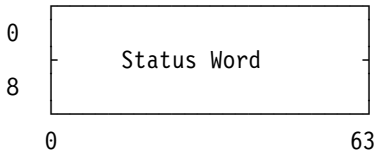



*Figure 7-8. Parameter Block for KM-Query*

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the CIPHER MESSAGE instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KM-Query function completes; condition code 3 is not applicable to this function.

### KM-DEA (KM Function Code 1)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-5 on page 7-27.

The parameter block used for the function has the following format:

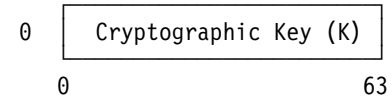



*Figure 7-9. Parameter Block for KM-DEA*

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the DEA algorithm with the 64-bit cryptographic key in the parameter block. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in the following figure:

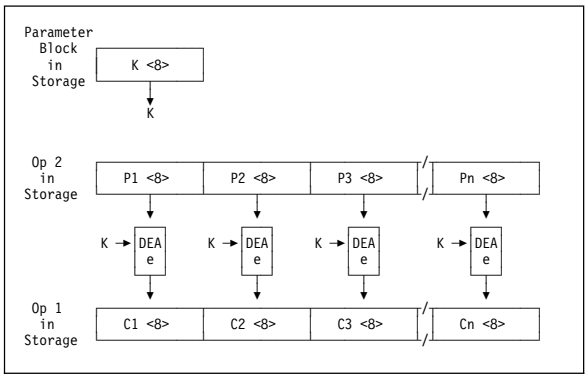



*Figure 7-10. KM-DEA Encipher Operation*

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the DEA algorithm with the 64-bit cryptographic key in the parameter block. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in the following figure:


LzLabs GmbH. Ex. 1008-333

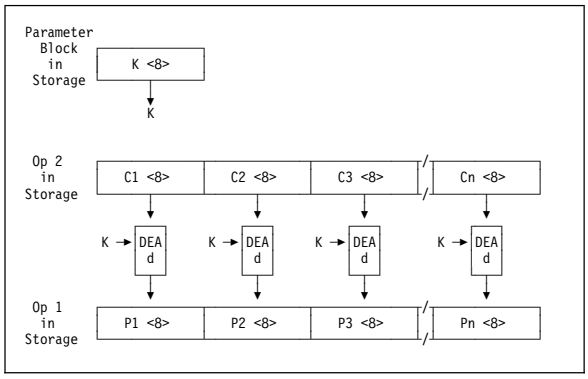



*Figure 7-11. KM-DEA Decipher Operation*

### KM-TDEA-128 (KM Function Code 2)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-5 on page 7-27.

The parameter block used for the function has the following format:

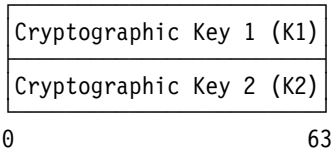



*Figure 7-12. Parameter Block for KM-TDEA-128*

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA (triple DEA) algorithm with the two 64-bit cryptographic keys in the parameter block. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in the following figure:

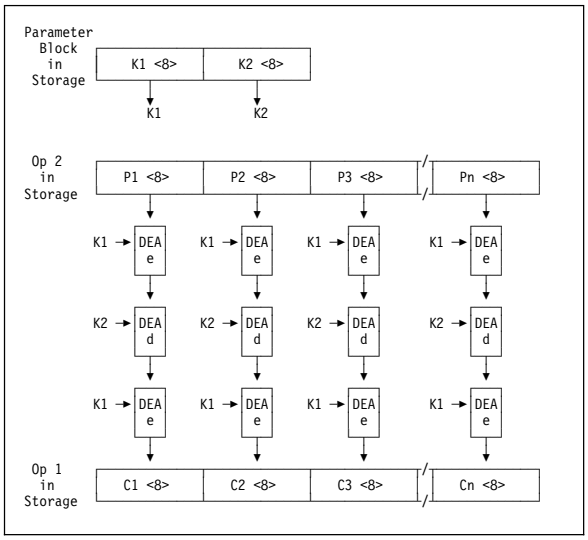



*Figure 7-13. KM-TDEA-128 Encipher Operation*

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA algorithm with the two 64-bit cryptographic keys in the parameter block. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in the following figure:

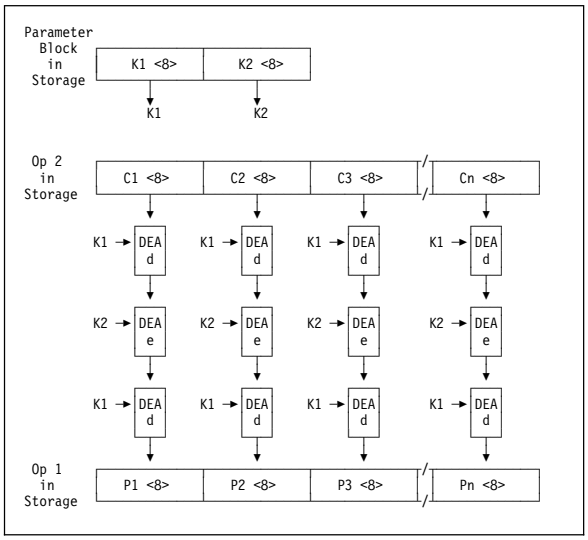



*Figure 7-14. KM-TDEA-128 Decipher Operation*

### KM-TDEA-192 (KM Function Code 3)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-5 on page 7-27.



LzLabs GmbH. Ex. 1008-334

The parameter block used for the function has the following format:

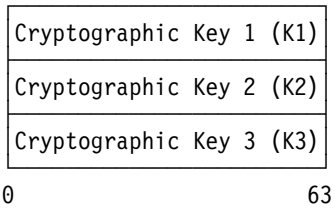



*Figure 7-15. Parameter Block for KM-TDEA-192*

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA algorithm with the three 64-bit cryptographic keys in the parameter block. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in the following figure:

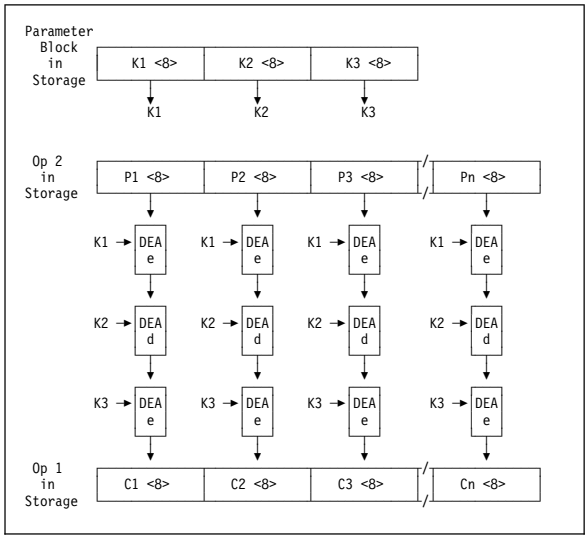



*Figure 7-16. KM-TDEA-192 Encipher Operation*

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA algorithm with the three 64-bit cryptographic keys in the parameter block. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in the following figure:

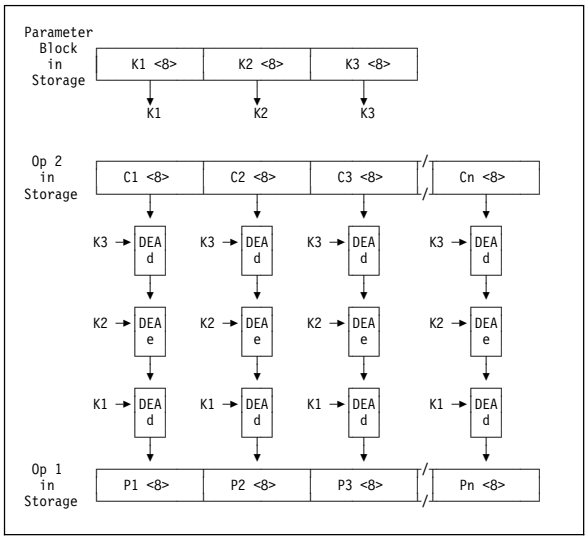



*Figure 7-17. KM-TDEA-192 Decipher Operation*

### KMC-Query (KMC Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-5 on page 7-27.

The parameter block used for the function has the following format:

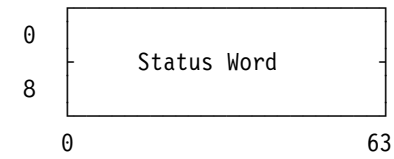



*Figure 7-18. Parameter Block for KMC-Query*

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the CIPHER MESSAGE WITH CHAINING instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KMC-Query function completes; condition code 3 is not applicable to this function.

### KMC-DEA (KMC Function Code 1)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-5 on page 7-27.

The parameter block used for the function has the following format:


LzLabs GmbH. Ex. 1008-335

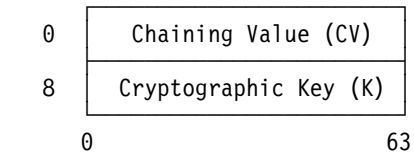



*Figure 7-19. Parameter Block for KMC-DEA*

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the DEA algorithm with the 64-bit cryptographic key and the 64-bit chaining value in the parameter block.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous ciphertext block. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field of the parameter block. The operation is shown in the following figure:

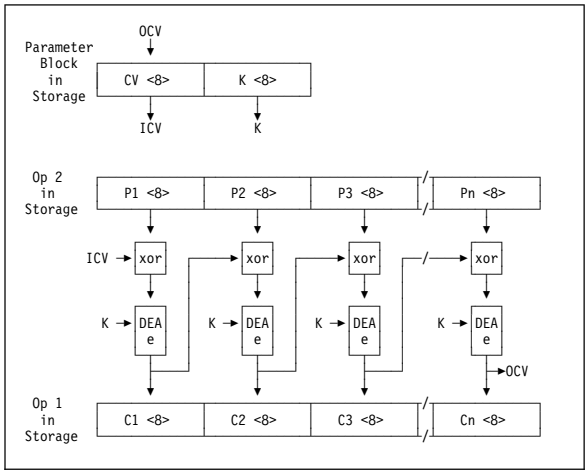



*Figure 7-20. KMC-DEA Encipher Operation*

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the DEA algorithm with the 64-bit cryptographic key and the 64-bit chaining value in the parameter block.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field in the parameter block. The operation is shown in the following figure:

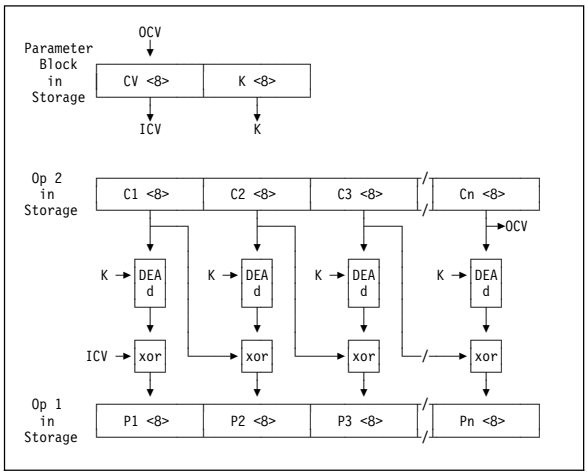



*Figure 7-21. KMC-DEA Decipher Operation*

**KMC-TDEA-128 (KMC Function Code 2)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-5 on page 7-27.

The parameter block used for the function has the following format:

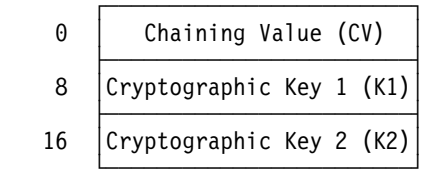



*Figure 7-22. Parameter Block for KMC-TDEA-128*

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA algorithm with the two 64-bit cryptographic keys and the 64-bit chaining value in the parameter block.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous



LzLabs GmbH. Ex. 1008-336

ciphertext block. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field of the parameter block. The operation is shown in the following figure:

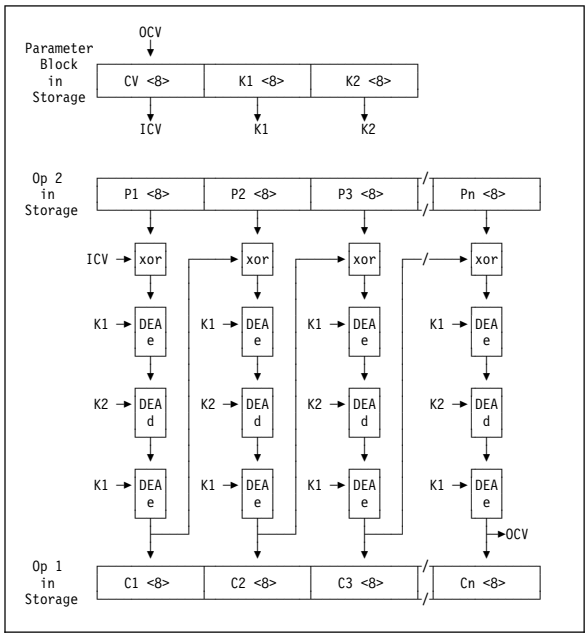



*Figure 7-23. KMC-TDEA-128 Encipher Operation*

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA algorithm with the two 64-bit cryptographic keys and the 64-bit chaining value in the parameter block.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field in the parameter block. The operation is shown in the following figure:

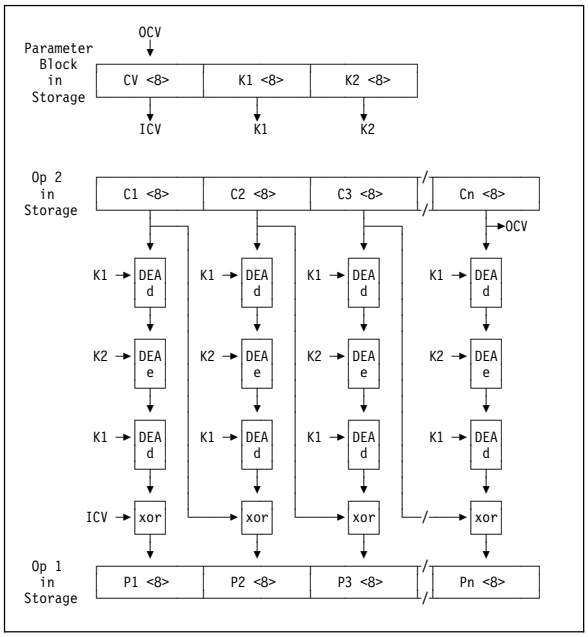



*Figure 7-24. KMC-TDEA-128 Decipher Operation*

### KMC-TDEA-192 (KMC Function Code 3)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-5 on page 7-27.

The parameter block used for the function has the following format:

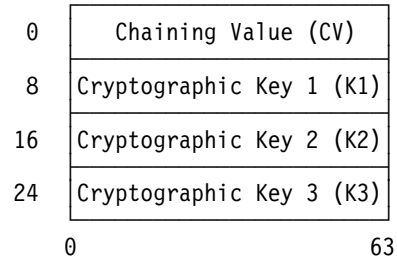



*Figure 7-25. Parameter Block for KMC-TDEA-192*

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA algorithm with the three 64-bit cryptographic keys and the 64-bit chaining value in the parameter block.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous


LzLabs GmbH. Ex. 1008-337

ciphertext block. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field of the parameter block. The operation is shown in the following figure:

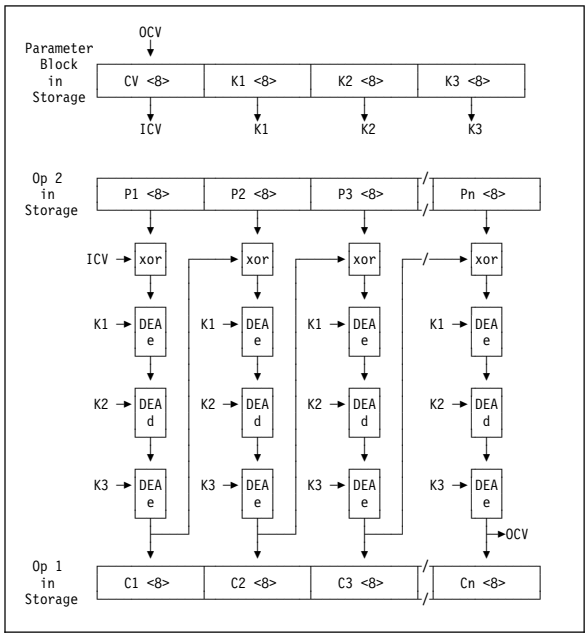



*Figure 7-26. KMC-TDEA-192 Encipher Operation*

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA algorithm with the three 64-bit cryptographic keys and the 64-bit chaining value in the parameter block.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field in the parameter block. The operation is shown in the following figure:

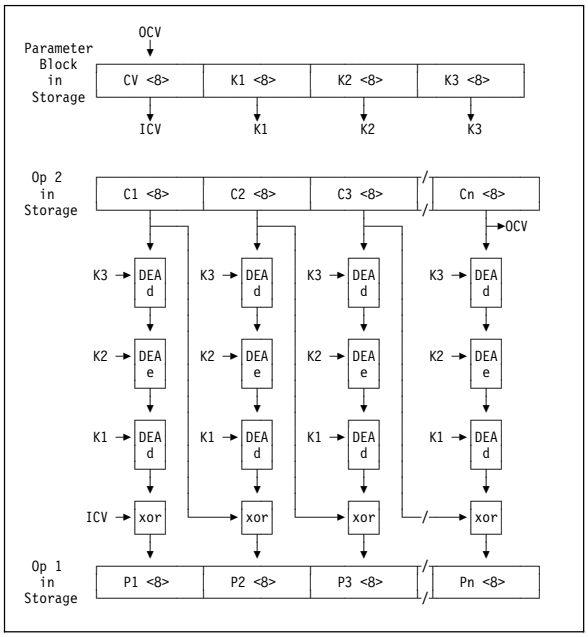



*Figure 7-27. KMC-TDEA-192 Decipher Operation*

### Special Conditions for KM and KMC

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bits 25-31 of general register 0 specify an unassigned or uninstalled function code.
2. The $R_1$ or $R_2$ field designates an odd-numbered register or general register 0.
3. The second operand length is not a multiple of the data block size of the designated function (see Figure 7-3 on page 7-26 to determine the data block sizes for CIPHER MESSAGE functions; see Figure 7-4 on page 7-26 to determine the data block sizes for CIPHER MESSAGE WITH CHAINING functions). This specification-exception condition does not apply to the query functions.

***Resulting Condition Code:***

0 Normal completion
1 --
2 --
3 Partial completion

***Program Exceptions:***

- Access (fetch, operand 2 and cryptographic key; store, operand 1; fetch and store, chaining value)
- Operation (if the message-security assist is not installed)



LzLabs GmbH. Ex. 1008-338

- Specification

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Operation exception. |
| 8. | Specification exception due to invalid function code or invalid register number. |
| 9. | Specification exception due to invalid operand length. |
| 10. | Condition code 0 due to second-operand length originally zero. |
| 11. | Access exceptions for an access to the parameter block, first, or second operand. |
| 12. | Condition code 0 due to normal completion (second-operand length originally nonzero, but stepped to zero). |
| 13. | Condition code 3 due to partial completion (second-operand length still nonzero). |

*Figure 7-28. Priority of Execution: KM and KMC*

**Programming Notes:**

1. When condition code 3 is set, the general registers containing the operand addresses and length, and, for CIPHER MESSAGE WITH CHAINING, the chaining value in the parameter block, are usually updated such that the program can simply branch back to the instruction to continue the operation.

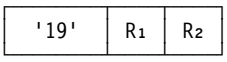
   For unusual situations, the CPU protects against endless reoccurrence of the no-progress case and also protects against setting condition code 3 when the portion of the first and second operands to be reprocessed overlap in storage. Thus, the program can safely branch back to the instruction whenever condition code 3 is set with no exposure to an endless loop and no exposure to incorrectly retrying the instruction.

2. If the length of the second operand is nonzero initially and condition code 0 is set, the registers are updated in the same manner as for condition code 3. For CIPHER MESSAGE WITH CHAINING, the chaining value in this case is such that additional operands can be processed as if they were part of the same chain.

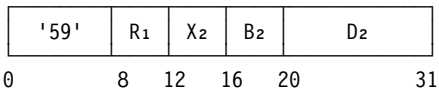
3. To save storage, the first and second operands may overlap exactly or the starting point of the first operand may be to the left of the starting point of the second operand. In either case, the overlap is not destructive.

## COMPARE

```
CR     R1,R2      [RR]
```




```
C      R1,D2(X2,B2)      [RX]
```




The first operand is compared with the second operand, and the result is indicated in the condition code. The operands are treated as 32-bit signed binary integers.

***Resulting Condition Code:***

0 Operands equal
1 First operand low
2 First operand high


LzLabs GmbH. Ex. 1008-339

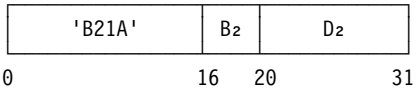
3 --

***Program Exceptions:***

- Access (fetch, operand 2 of C only)

## COMPARE AND FORM CODEWORD

CFC $D_2(B_2)$ [S]




General register 2 contains an index, which is used along with the contents of general registers 1 and 3 to designate the starting addresses of two fields in storage, called the first and third operands. The first and third operands are logically compared, and a codeword is formed for use in sort/merge algorithms.

The second-operand address is not used to address data. Bits 17-30 of the second-operand address, with one rightmost and one leftmost zero appended, are used as a 16-bit index limit. Bit 31 of the second-operand address is the operand-control bit. When bit 31 is zero, the codeword is formed from the high operand; when bit 31 is one, the codeword is formed from the low operand. The remainder of the second-operand address is ignored.

General registers 1 and 3 contain the base addresses of the first and third operands. Bits 16-31 of general register 2 are used as an index for addressing both the first and third operands. General registers 1, 2, and 3 must all initially contain even values; otherwise, a specification exception is recognized.

In the access-register mode, access register 1 specifies the address space containing the first and third operands.

The operation consists in comparing the first and third operands halfword by halfword and incrementing the index until an unequal pair of halfwords is found or the index exceeds the index limit. This proceeds in units of operation, between which interruptions may occur. The condition code is unpredictable if the instruction is interrupted.

At the start of a unit of operation, the index, bits 16-31 of general register 2, is logically compared with the index limit. If the index is larger, the instruction is completed by placing the contents of general register 3, with bit 0 set to one, in general register 2, and by setting condition code 0.

If the index is less than or equal to the index limit, the index is applied to the first-operand and third-operand base addresses to locate the current pair of halfwords to be compared. The index, with 16 leftmost zeros appended, and the contents of general register 1 are added to form a 32-bit intermediate value. A carry out of bit position 0, if any, is ignored. The address of the current first-operand halfword is generated from the intermediate value by following the normal rules for operand address generation. The address of the current third-operand halfword is formed in the same manner by adding the contents of general register 3 and the index.

The current first-operand and third-operand halfwords are logically compared. If they are equal, the contents of general register 2 are incremented by 2, and a unit of operation ends.

If the compare values are unequal, the contents of general register 2 are incremented by 2 and then shifted left logically by 16 bit positions. If the operand-control bit is zero, (1) the one's complement of the higher halfword is placed in the right half of general register 2, and (2) if operand 1 was higher, the contents of general registers 1 and 3 are interchanged. If the operand-control bit is one, (1) the lower halfword is placed in the right half of general register 2, and (2) if operand 1 was lower, the contents of general registers 1 and 3 are interchanged.

For the purpose of recognizing access exceptions, operand 1 and operand 3 are both considered to have a length equal to 2 more than the value of the index limit minus the index. When the index is initially larger than the index limit, access exceptions are not recognized for the storage operands. For operands longer than 4K bytes, access exceptions are not recognized more than 4K bytes beyond the byte being processed. Access exceptions are not recognized when a specification-exception condition exists.



LzLabs GmbH. Ex. 1008-340

If the $B_2$ field designates general register 2, it is unpredictable whether or not the index limit is recomputed; thus, in this case the operand length is unpredictable. However, in no case can the operands exceed $2^{15}$ bytes in length.

***Resulting Condition Code:***

0 Operands equal
1 Operand-control bit zero and operand 1 low, or operand-control bit one and operand 3 low
2 Operand-control bit zero and operand 1 high, or operand-control bit one and operand 3 high
3 --

***Program Exceptions:***

- Access (fetch, operands 1 and 3)
- Specification

**Programming Notes:**

1. An example of the use of COMPARE AND FORM CODEWORD is given in "Sorting Instructions" in Appendix A, "Number Representation and Instruction-Use Examples."

2. The offset of the halfword of the first and third operands at which comparison is to begin should be placed in bit positions 16-31 of general register 2 before executing COMPARE AND FORM CODEWORD. The index limit derived from the second-operand address should be the offset of the last halfword of the first and third operands for which comparison can be made. When the operands do not compare equal, the left half of the codeword formed in general register 2 by the execution of COMPARE AND FORM CODEWORD gives the offset of the first halfword not compared. If the codewords compare equal in an UPDATE TREE operation, bit positions 0-15 of general register 2 will contain the offset at which another COMPARE AND FORM CODEWORD should resume comparison for breaking codeword ties. Operand-control-bit values of zero or one are used for sorting operands in ascending or descending order, respectively. Refer to "Sorting Instructions" on page A-51 for a discussion of the use of codewords in sorting.

3. The condition code indicates the results of comparing operands up to 32,768 bytes long. Equal operands result in a negative codeword in general register 2. A negative codeword also results when the index limit is 32,766 and the operands that are compared differ in only their last two bytes. If this latter codeword is used by UPDATE TREE, an incorrect result may be indicated in general registers 0 and 1. Therefore, the index limit should not exceed 32,764 when the resulting codeword is to be used by UPDATE TREE.

4. Special precautions should be taken if COMPARE AND FORM CODEWORD is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.

5. Further programming notes concerning interruptible instructions are included in "Interruptible Instructions" in Chapter 5, "Program Execution."

6. The storage-operand references of COMPARE AND FORM CODEWORD may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

7. Figure 7-29 on page 7-38 and Figure 7-30 on page 7-39 contain summaries of the operation.


LzLabs GmbH. Ex. 1008-341

| Operand-Control Bit | Relation | Resulting Condition Code | Result in GR2 | Result in GR1 | Result in GR3 |
|---|---|---|---|---|---|
| 0 | op1 = op3 | 0 | OGR3b1 | - | - |
| 0 | op1 < op3 | 1 | X, nop3 | - | - |
| 0 | op1 > op3 | 2 | X, nop1 | OGR3 | OGR1 |
| 1 | op1 = op3 | 0 | OGR3b1 | - | - |
| 1 | op1 < op3 | 2 | X, top1 | OGR3 | OGR1 |
| 1 | op1 > op3 | 1 | X, top3 | - | - |

**Explanation:**

- The contents of the register remain unchanged.

OGR1 The original contents of GR1.

OGR3 The original contents of GR3.

OGR3b1 The original contents of GR3 with bit 0 set to one.

X Bits 0-15 of GR2 are 2 more than the index of the first unequal halfword.

nop1 Bits 16-31 of GR2 are the one's complement of the first unequal halfword in operand 1.

nop3 Bits 16-31 of GR2 are the one's complement of the first unequal halfword in operand 3.

top1 Bits 16-31 of GR2 are the first unequal halfword in operand 1.

top3 Bits 16-31 of GR2 are the first unequal halfword in operand 3.

*Figure 7-29. Operation of COMPARE AND FORM CODEWORD*



LzLabs GmbH. Ex. 1008-342

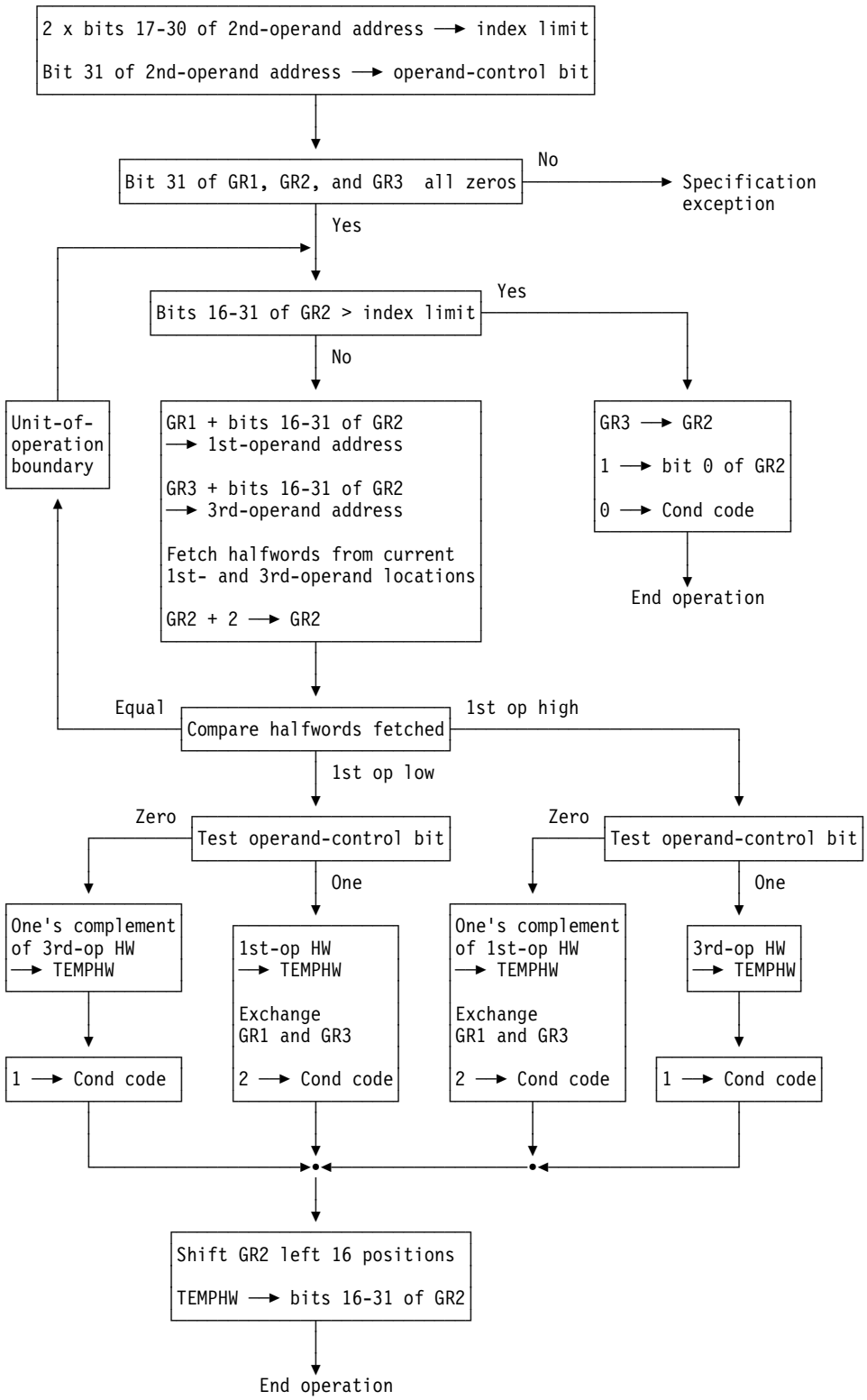


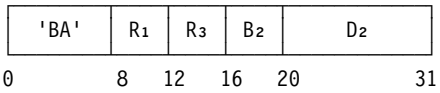

*Figure 7-30. Execution of COMPARE AND FORM CODEWORD*



LzLabs GmbH. Ex. 1008-343

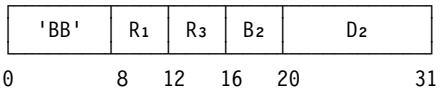
## COMPARE AND SWAP

CS R1,R3,D2(B2) [RS]




## COMPARE DOUBLE AND SWAP

CDS R1,R3,D2(B2) [RS]




The first and second operands are compared. If they are equal, the third operand is stored at the second-operand location. If they are unequal, the second operand is loaded at the first-operand location. The result of the comparison is indicated in the condition code.

For COMPARE AND SWAP, the first and third operands are 32 bits in length, with each operand occupying a general register. The second operand is a word in storage.

For COMPARE DOUBLE AND SWAP, the first and third operands are 64 bits in length, with each operand occupying an even-odd pair of general registers. The second operand is a doubleword in storage.

When an equal comparison occurs, the third operand is stored at the second-operand location. The fetch of the second operand for purposes of comparison and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs.

When the result of the comparison is unequal, the second operand is loaded at the first-operand location, and the second-operand location remains unchanged. However, on some models, the contents may be fetched and subsequently stored back unchanged at the second-operand location. This update appears to be a block-concurrent interlocked-update reference as observed by other CPUs.

A serialization function is performed before the operand is fetched and again after the operation is completed.

The second operand of COMPARE AND SWAP must be designated on a word boundary. The $R_1$ and $R_3$ fields for COMPARE DOUBLE AND SWAP must each designate an even-numbered register, and the second operand for the CDS instruction must be designated on a doubleword boundary. Otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 First and second operands equal, second operand replaced by third operand
1 First and second operands unequal, first operand replaced by second operand
2 --
3 --

***Program Exceptions:***

- Access (fetch and store, operand 2)
- Specification

**Programming Notes:**

1. Several examples of the use of the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions are given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. COMPARE AND SWAP can be used by CPU programs sharing common storage areas in either a multiprogramming or multiprocessing environment. Two examples are:
   a. By performing the following procedure, a CPU program can modify the contents of a storage location even though the possibility exists that the CPU program may be interrupted by another CPU program that will update the location or that another CPU program may simultaneously update the location. First, the entire word containing the byte or bytes to be updated is loaded into a general register. Next, the updated value is computed and placed in another general register. Then COMPARE AND SWAP is executed with the $R_1$ field designating the register that contains the original value and the $R_3$ field designating the register that contains the updated value. If the update has been



LzLabs GmbH. Ex. 1008-344

successful, condition code 0 is set. If the storage location no longer contains the original value, the update has not been successful, the general register designated by the $R_1$ field of the COMPARE AND SWAP instruction contains the new current value of the storage location, and condition code 1 is set. When condition code 1 is set, the CPU program can repeat the procedure using the new current value.

b. COMPARE AND SWAP can be used for controlled sharing of a common storage area, including the capability of leaving a message (in a chained list of messages) when the common area is in use. To accomplish this, a word in storage can be used as a control word, with a zero value in the word indicating that the common area is not in use and that no messages exist, a negative value indicating that the area is in use and that no messages exist, and a nonzero positive value indicating that the common area is in use and that the value is the address of the most recent message added to the list. Thus, any number of CPU programs desiring to seize the area can use COMPARE AND SWAP to update the control word to indicate that the area is in use or to add messages to the list. The single CPU program which has seized the area can also safely use COMPARE AND SWAP to remove messages from the list.

3. COMPARE DOUBLE AND SWAP can be used in a manner similar to that described for COMPARE AND SWAP. In addition, it has another use. Consider a chained list, with a control word used to address the first message in the list, as described in programming note 2b above. If multiple CPU programs are to be permitted to delete messages by using COMPARE AND SWAP (and not just the single CPU program which has seized the common area), there is a possibility the list will be incorrectly updated. This would occur if, for example, after one CPU program has fetched the address of the most recent message in order to remove the message, another CPU program removes the first two messages and then adds the first message back into the chain. The first CPU program, on continuing, cannot easily detect that the list is changed. By increasing the size of the control word to a doubleword containing both the first message address and a word with a change number that is incremented for each modification of the list, and by using COMPARE DOUBLE AND SWAP to update both fields together, the possibility of the list being incorrectly updated is reduced to a negligible level. That is, an incorrect update can occur only if the first CPU program is delayed while changes exactly equal in number to a multiple of $2^{32}$ take place *and* only if the last change places the original message address in the control word.

4. COMPARE AND SWAP and COMPARE DOUBLE AND SWAP do not interlock against storage accesses by channel programs. Therefore, the instructions should not be used to update a location at which a channel program may store, since the channel-program data may be lost.

5. To ensure successful updating of a common storage field by two or more CPUs, all updates must be done by means of an interlocked-update reference. COMPARE AND SWAP, COMPARE AND SWAP AND PURGE, COMPARE DOUBLE AND SWAP, and TEST AND SET are the only instructions that perform an interlocked-update reference. For example, if one CPU executes OR IMMEDIATE and another CPU executes COMPARE AND SWAP to update the same byte, the fetch by OR IMMEDIATE may occur either before the fetch by COMPARE AND SWAP or between the fetch and the store by COMPARE AND SWAP, and then the store by OR IMMEDIATE may occur after the store by COMPARE AND SWAP, in which case the change made by COMPARE AND SWAP is lost.

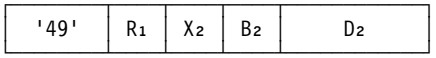
6. For the case of a condition-code setting of 1, COMPARE AND SWAP and COMPARE DOUBLE AND SWAP may or may not, depending on the model, cause any of the following to occur for the second-operand location: a PER storage-alteration event may be recognized; a protection exception for storing may be recognized; and, provided no access exception exists, the change bit may be set to one. Because the contents of storage remain unchanged, the change bit


LzLabs GmbH. Ex. 1008-345

may or may not be one when a PER storage-alteration event is recognized.

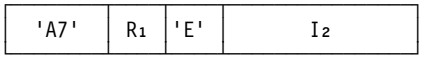
## COMPARE HALFWORD

```
CH      R1,D2(X2,B2)      [RX]
```




## COMPARE HALFWORD IMMEDIATE

```
CHI     R1,I2     [RI]
```




The first operand is compared with the second operand, and the result is indicated in the condition code. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand is treated as a 32-bit signed binary integer.

***Resulting Condition Code:***

0 Operands equal
1 First operand low
2 First operand high
3 --

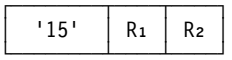
***Program Exceptions:***

- Access (fetch, operand 2 of CH only)
- Operation (CHI if the immediate-and-relative-instruction facility is not installed)

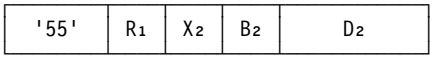
**Programming Note:** An example of the use of the COMPARE HALFWORD instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

## COMPARE LOGICAL

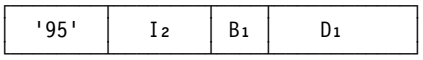
```
CLR     R1,R2     [RR]
```



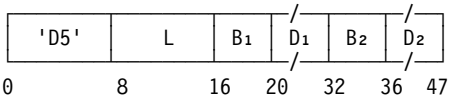

```
CL      R1,D2(X2,B2)      [RX]
```




```
CLI     D1(B1),I2         [SI]
```




```
CLC     D1(L,B1),D2(B2)          [SS]
```




The first operand is compared with the second operand, and the result is indicated in the condition code.

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the fields is reached. For COMPARE LOGICAL (CL) and COMPARE LOGICAL (CLC), access exceptions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte.

***Resulting Condition Code:***

0 Operands equal
1 First operand low
2 First operand high
3 --

***Program Exceptions:***

- Access (fetch, operand 2, CL and CLC; fetch, operand 1, CLI and CLC)

**Programming Notes:**

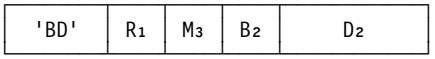
1. Examples of the use of the COMPARE LOGICAL instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. COMPARE LOGICAL treats all bits of each operand alike as part of a field of unstructured logical data. For COMPARE LOGICAL (CLC), the comparison may extend to field lengths of 256 bytes.



LzLabs GmbH. Ex. 1008-346

## COMPARE LOGICAL CHARACTERS UNDER MASK

```
CLM    R1,M3,D2(B2)      [RS]
```




The first operand is compared with the second operand under control of a mask, and the result is indicated in the condition code.

The contents of the $M_3$ field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of general register $R_1$. The byte positions corresponding to ones in the mask are considered as a contiguous field and are compared with the second operand. The second operand is a contiguous field in storage, starting at the second-operand address and equal in length to the number of ones in the mask. The bytes in the general register corresponding to zeros in the mask do not participate in the operation.

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the fields is reached.

When the mask is not zero, exceptions associated with storage-operand access are recognized for no more than the number of bytes specified by the mask. Access exceptions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte. When the mask is zero, access exceptions are recognized for one byte at the second-operand address.

***Resulting Condition Code:***

0 Operands equal, or mask bits all zeros
1 First operand low
2 First operand high
3 --

***Program Exceptions:***

- Access (fetch, operand 2)

**Programming Note:** An example of the use of the COMPARE LOGICAL CHARACTERS UNDER MASK instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”

## COMPARE LOGICAL LONG

```
CLCL   R1,R2      [RR]
```




The first operand is compared with the second operand, and the result is indicated in the condition code. The shorter operand is considered to be extended on the right with padding bytes.

The $R_1$ and $R_2$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively. The number of bytes in the first-operand and second-operand locations is specified by unsigned binary integers in bit positions 8-31 of general registers $R_1 + 1$ and $R_2 + 1$, respectively. Bit positions 0-7 of general register $R_2 + 1$ contain the padding byte. The contents of bit positions 0-7 of general register $R_1 + 1$ are ignored.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit position 0 are ignored.

The contents of the registers just described are shown in Figure 7-31 on page 7-44.


LzLabs GmbH. Ex. 1008-347

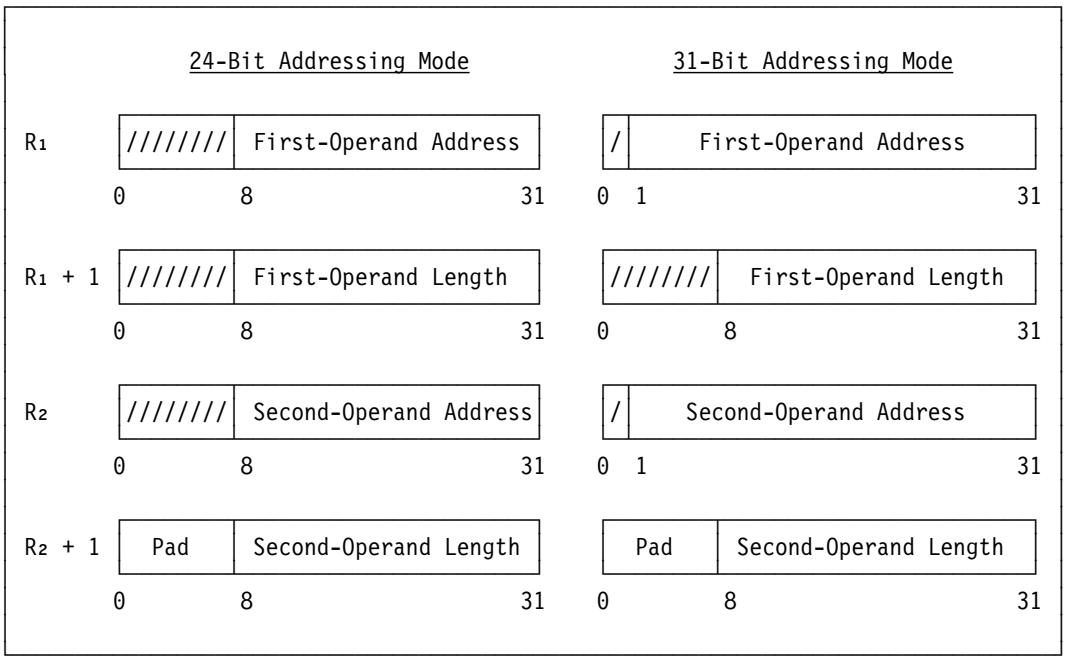



*Figure 7-31. Register Contents for COMPARE LOGICAL LONG*

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the longer operand is reached. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of padding bytes.

If both operands are of zero length, the operands are considered to be equal.

The execution of the instruction is interruptible. When an interruption occurs, other than one that follows termination, the contents of general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes compared, and the contents of general registers $R_1$ and $R_2$ are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. The leftmost bits which are not part of the address in general registers $R_1$ and $R_2$ are set to zeros; the contents of bit positions 0-7 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged; and the condition code is unpredictable. If the operation is interrupted after the shorter operand has been exhausted, the length field pertaining to the shorter operand is zero, and its address is updated accordingly.

If the operation ends because of an inequality, the address fields in general registers $R_1$ and $R_2$ at completion identify the first unequal byte in each operand. The lengths in bit positions 8-31 of general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes that were equal, unless the inequality occurred with the padding byte, in which case the length field for the shorter operand is set to zero. The addresses in general registers $R_1$ and $R_2$ are incremented by the amounts by which the corresponding length fields were reduced.

If the two operands, including the padding byte, if necessary, are equal, both length fields are made zero at completion, and the addresses are incremented by the corresponding operand-length values.

At the completion of the operation, the leftmost bits which are not part of the address in general registers $R_1$ and $R_2$ are set to zeros, even when one or both of the initial length values are zero. The contents of bit positions 0-7 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged.

Access exceptions for the portion of a storage operand to the right of the first unequal byte may or may not be recognized. For operands longer than 2K bytes, access exceptions are not recog-



LzLabs GmbH. Ex. 1008-348

nized more than 2K bytes beyond the byte being processed. Access exceptions are not indicated for locations more than 2K bytes beyond the first unequal byte.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

***Resulting Condition Code:***

0 Operands equal, or both zero length
1 First operand low
2 First operand high
3 --

***Program Exceptions:***

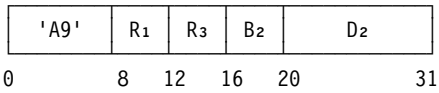
- Access (fetch, operands 1 and 2)
- Specification

**Programming Notes:**

1. An example of the use of the COMPARE LOGICAL LONG instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When the $R_1$ and $R_2$ fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, except that the contents of the designated registers are incremented or decremented only by the number of bytes compared, not by twice the number of bytes compared. In the absence of dynamic modification of the operand area by another CPU or by a channel program, condition code 0 is set. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed.
3. Special precautions should be taken when COMPARE LOGICAL LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.
4. Other programming notes concerning interruptible instructions are included in "Interruptible Instructions" in Chapter 5, "Program Execution."
5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

## COMPARE LOGICAL LONG EXTENDED

CLCLE $R_1,R_3,D_2(B_2)$ [RS]




The first operand is compared with the third operand until unequal bytes are compared, the end of the longer operand is reached, or a CPU-determined number of bytes have been compared, whichever occurs first. The shorter operand is considered to be extended on the right with padding bytes. The result is indicated in the condition code.

The $R_1$ and $R_3$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and third operand is designated by the contents of general registers $R_1$ and $R_3$, respectively. The number of bytes in the first-operand and third-operand locations is specified by bits 0-31 of general registers $R_1 + 1$ and $R_3 + 1$, respectively. The contents of general registers $R_1 + 1$ and $R_3 + 1$ are treated as 32-bit unsigned binary integers.

The handling of the addresses in general registers $R_1$ and $R_3$ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_3$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general registers $R_1$ and $R_3$ constitute the address, and the contents of bit position 0 are ignored.

The second-operand address is not used to address data; instead, the rightmost eight bits of the second-operand address, bits 24-31, are the padding byte. Bits 0-23 of the second-operand address are ignored.


LzLabs GmbH. Ex. 1008-349

The contents of the registers and address just described are shown in Figure 7-32 on page 7-46.

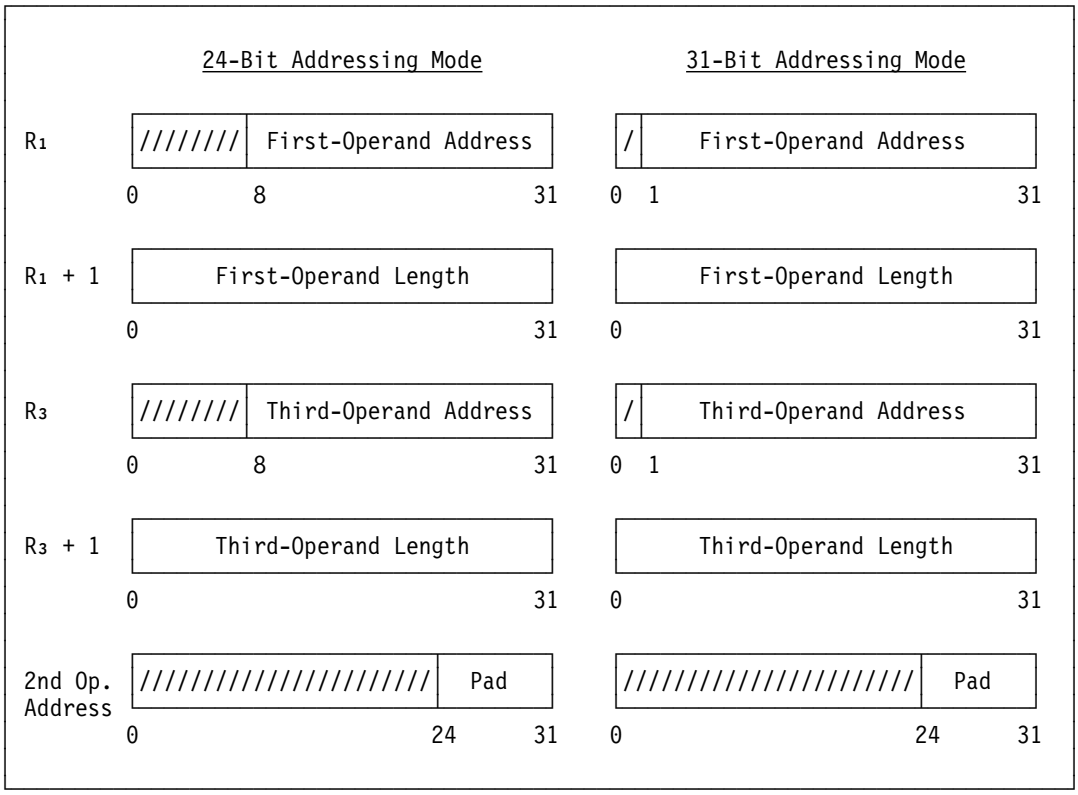



*Figure 7-32. Register Contents and Second-Operand Address for COMPARE LOGICAL LONG EXTENDED*

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found, the end of the longer operand is reached, or a CPU-determined number of bytes have been compared, whichever occurs first. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of padding bytes.

If both operands are of zero length, the operands are considered to be equal.

If the operation ends because of an inequality, the address fields in general registers $R_1$ and $R_3$ at completion identify the first unequal byte in each operand. The lengths in general registers $R_1 + 1$ and $R_3 + 1$ are decremented by the number of bytes that were equal, unless the inequality occurred with the padding byte, in which case the length field for the shorter operand is set to zero. The addresses in general registers $R_1$ and $R_3$ are incremented by the amounts by which the corresponding length fields were decremented. Condition code 1 is set if the first operand is low, or condition code 2 is set if the first operand is high.

If the two operands, including the padding byte, if necessary, are equal, both length fields are made zero at completion, and the addresses are incremented by the corresponding operand-length values. Condition code 0 is set.

If the operation is completed because a CPU-determined number of bytes have been compared without finding an inequality or reaching the end of the longer operand, the lengths in general registers $R_1 + 1$ and $R_3 + 1$ are decremented by the number of bytes compared, and the addresses in general registers $R_1$ and $R_3$ are incremented by the same number, so that the instruction, when reexecuted, resumes at the next bytes to be compared. If the operation is completed after the shorter operand has been exhausted, the length field pertaining to the shorter operand is zero, and



LzLabs GmbH. Ex. 1008-350

the operand address is updated accordingly. Condition code 3 is set.

The padding byte may be formed from $D_2(B_2)$ multiple times during the execution of the instruction, and the registers designated by $R_1$ and $R_3$ may be updated multiple times. Therefore, if $B_2$ equals $R_1$, $R_1 + 1$, $R_3$, or $R_3 + 1$ and is subject to change during the execution of the instruction, the results are unpredictable.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed. The maximum amount is approximately 4K bytes of either operand.

At the completion of the operation, the leftmost bits which are not part of the address in general registers $R_1$ and $R_3$ may be set to zeros or may remain unchanged from their original values, even when one or both of the initial length values are zero.

Access exceptions for the portion of a storage operand to the right of the first unequal byte may or may not be recognized. For operands longer than 4K bytes, access exceptions are not recognized more than 4K bytes beyond the byte being processed. Access exceptions are not indicated for locations more than 4K bytes beyond the first unequal byte.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

***Resulting Condition Code:***

0 All bytes compared; operands equal, or both zero length
1 First operand low
2 First operand high
3 CPU-determined number of bytes compared without finding an inequality

***Program Exceptions:***

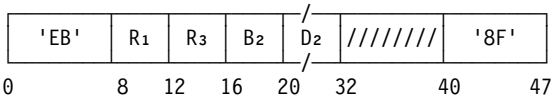
- Access (fetch, operands 1 and 3)
- Operation (if the compare-and-move-extended facility is not installed)
- Specification

**Programming Notes:**

1. COMPARE LOGICAL LONG EXTENDED is intended for use in place of COMPARE LOGICAL LONG when the operand lengths are specified as 32-bit binary integers. COMPARE LOGICAL LONG EXTENDED sets condition code 3 in cases in which COMPARE LOGICAL LONG would be interrupted.
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the comparison. The program need not determine the number of bytes that were compared.
3. The function of not processing more than approximately 4K bytes of either operand is intended to permit software polling of a flag that may be set by a program on another CPU during long operations.
4. When the $R_1$ and $R_3$ fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, except that the contents of the designated registers are incremented or decremented only by the number of bytes compared, not by twice the number of bytes compared. In the absence of dynamic modification of the operand area by another CPU or by a channel program, the condition code is finally set to 0 after possible settings to 3. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed. If storage is not accessed, condition code 3 may or may not be set regardless of the operand length.
5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

## COMPARE LOGICAL LONG UNICODE

```
CLCLU    R1,R3,D2(B2)      [RSE]
```




The first operand is compared with the third operand until unequal two-byte Unicode characters are compared, the end of the longer operand


LzLabs GmbH. Ex. 1008-351

is reached, or a CPU-determined number of characters have been compared, whichever occurs first. The shorter operand is considered to be extended on the right with two-byte padding characters. The result is indicated in the condition code.

The $R_1$ and $R_3$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost character of the first operand and third operand is designated by the contents of general registers $R_1$ and $R_3$, respectively. The number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 0-31 of general registers $R_1 + 1$ and $R_3 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers.

The contents of general registers $R_1 + 1$ and $R_3 + 1$ must specify an even number of bytes; otherwise, a specification exception is recognized.

The handling of the addresses in general registers $R_1$ and $R_3$ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_3$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of the registers constitute the address, and the contents of bit position 0 are ignored.

The second-operand address is not used to address data; instead, the rightmost 16 bits of the second-operand address, bits 16-31, are the two-byte padding character. Bits 0-15 of the second-operand address are ignored.

The contents of the registers and address just described are shown in Figure 7-33 on page 7-49.

The comparison proceeds left to right, character by character, and ends as soon as an inequality is found, the end of the longer operand is reached, or a CPU-determined number of characters have been compared, whichever occurs first. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of two-byte padding characters.

If both operands are of zero length, the operands are considered to be equal.

If the operation ends because of an inequality, the address fields in general registers $R_1$ and $R_3$ at completion identify the first unequal two-byte character in each operand. The lengths in bit positions 0-31 of general registers $R_1 + 1$ and $R_3 + 1$ are decremented by 2 times the number of characters that were equal, unless the inequality occurred with the two-byte padding character, in which case the length field for the shorter operand is set to zero. The addresses in general registers $R_1$ and $R_3$ are incremented by the amounts by which the corresponding length fields were decremented. Condition code 1 is set if the first operand is low, or condition code 2 is set if the first operand is high.

If the two operands, including the two-byte padding character, if necessary, are equal, both length fields are made zero at completion, and the addresses are incremented by the corresponding operand-length values. Condition code 0 is set.

If the operation is completed because a CPU-determined number of characters have been compared without finding an inequality or reaching the end of the longer operand, the lengths in general registers $R_1 + 1$ and $R_3 + 1$ are decremented by 2 times the number of characters compared, and the addresses in general registers $R_1$ and $R_3$ are incremented by the same number, so that the instruction, when reexecuted, resumes at the next characters to be compared. If the operation is completed after the shorter operand has been exhausted, the length field pertaining to the shorter operand is zero, and the operand address is updated accordingly. Condition code 3 is set.

The two-byte padding character may be formed from $D_2(B_2)$ multiple times during the execution of the instruction, and the registers designated by $R_1$ and $R_3$ may be updated multiple times. Therefore, if $B_2$ equals $R_1$, $R_1 + 1$, $R_3$, or $R_3 + 1$ and is subject to change during the execution of the instruction, the results are unpredictable.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system perform-



LzLabs GmbH. Ex. 1008-352

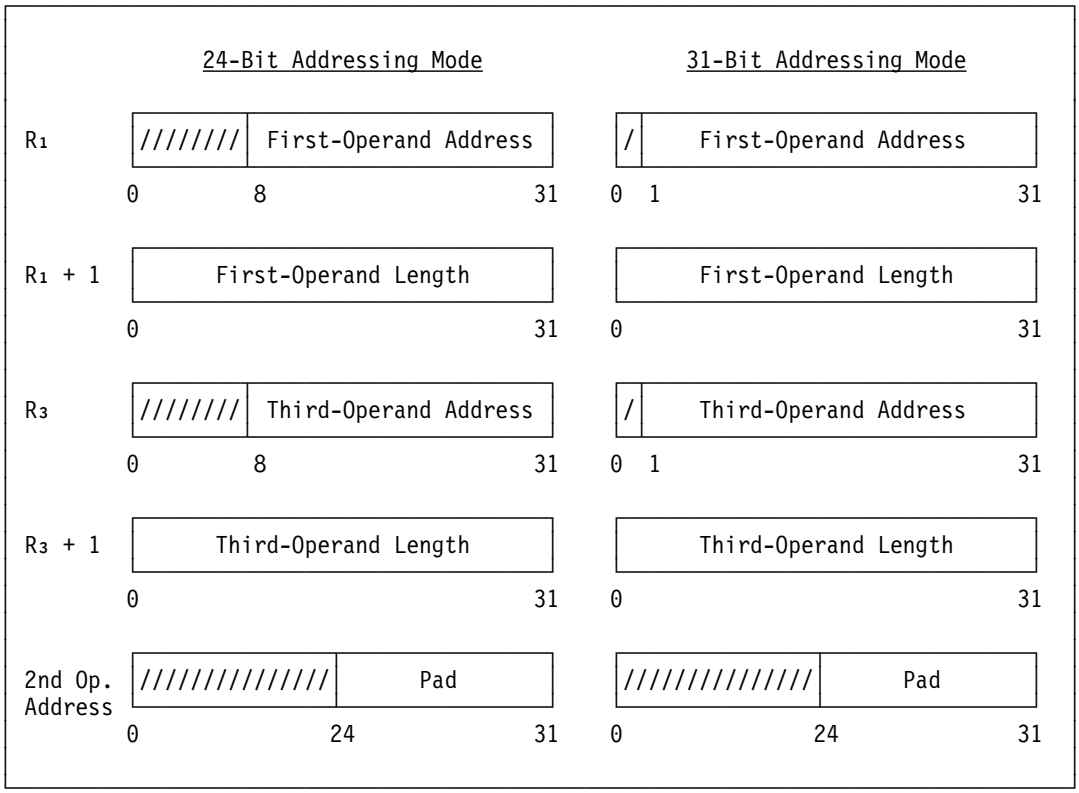



*Figure 7-33. Register Contents and Second-Operand Address for COMPARE LOGICAL LONG UNICODE*

ance, and it may be a different amount each time the instruction is executed.

At the completion of the operation, the leftmost bits which are not part of the address in general registers $R_1$ and $R_3$ may be set to zeros or may remain unchanged from their original values, including the case when one or both of the initial length values are zero.

Access exceptions for the portion of a storage operand to the right of the first unequal character may or may not be recognized. For operands longer than 4K bytes, access exceptions are not recognized more than 4K bytes beyond the character being processed. Access exceptions are not indicated for locations more than 4K bytes beyond the first unequal character.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field or length associated with that operand is odd.

***Resulting Condition Code:***

0 All characters compared; operands equal, or both zero length
1 First operand low
2 First operand high
3 CPU-determined number of characters compared without finding an inequality

***Program Exceptions:***

- Access (fetch, operands 1 and 3)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

**Programming Notes:**

1. COMPARE LOGICAL LONG UNICODE is intended for use in place of COMPARE LOGICAL LONG or COMPARE LOGICAL LONG EXTENDED when two-byte characters are to be compared. The characters may be Unicode characters or any other double-byte characters. COMPARE LOGICAL LONG UNICODE sets condition code 3 in cases in which COMPARE LOGICAL LONG would be interrupted.

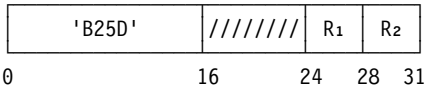

LzLabs GmbH. Ex. 1008-353

2. When condition code 3 is set, the program can simply branch back to the instruction to continue the comparison. The program need not determine the number of characters that were compared.
3. When the $R_1$ and $R_3$ fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, except that the contents of the designated registers are incremented or decremented only by 2 times the number of characters compared, not by 4 times the number of characters compared. In the absence of dynamic modification of the operand area by another CPU or by a channel program, the condition code is finally set to 0 after possible settings to 3. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed. If storage is not accessed, condition code 3 may or may not be set regardless of the operand length.
4. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
5. If padding with a Unicode space character is required (or any character whose representation is less than or equal to FFF hex), the character may be represented in the displacement field of the instruction, for example:

```
CLCLU   6,8,X'020'
```

## COMPARE LOGICAL STRING

```
CLST     R1,R2     [RRE]
```




The first operand is compared with the second operand until unequal bytes are compared, the end of either operand is reached, or a CPU-determined number of bytes have been compared, whichever occurs first. The CPU-determined number is at least 256. The result is indicated in the condition code.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit position 0 are ignored.

The first and second operands may be of the same or different lengths. The end of an operand is indicated by an ending character in the last byte position of the operand. The ending character to be used to determine the end of an operand is specified in bit positions 24-31 of general register 0. Bit positions 0-23 of general register 0 are reserved for possible future extensions and must contain all zeros; otherwise, a specification exception is recognized.

The operation proceeds left to right, byte by byte, and ends as soon as the ending character is encountered in either or both operands, unequal bytes which do not include an ending character are compared, or a CPU-determined number of bytes have been compared, whichever occurs first. The CPU-determined number is at least 256. When the ending character is encountered simultaneously in both operands, even when it is in the first byte position of the operands, the operands are of the same length and are considered to be equal, and condition code 0 is set. When the ending character is encountered in only one operand, that operand, which is the shorter operand, is considered to be low, and condition code 1 or 2 is set. Condition code 1 is set if the first operand is low, or condition code 2 is set if the second operand is low. Similarly, when unequal bytes which do not include an ending character are compared, condition code 1 is set if the lower byte is in the first operand, or condition code 2 is set if the lower byte is in the second operand. When a CPU-determined number of bytes have been compared, condition code 3 is set.

When condition code 1 or 2 is set, the address of the last byte processed in the first and second operands is placed in general registers $R_1$ and $R_2$, respectively. That is, when condition code 1



LzLabs GmbH. Ex. 1008-354

is set, the address of the ending character or first unequal byte in the first operand, whichever was encountered, is placed in general register $R_1$, and the address of the second-operand byte corresponding in position to the first-operand byte is placed in general register $R_2$. When condition code 2 is set, the address of the ending character or first unequal byte in the second operand, whichever was encountered, is placed in general register $R_2$, and the address of the first-operand byte corresponding in position to the second-operand byte is placed in general register $R_1$. When condition code 3 is set, the address of the next byte to be processed in the first and second operands is placed in general registers $R_1$ and $R_2$, respectively. Whenever an address is placed in a general register, bits 0-7 of the register, in the 24-bit mode, or bit 0, in the 31-bit mode, are set to zeros.

When condition code 0 is set, the contents of general registers $R_1$ and $R_2$ remain unchanged.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

Access exceptions for the first and second operands are recognized only for that portion of the operand which is necessarily examined in the operation.

***Resulting Condition Code:***

0 Entire operands equal; general registers $R_1$ and $R_2$ unchanged
1 First operand low; general registers $R_1$ and $R_2$ updated with addresses of last bytes processed
2 First operand high; general registers $R_1$ and $R_2$ updated with addresses of last bytes processed
3 CPU-determined number of bytes equal; general registers $R_1$ and $R_2$ updated with addresses of next bytes

***Program Exceptions:***

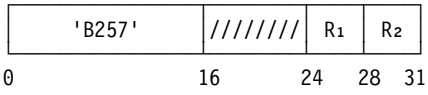
- Access (fetch, operands 1 and 2)
- Operation (if the string-instruction facility is not installed)
- Specification

**Programming Notes:**

1. Several examples of the use of the COMPARE LOGICAL STRING instruction are given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. When condition code 0 is set, no indication is given of the position of either ending character.
3. When condition code 3 is set, the program can simply branch back to the instruction to continue the comparison. The program need not determine the number of bytes that were compared.
4. $R_1$ or $R_2$ may be zero, in which case general register 0 is treated as containing an address and also the ending character.
5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

## COMPARE UNTIL SUBSTRING EQUAL

```
CUSE     R1,R2     [RRE]
```




The first operand is compared with the second operand until equal substrings (sequences of bytes) of a specified length are found, the end of the longer operand is reached, or a CPU-determined number of unequal bytes have been compared, whichever occurs first. The shorter operand is considered to be extended on the right with padding bytes. The CPU-determined number is at least 256. The result is indicated in the condition code.

The $R_1$ and $R_2$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is specified by the contents of the $R_1$ and $R_2$ general registers, respectively. The number of bytes in the first-operand and second-operand locations is specified by the 32-bit signed binary integer in general


LzLabs GmbH. Ex. 1008-355

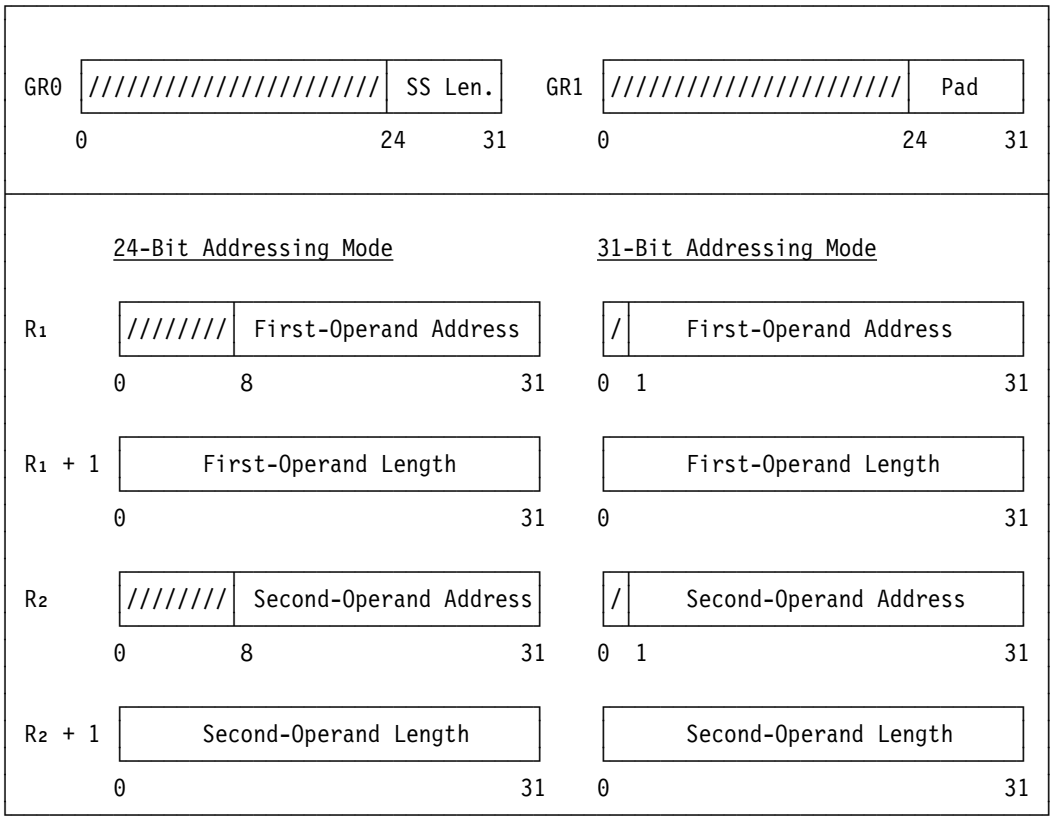



*Figure 7-34. Register Contents for COMPARE UNTIL SUBSTRING EQUAL*

registers $R_1 + 1$ and $R_2 + 1$, respectively. When an operand length is negative, it is treated as zero, and it remains unchanged upon completion of the instruction.

Bits 24-31 of general register 0 specify the unsigned substring length, a value of 0-255, in bytes. Bits 24-31 of general register 1 are the padding byte. Bits 0-23 of general registers 0 and 1 are ignored.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit position 0 are ignored.

The contents of the registers just described are shown in Figure 7-34.

The result is obtained as if the operands were processed from left to right. However, multiple accesses may be made to all or some of the bytes of each operand.

The comparison proceeds left to right, byte by byte, and ends as soon as (1) equal substrings of the specified length are found, (2) the end of the longer operand is reached without finding equal substrings of the specified length, or (3) the last bytes compared are unequal, and a CPU-determined number of bytes have been compared. The CPU-determined number is at least 256. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of padding bytes.

If the operation ends because equal substrings of the specified length were found, the condition code is set to 0. If the operation ends because the end of the longer operand was reached without finding equal substrings of the specified length, the condition code is set to 1 if equal bytes were the last bytes compared, or it is set to 2 if



LzLabs GmbH. Ex. 1008-356

unequal bytes were the last bytes compared. If the operation ends because unequal bytes were compared when a CPU-determined number of bytes had been compared, the condition code is set to 3.

If the specified substring length is zero, it is considered that equal substrings of the specified length were found, and condition code 0 is set.

If both operands are of zero length but the specified substring length is not zero, it is considered that the end of the longer operand was reached when unequal bytes were the last bytes compared, and condition code 2 is set.

If equal bytes have been compared but then unequal bytes are compared, it is considered that all bytes so far compared are unequal.

At the completion of the operation, the operand-length fields in the $R_1 + 1$ and $R_2 + 1$ registers are decremented by the number of unequal bytes compared (including equal bytes before unequal bytes compared), and the addresses in the $R_1$ and $R_2$ registers are incremented by the same number. However, in the case when a byte of the longer operand is compared against the padding byte, the length field for the shorter operand is not decremented below zero, and the corresponding address is not incremented above the address of the first byte after the shorter operand. The left-most bits which are not part of the addresses in registers $R_1$ and $R_2$ are set to zeros, even if the substring length is zero or both operand lengths are initially zero.

Thus, when condition code 0 or 1 is set, the resulting addresses in the $R_1$ and $R_2$ registers designate the first bytes of equal substrings in the two operands, and the lengths in the $R_1 + 1$ and $R_2 + 1$ registers have been decremented by the number of bytes preceding the equal substrings, except when the equal substring in the shorter operand begins with the padding byte, in which case the length field for the shorter operand is zero, and the corresponding address field has been incremented by the operand length. When condition code 2 is set, each address field designates the first byte after the corresponding operand, and both length fields are zero. When condition code 3 is set, each address field designates the first byte after the last compared byte of the corresponding operand, and both length fields have been decremented by the number of bytes compared, except that a length field is not decremented below zero.

When the contents of the $R_1$ and $R_2$ fields are the same, the first and second operands may be compared, or the condition code may be set to 0 or 1 without comparing the operands.

The substring length and padding byte may be fetched from general registers 0 and 1 multiple times during the execution of the instruction, and the registers designated by $R_1$ and $R_2$ may be updated multiple times. Therefore, if $R_1$ or $R_2$ is zero, the results are unpredictable.

When condition code 3 is set, the general registers used by the instruction have been set so that the remainder of the operands can be processed by simply branching back and reexecuting the instruction.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

The execution of the instruction is interruptible when the last bytes compared are unequal; it is not interruptible when the last bytes compared are equal. When an interruption occurs, other than one that follows termination, the contents of the registers designated by the $R_1$ and $R_2$ fields are updated the same as upon normal completion of the instruction, so that the instruction, when reexecuted, resumes at the point of interruption. The condition code is unpredictable.

Access exceptions for the portion of a storage operand to the right of the last byte processed may or may not be recognized. For operands longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Although the operand address and length fields remain unchanged when a zero substring length is specified, the recognition of access exceptions is not necessarily prevented.


LzLabs GmbH. Ex. 1008-357

***Resulting Condition Code:***

0 Equal substrings of specified length found
1 End of longer operand reached when last bytes compared are equal
2 End of longer operand reached when last bytes compared are unequal
3 Last bytes compared are unequal, and CPU-determined number of bytes compared

***Program Exceptions:***

- Access (fetch, operands 1 and 2)
- Specification

**Programming Notes:**

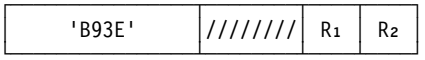
1. When the $R_1$ and $R_2$ fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, and, in the absence of dynamic modification of the operand area by another CPU or by a channel program, condition code 0, 1, or 2 is set (as explained in the next note). However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed.
2. If the contents of the $R_1$ and $R_2$ fields are the same and the operand length is nonzero, and provided that another CPU or a channel program is not changing an operand, condition code 0 is set if the operand length is equal to or greater than the specified substring length, or condition code 1 is set if the operand length is less than the specified substring length. Whether or not $R_1$ equals $R_2$, if both operand lengths are zero, condition code 0 is set if the specified substring length is zero, or condition code 2 is set if the specified substring length is nonzero. In all of these cases, the addresses in the $R_1$ and $R_2$ registers and the lengths in the $R_1 + 1$ and $R_2 + 1$ registers remain unchanged.
3. Special precautions should be taken when COMPARE UNTIL SUBSTRING EQUAL is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.
4. Other programming notes concerning interruptible instructions are included in “Interruptible Instructions” on page 5-17.
5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
6. The storage-operand references of COMPARE UNTIL SUBSTRING EQUAL may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)



LzLabs GmbH. Ex. 1008-358

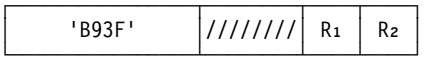
## COMPUTE INTERMEDIATE MESSAGE DIGEST (KIMD)

```
KIMD     R1,R2        [RRE]
```




## COMPUTE LAST MESSAGE DIGEST (KLMD)

```
KLMD     R1,R2        [RRE]
```




A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction and the $R_1$ field are ignored.

Bit positions 25-31 of general register 0 contain the function code. Figures 7-35 and 7-36 show the assigned function codes for COMPUTE INTERMEDIATE MESSAGE DIGEST and COMPUTE LAST MESSAGE DIGEST, respectively. All other function codes are unassigned. Bit 24 of general register 0 must be zero; otherwise, a specification exception is recognized. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 8-31 of general register 1 constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general register 1 constitute the address, and the content of bit position 0 is ignored.

The function codes for COMPUTE INTERMEDIATE MESSAGE DIGEST are as follows.

*Figure 7-35. Function Codes for COMPUTE INTERMEDIATE MESSAGE DIGEST*

| Code | Function | Parm. Block Size (bytes) | Data Block Size (bytes) |
|---|---|---|---|
| 0 | KIMD-Query | 16 | — |
| 1 | KIMD-SHA-1 | 20 | 64 |

**Explanation:**

— Not applicable

The function codes for COMPUTE LAST MESSAGE DIGEST are as follows.

*Figure 7-36. Function Codes for COMPUTE LAST MESSAGE DIGEST*

| Code | Function | Parm. Block Size (bytes) | Data Block Size (bytes) |
|---|---|---|---|
| 0 | KLMD-Query | 16 | — |
| 1 | KLMD-SHA-1 | 28 | 64 |

**Explanation:**

— Not applicable

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers $R_2$ and $R_2 + 1$ are ignored for the query function.

For all other functions, the second operand is processed as specified by the function code using an initial chaining value in the parameter block, and the result replaces the chaining value. For COMPUTE LAST MESSAGE DIGEST, the operation also uses a message bit length in the parameter block. The operation proceeds until the end of the second-operand location is reached or a CPU-determined number of bytes have been processed, whichever occurs first. The result is indicated in the condition code.

The $R_2$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.


LzLabs GmbH. Ex. 1008-359

The location of the leftmost byte of the second operand is specified by the contents of the $R_2$ general register. The number of bytes in the second-operand location is specified in general register $R_2 + 1$.

As part of the operation, the address in general register $R_2$ is incremented by the number of bytes processed from the second operand, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the address and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general register $R_2$ constitute the address of second operand, and the contents of bit positions 0-7 are ignored; bits 8-31 of the updated address replace the corresponding bits in general register $R_2$, carries out of bit position 8 of the updated address are ignored, and the contents of bit positions 0-7 of general register $R_2$ are set to zeros. In the 31-bit addressing mode, the contents of bit positions 1-31 of general register $R_2$ constitute the address of second operand, and the content of bit position 0 is ignored; bits 1-31 of the updated address replace the corresponding bits in general register $R_2$, carries out of bit position 1 of the updated address are ignored, and bit 0 of general register $R_2$ is set to zero.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 0-31 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which specifies the number of bytes in the second operand; and the updated value replaces the contents of bit positions 0-31 of general register $R_2 + 1$.

Figure 7-37 shows the contents of the general registers just described.

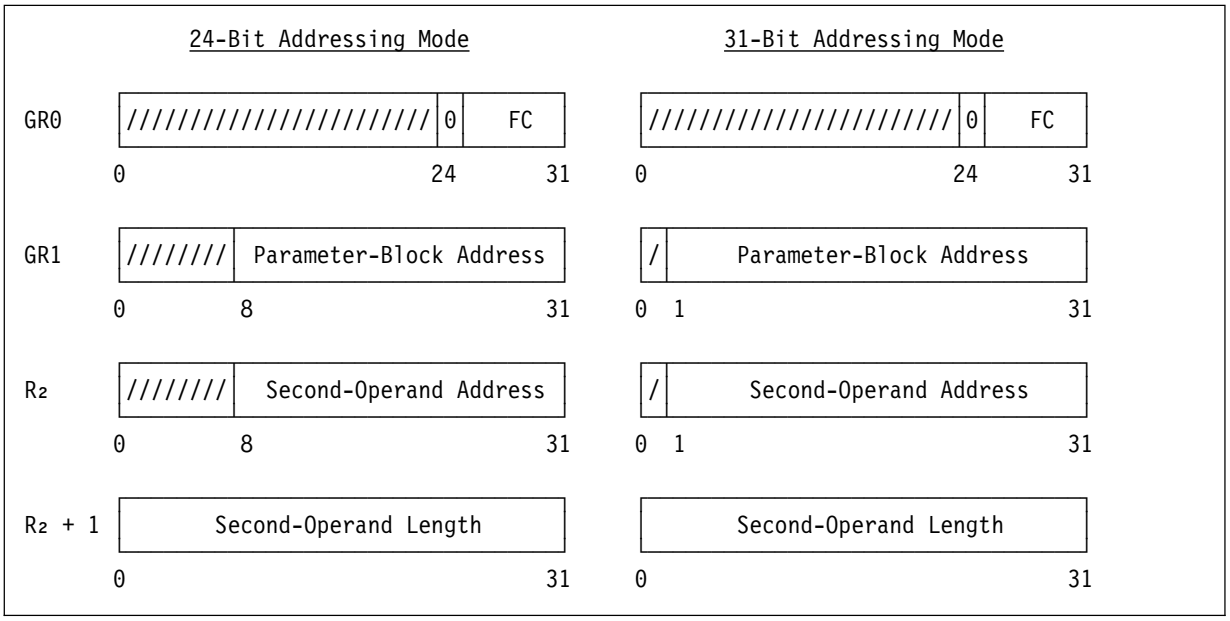



*Figure 7-37. General Register Assignment for KIMD and KLMD*

In the access-register mode, access registers 1 and $R_2$ specify the address spaces containing the parameter block and second operand, respectively.

The result is obtained as if processing starts at the left end of the second operand and proceeds to the right, block by block. The operation is ended when all source bytes in the second operand have been processed (called normal completion), or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually nonzero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless reoccurrence of this no-progress case.



LzLabs GmbH. Ex. 1008-360

When the chaining-value field overlaps any portion of the second operand, the result in the chaining-value field is unpredictable.

For COMPUTE INTERMEDIATE MESSAGE DIGEST, normal completion occurs when the number of bytes in the second operand as specified in general register $R_2 + 1$ have been processed. For COMPUTE LAST MESSAGE DIGEST, after all bytes in the second operand as specified in general register $R_2 + 1$ have been processed, the padding operation is performed, and then normal completion occurs.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in $R_2 + 1$ is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in $R_2 + 1$ is nonzero.

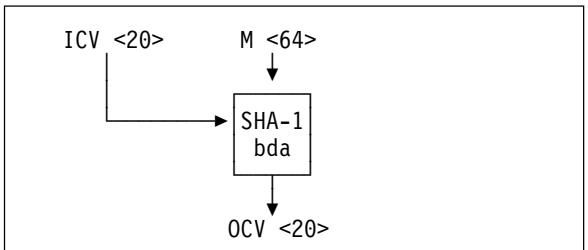
When the second-operand length is initially zero, the second operand is not accessed, general registers $R_2$ and $R_2 + 1$ are not changed, and condition code 0 is set. For COMPUTE INTERMEDIATE MESSAGE DIGEST, the parameter block is not accessed. However, for COMPUTE LAST MESSAGE DIGEST, the empty block (L = 0) case padding operation is performed and the result is stored into the parameter block.

As observed by other CPUs and channel programs, references to the parameter block and storage operands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

Access exceptions may be reported for a larger portion of the second operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of the second operand nor for locations more than 4K bytes beyond the current location being processed.

### Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the COMPUTE INTERMEDIATE MESSAGE DIGEST and COMPUTE LAST MESSAGE DIGEST functions. Further description of the secure hash algorithm may be found in *Secure Hash Standard*, Federal Information Processing Standards publication 180-1, National Institute of Standards and Technology, Washington DC, April 17, 1995.




| Symbol | Explanation |
|---|---|
| <n> | Length of item in bytes |
| ICV | Initial chaining value |
| M | Message block |
| OCV | Output chaining value |

*Figure 7-38. Symbol for SHA-1 Block Digest Algorithm*

### KIMD-Query (KIMD Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-37 on page 7-56.

The parameter block used for the function has the following format:

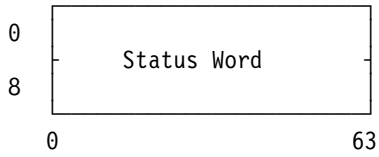



*Figure 7-39. Parameter Block for KIMD-Query*

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the COMPUTE INTERMEDIATE MESSAGE DIGEST instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KIMD-Query function completes; condition code 3 is not applicable to this function.

### KIMD-SHA-1 (KIMD Function Code 1)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-37 on page 7-56.


LzLabs GmbH. Ex. 1008-361

The parameter block used for the function has the following format:

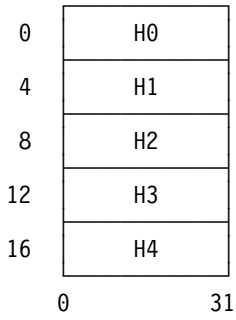



*Figure 7-40. Parameter Block for KIMD-SHA-1*

A 20-byte intermediate message digest is generated for the the 64-byte message blocks in operand 2 using the SHA-1 block digest algorithm with the 20-byte chaining value in the parameter block. The generated intermediate message digest, also called the output chaining value (OCV), is stored in the chaining-value field of the parameter block. The operation is shown in the following figure:

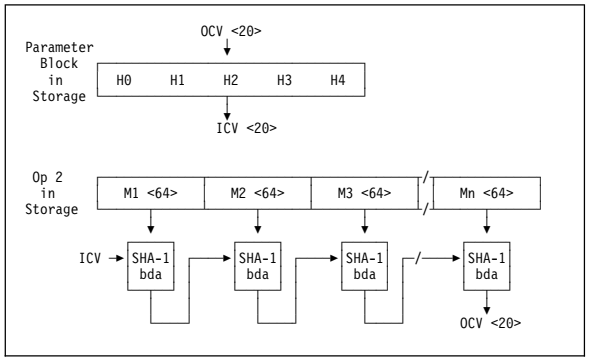



*Figure 7-41. KIMD-SHA-1*

**KLMD-Query (KLMD Function Code 0)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-37 on page 7-56.

The parameter block used for the function has the following format:

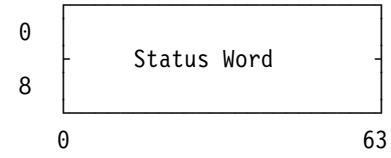



*Figure 7-42. Parameter Block for KLMD-Query*

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the COMPUTE LAST MESSAGE DIGEST instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KLMD-Query function completes; condition code 3 is not applicable to this function.

**KLMD-SHA-1 (KLMD Function Code 1)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-37 on page 7-56.

The parameter block used for the function has the following format:

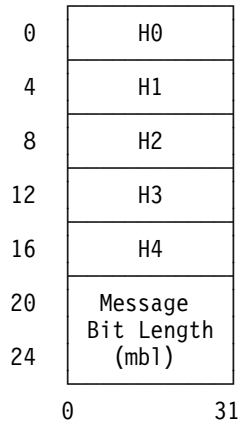



*Figure 7-43. Parameter Block for KLMD-SHA-1*

The message digest for the message (M) in operand 2 is generated using the SHA-1 algorithm with the chaining value and message-bit-length information in the parameter block.

If the length of the message in operand 2 is equal to or greater than 64 bytes, an intermediate message digest is generated for each 64-byte message block using the SHA-1 block digest algorithm with the 20-byte chaining value in the parameter block, and the generated intermediate message digest, also called the output chaining value (OCV), is stored into the chaining-value field of the parameter block. This operation is shown in Figure 7-44 on page 7-59 and repeats until the remaining message is less than 64 bytes.



LzLabs GmbH. Ex. 1008-362

If the length of the message or the remaining message is zero bytes, then the operation in Figure 7-45 on page 7-59 is performed. If the length of the message or the remaining message is between one byte and 55 bytes inclusive, then the operation in Figure 7-46 is performed; if the length is between 56 bytes and 63 bytes inclusive, then the operation in Figure 7-47 on page 7-60 is performed; The message digest, also called the output chaining value (OCV), is stored into the chaining-value field of the parameter block.

### Additional Symbols Used in KLMD Functions

The following additional symbols are used in the description of the COMPUTE LAST MESSAGE DIGEST functions.

| Symbol | Explanation for KLMD Function Figures |
|---|---|
| L | Byte length of operand 2 in storage. |
| p <n> | n padding bytes; leftmost byte is 80 hex; all other bytes are 00 hex. |
| z <56> | 56 padding bytes of zero. |
| mbl | an 8-byte value specifying the bit length of the total message. |
| q <64> | a padding block, consisting of 56 bytes of zero followed by an 8-byte mbl. |

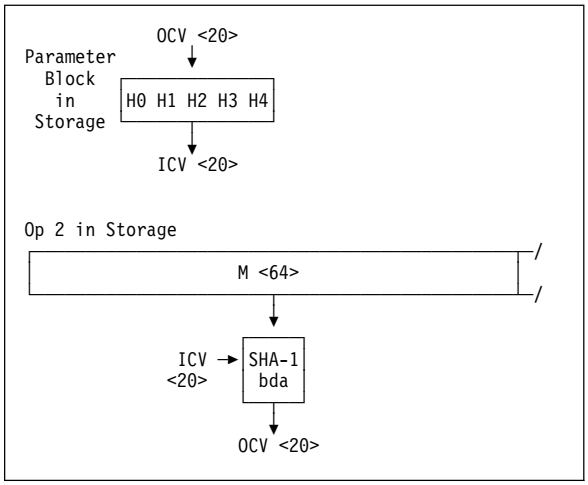



*Figure 7-44. KLMD-SHA-1 Full Block (L ≥ 64)*

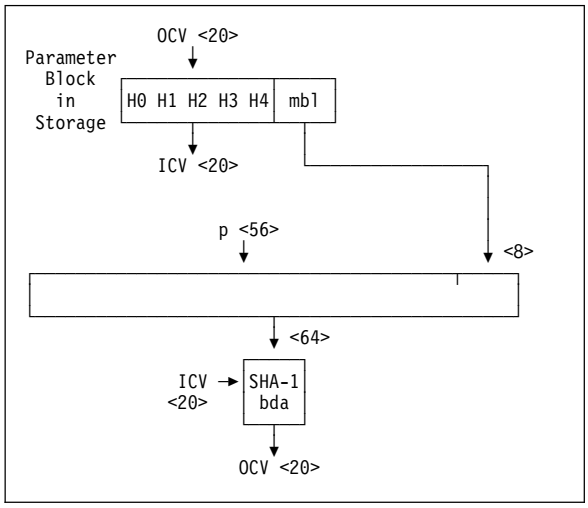



*Figure 7-45. KLMD-SHA-1 Empty Block (L = 0)*

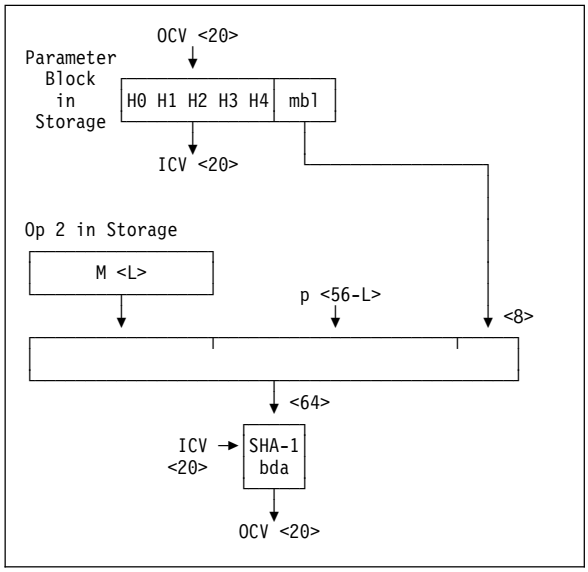



*Figure 7-46. KLMD-SHA-1 Partial-Block Case 1 (1 ≤ L ≤ 55)*


LzLabs GmbH. Ex. 1008-363

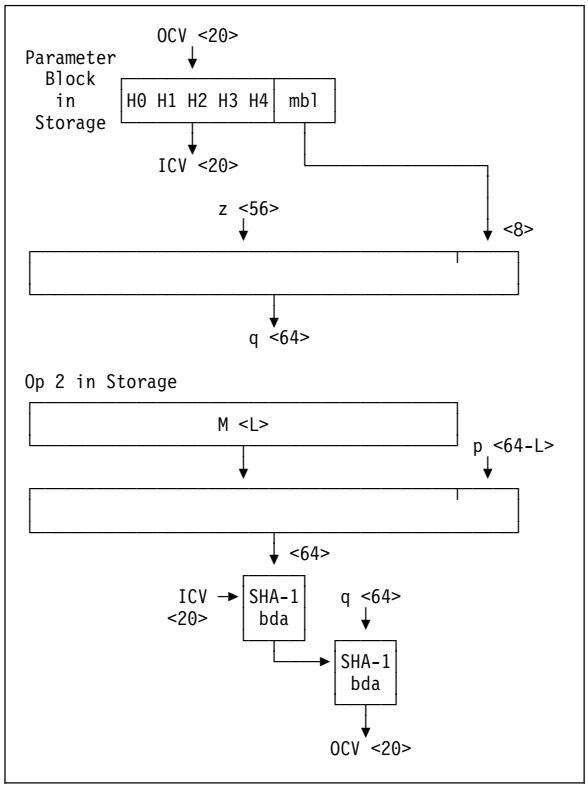



*Figure 7-47. KLMD-SHA-1 Partial-Block Case 2 (56 ≤ L ≤ 63)*

### Special Conditions for KIMD and KLMD

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bit 24 of general register 0 is not zero.
2. Bits 25-31 of general register 0 specify an unassigned or uninstalled function code.
3. The $R_2$ field designates an odd-numbered register or general register 0.
4. For COMPUTE INTERMEDIATE MESSAGE DIGEST, the second-operand length is not a multiple of the data block size of the designated function (see Figure 7-35 on page 7-55 to determine the data block sizes for COMPUTE INTERMEDIATE MESSAGE DIGEST functions). This specification-exception condition does not apply to the the query function, nor does it apply to COMPUTE LAST MESSAGE DIGEST.

***Resulting Condition Code:***

0 Normal completion
1 --
2 --
3 Partial completion

***Program Exceptions:***

- Access (fetch, operand 2 and message bit length; fetch and store, chaining value)
- Operation (if the message-security assist is not installed)
- Specification

```
1.-6.  Exceptions with the same priority as the priority of program-
       interruption conditions for the general case.

7.A    Access exceptions for second instruction halfword.

7.B    Operation exception.

8.     Specification exception due to invalid function code or
       invalid register number.

9.     Specification exception due to invalid operand length.

10.    Condition code 0 due to second-operand length originally zero.

11.    Access exceptions for an access to the parameter block or
       second operand.

12.    Condition code 0 due to normal completion (second-operand
       length originally nonzero, but stepped to zero).

13.    Condition code 3 due to partial completion (second-operand
       length still nonzero).
```

*Figure 7-48. Priority of Execution: KIMD and KLMD*



LzLabs GmbH. Ex. 1008-364

**Programming Notes:**

1. Bit 24 of general register 0 is reserved for future extension and should be set to zero.

2. When condition code 3 is set, the second operand address and length in general registers $R_2$ and $R_2 + 1$, respectively, and the the chaining-value in the parameter block are usually updated such that the program can simply branch back to the instruction to continue the operation.

   For unusual situations, the CPU protects against endless reoccurrence for the no-progress case. Thus, the program can safely branch back to the instruction whenever condition code 3 is set with no exposure to an endless loop.

3. If the length of the second operand is nonzero initially and condition code 0 is set, the registers are updated in the same manner as for condition code 3; the chaining value in this case is such that additional operands can be processed as if they were part of the same chain.

4. The instructions COMPUTE INTERMEDIATE MESSAGE DIGEST and COMPUTE LAST MESSAGE DIGEST are designed to be used by a security service application programming interface (API). These APIs provide the program with means to compute the digest of messages of almost unlimited size, including those too large to fit in storage all at once. This is accomplished by permitting the program to pass the message to the API in parts. The following programming notes are described in terms of these APIs.

5. Before processing the first part of a message, the program must set the initial values for the chaining-value field. For SHA-1, the initial chaining values are listed as follows:

   ```
   H0 = x'6745 2301'
   H1 = x'EFCD AB89'
   H2 = x'98BA DCFE'
   H3 = x'1032 5476'
   H4 = x'C3D2 E1F0'
   ```

6. When processing message parts other than the last, the program must process message parts in multiples of 512 bits (64 bytes) and use the COMPUTE INTERMEDIATE MESSAGE DIGEST instruction.

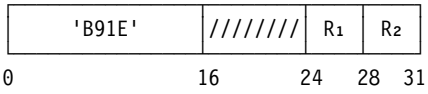
7. When processing the last message part, the program must compute the length of the original message in bits and place this 64-bit value in the message-bit-length field of the parameter block, and use the COMPUTE LAST MESSAGE DIGEST instruction.

8. The COMPUTE LAST MESSAGE DIGEST instruction does not require the second operand to be a multiple of the block size. It first processes complete blocks, and may set condition code 3 before processing all blocks. After processing all complete blocks, it then performs the padding operation including the remaining portion of the second operand. This may require one or two iterations of the SHA-1 block digest algorithm.

9. The COMPUTE LAST MESSAGE DIGEST instruction provides the SHA-1 padding for messages that are a multiple of eight bits in length. If SHA-1 is to be applied to a bit string which is not a multiple of eight bits, the program must perform the padding and use the COMPUTE INTERMEDIATE MESSAGE DIGEST instruction.

# COMPUTE MESSAGE AUTHENTICATION CODE (KMAC)

KMAC $R_1,R_2$ [RRE]




A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction and the $R_1$ field are ignored.

Bit positions 25-31 of general register 0 contain the function code. Figure 7-49 shows the assigned function codes. All other function codes are unassigned. Bit 24 of general register 0 must be zero; otherwise, a specification exception is recognized. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 8-31 of general register 1


LzLabs GmbH. Ex. 1008-365

constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general register 1 constitute the address, and the content of bit position 0 is ignored.

The function codes for COMPUTE MESSAGE AUTHENTICATION CODE are as follows.

*Figure 7-49. Function Codes for COMPUTE MESSAGE AUTHENTICATION CODE*

| Code | Function | Parm. Block Size (bytes) | Data Block Size (bytes) |
|---|---|---|---|
| 0 | KMAC-Query | 16 | — |
| 1 | KMAC-DEA | 16 | 8 |
| 2 | KMAC-TDEA-128 | 24 | 8 |
| 3 | KMAC-TDEA-192 | 32 | 8 |

**Explanation:**

— Not applicable

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers $R_2$ and $R_2 + 1$ are ignored.

For all other functions, the second operand is processed as specified by the function code using an initial chaining value in the parameter block and the result replaces the chaining value. The operation also uses a cryptographic key in the parameter block. The operation proceeds until the end of the second-operand location is reached or a CPU-determined number of bytes have been processed, whichever occurs first. The result is indicated in the condition code.

The $R_2$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the second operand is specified by the contents of the $R_2$ general register. The number of bytes in the second-operand location is specified in general register $R_2 + 1$.

As part of the operation, the address in general register $R_2$ is incremented by the number of bytes processed from the second operand, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the address and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general register $R_2$ constitute the address of second operand, and the contents of bit positions 0-7 are ignored; bits 8-31 of the updated address replace the corresponding bits in general register $R_2$, carries out of bit position 8 of the updated address are ignored, and the contents of bit positions 0-7 of general register $R_2$ are set to zeros. In the 31-bit addressing mode, the contents of bit positions 1-31 of general register $R_2$ constitute the address of second operand, and the content of bit position 0 is ignored; bits 1-31 of the updated address replace the corresponding bits in general register $R_2$, carries out of bit position 1 of the updated address are ignored, and the content of bit position 0 of general register $R_2$ is set to zero.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 0-31 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which specifies the number of bytes in the second operand; and the updated value replaces the contents of bit positions 0-31 of general register $R_2 + 1$.

Figure 7-50 on page 7-63 shows the contents of the general registers just described.



LzLabs GmbH. Ex. 1008-366

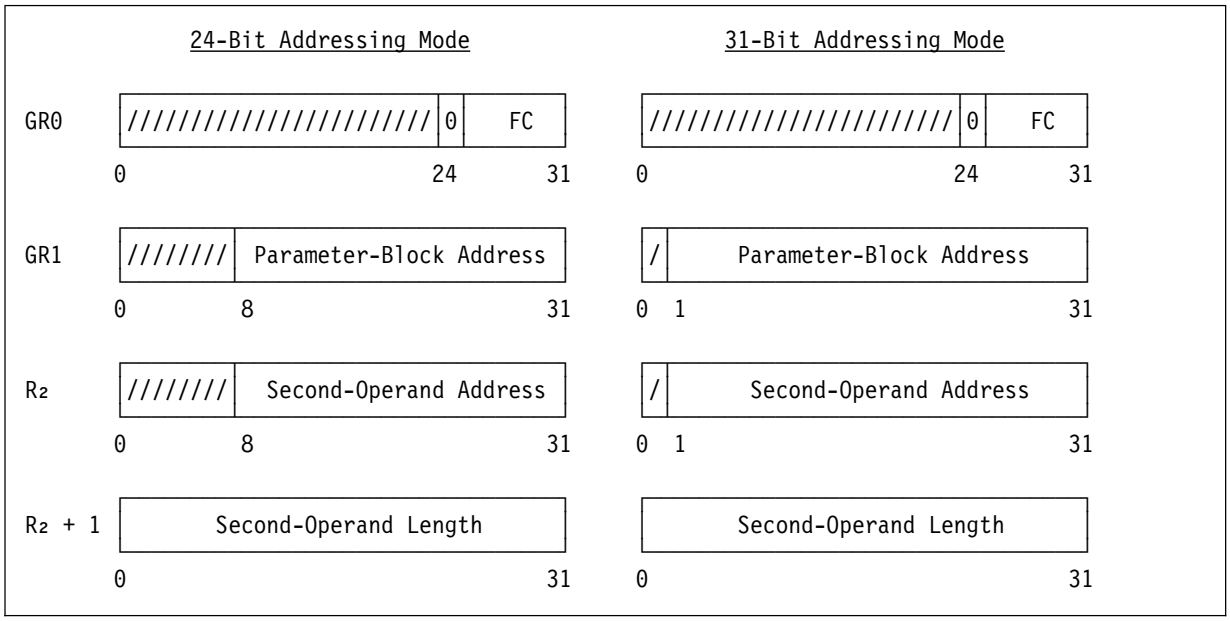



*Figure 7-50. General Register Assignment for KMAC*

In the access-register mode, access registers 1 and $R_2$ specify the address spaces containing the parameter block and second operand, respectively.

The result is obtained as if processing starts at the left end of the second operand and proceeds to the right, block by block. The operation is ended when all source bytes in the second operand have been processed (called normal completion), or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually nonzero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless reoccurrence of this no-progress case.

When the chaining-value field overlaps any portion of the second operand, the result in the chaining-value field is unpredictable.

Normal completion occurs when the number of bytes in the second operand as specified in general register $R_2 + 1$ have been processed.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in $R_2 + 1$ is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in $R_2 + 1$ is nonzero.

When the second-operand length is initially zero, the second operand and the parameter block are not accessed, general registers $R_2$ and $R_2 + 1$ are not changed, and condition code 0 is set.

As observed by other CPUs and channel programs, references to the parameter block and storage operands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

Access exceptions may be reported for a larger portion of the second operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of the second operand nor for locations more than 4K bytes beyond the current location being processed.

### Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the COMPUTE MESSAGE AUTHENTICATION CODE functions. For data-encryption-algorithm (DEA) functions, the DEA-key-parity bit in each byte of the DEA key is ignored, and the operation proceeds normally,


LzLabs GmbH. Ex. 1008-367

regardless of the DEA-key parity of the key. Further description of the data-encryption algorithm may be found in *Data Encryption Algorithm*, ANSI-X3.92.1981, American National Standard for Information Systems.

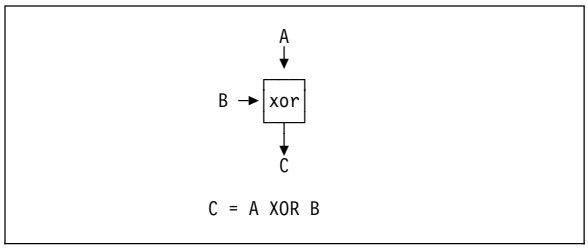



*Figure 7-51. Symbol For Bit-Wise Exclusive Or*

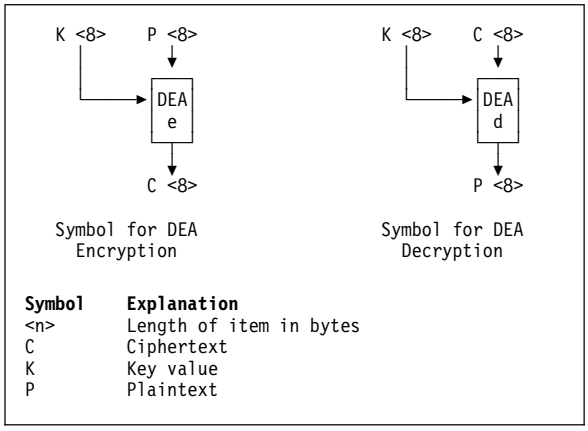



*Figure 7-52. Symbols for DEA Encryption and Decryption*

### KMAC-Query (Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-50 on page 7-63.

The parameter block used for the function has the following format:

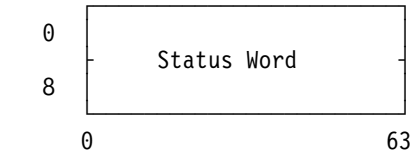



*Figure 7-53. Parameter Block for KMAC-Query*

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the KMAC instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KMAC-Query function completes; condition code 3 is not applicable to this function.

### KMAC-DEA (Function Code 1)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-50 on page 7-63.

The parameter block used for the function has the following format:

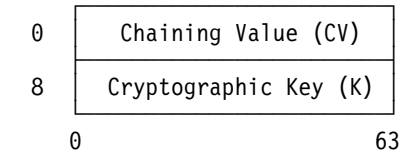



*Figure 7-54. Parameter Block for KMAC-DEA*

The message authentication code for the 8-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the DEA algorithm with the 64-bit cryptographic key and the 64-bit chaining value in the parameter block.

The message authentication code, also called the output chaining value (OCV), is stored in the chaining-value field of the parameter block. The operation is shown in the following figure:

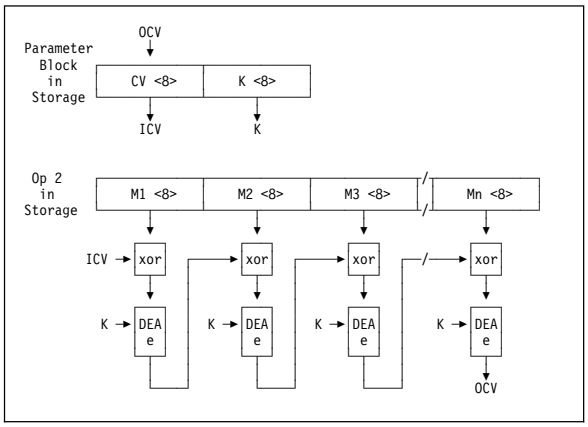



*Figure 7-55. KMAC-DEA*

### KMAC-TDEA-128 (Function Code 2)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-50 on page 7-63.



LzLabs GmbH. Ex. 1008-368

The parameter block used for the function has the following format:

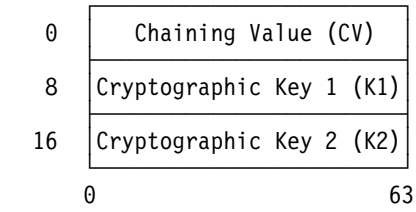



*Figure 7-56. Parameter Block for KMAC-TDEA-128*

The message authentication code for the 8-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the TDEA algorithm with the two 64-bit cryptographic keys and the 64-bit chaining value in the parameter block.

The message authentication code, also called the output chaining value (OCV), is stored in the chaining-value field of the parameter block. The operation is shown in the following figure:

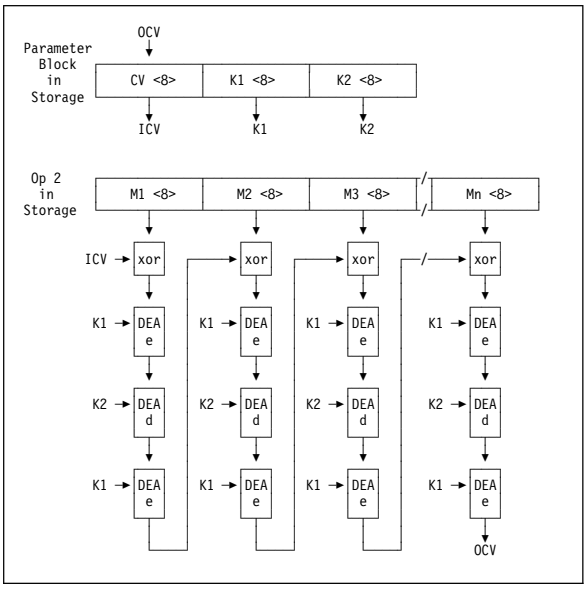



*Figure 7-57. KMAC-TDEA-128*

**KMAC-TDEA-192 (Function Code 3)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-50 on page 7-63.

The parameter block used for the function has the following format:

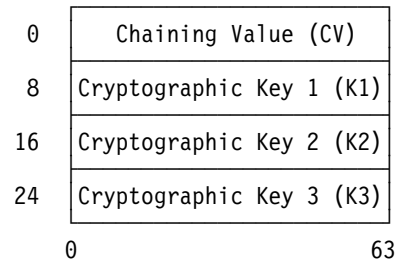



*Figure 7-58. Parameter Block for KMAC-TDEA-192*

The message authentication code for the 8-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the TDEA algorithm with the three 64-bit cryptographic keys and the 64-bit chaining value in the parameter block.

The message authentication code, also called the output chaining value (OCV), is stored in the chaining-value field of the parameter block. The operation is shown in the following figure:

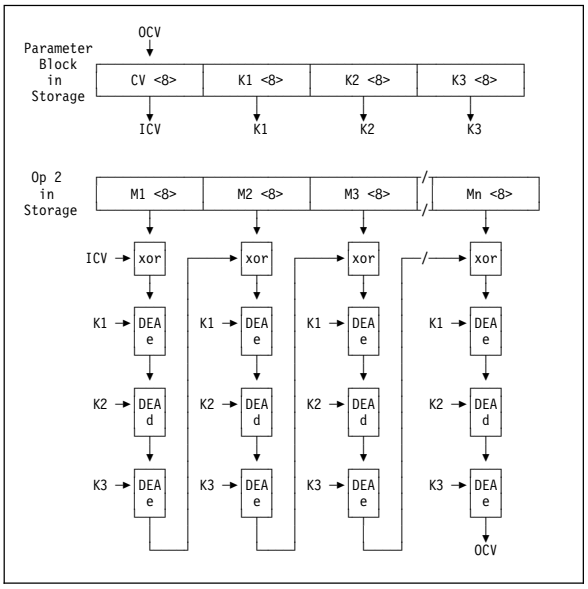



*Figure 7-59. KMAC-TDEA-192*

**Special Conditions for KMAC**

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bit 24 of general register 0 is not zero.
2. Bits 25-31 of general register 0 specify an unassigned or uninstalled function code.
3. The $R_2$ field designates an odd-numbered register or general register 0.


LzLabs GmbH. Ex. 1008-369

4. The second-operand length is not a multiple of the data block size of the designated function (see Figure 7-49 on page 7-62 to determine the data block size for COMPUTE MESSAGE AUTHENTICATION CODE functions).

***Resulting Condition Code:***

0 Normal completion
1 --
2 --
3 Partial completion

***Program Exceptions:***

- Access (fetch, operand 2, cryptographic key; fetch and store, chaining value)
- Operation (if the message-security assist is not installed)
- Specification

```
1.-6.  Exceptions with the same priority as the priority of program-
       interruption conditions for the general case.

7.A    Access exceptions for second instruction halfword.

7.B    Operation exception.

8.     Specification exception due to invalid function code or
       invalid register number.

9.     Specification exception due to invalid operand length.

10.    Condition code 0 due to second-operand length originally zero.

11.    Access exceptions for an access to the parameter block or
       second operand.

12.    Condition code 0 due to normal completion (second-operand
       length originally nonzero, but stepped to zero).

13.    Condition code 3 due to partial completion (second-operand
       length still nonzero).
```

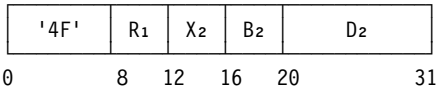
*Figure 7-60. Priority of Execution: KMAC*

**Programming Notes:**

1. Bit 24 of general register 0 is reserved for future extension and should be set to zero.
2. When condition code 3 is set, the second operand address and length in general registers $R_2$ and $R_2 + 1$, respectively, and the chaining-value in the parameter block are usually updated such that the program can simply branch back to the instruction to continue the operation. For unusual situations, the CPU protects against endless reoccurrence for the no-progress case. Thus, the program can safely branch back to the instruction whenever condition code 3 is set with no exposure to an endless loop.
3. If the length of the second operand is nonzero initially and condition code 0 is set, the registers are updated in the same manner as for condition code 3; the chaining value in this case is such that additional operands can be processed as if they were part of the same chain.
4. Before processing the first part of a message, the program must set the initial values for the chaining-value field. To comply with ANSI X9.9 or X9.19, the initial chaining value shall be set to all binary zeros.

## CONVERT TO BINARY

```
CVB    R1,D2(X2,B2)      [RX]
```




The second operand is changed from decimal to binary, and the result is placed at the first-operand location.



LzLabs GmbH. Ex. 1008-370

The second operand occupies eight bytes in storage and has the format of packed decimal data, as described in Chapter 8, “Decimal Instructions.” It is checked for valid sign and digit codes, and a decimal-operand data exception is recognized when an invalid code is detected.

The result of the conversion is a 32-bit signed binary integer, which is placed in general register $R_1$. The maximum positive number that can be converted and still be contained in a 32-bit register is 2,147,483,647; the maximum negative number (the negative number with the greatest absolute value) that can be converted is –2,147,483,648. For any decimal number outside this range, the operation is completed by placing the 32 rightmost bits of the binary result in the register, and a fixed-point-divide exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

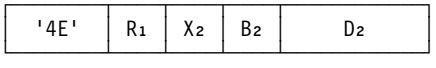
- Access (fetch, operand 2)
- Data
- Fixed-point divide

**Programming Notes:**

1. An example of the use of the CONVERT TO BINARY instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. When the second operand is negative, the result is in two's-complement notation.
3. The storage-operand references for CONVERT TO BINARY may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)

## CONVERT TO DECIMAL

```
CVD    R1,D2(X2,B2)      [RX]
```




The first operand is changed from binary to decimal, and the result is stored at the second-operand location. The first operand is treated as a 32-bit signed binary integer.

The result occupies eight bytes in storage and is in the format for packed decimal data, as described in Chapter 8, “Decimal Instructions.” The rightmost four bits of the result represent the sign. A positive sign is encoded as 1100; a negative sign is encoded as 1101.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

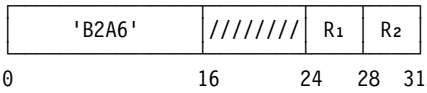
- Access (store, operand 2)

**Programming Notes:**

1. An example of the use of the CONVERT TO DECIMAL instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. The number to be converted is a 32-bit signed binary integer obtained from a general register. Since 15 decimal digits are available for the result, and the decimal equivalent of 31 bits requires at most 10 decimal digits, an overflow cannot occur.
3. The storage-operand references for CONVERT TO DECIMAL may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)

## CONVERT UNICODE TO UTF-8

```
CUUTF    R1,R2      [RRE]
```




The two-byte Unicode characters of the second operand are converted to UTF-8 characters and placed at the first-operand location. The UTF-8 characters are one, two, three, or four bytes, depending on the Unicode characters that are converted. The operation proceeds until the end of the first or second operand is reached or a CPU-determined number of characters have been converted, whichever occurs first. The result is indicated in the condition code.

The $R_1$ and $R_2$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.


LzLabs GmbH. Ex. 1008-371

The location of the leftmost byte of the first operand and the second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 0-31 of general registers $R_1 + 1$ and $R_2 + 1$, respectively. The contents of general registers $R_1 + 1$ and $R_2 + 1$ are treated as 32-bit unsigned binary integers.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of the registers constitute the address, and the contents of bit position 0 are ignored.

The contents of the registers just described are shown in Figure 7-61 on page 7-69.

The characters of the second operand are selected one by one for conversion, proceeding left to right. The bytes resulting from a conversion are placed at the first-operand location, proceeding left to right. The operation proceeds until the first-operand or second-operand location is exhausted or a CPU-determined number of second-operand characters have been converted.

To show the method of converting a Unicode character to a UTF-8 character, the bits of a Unicode character are identified by letters as follows:

```
Unicode Character                    111111
Bit Numbers              01234567 89012345

Identifying Bit Letters  abcdefgh ijklmnop
```

In the case of a Unicode surrogate pair, which is a character pair consisting of a character called a high surrogate followed by a character called a low surrogate, the bits are identified by letters as follows:

```
Unicode High Surrogate               111111
Bit Numbers              01234567 89012345

Identifying Bit Letters  110110ab cdefghij


Unicode Low Surrogate    11112222 22222233
Bit Numbers              67890123 45678901

Identifying Bit Letters  110111kl mnopqrst
```

Any Unicode character in the range 0000 to 007F hex is converted to a one-byte UTF-8 character as follows:

```
Unicode    00000000 0jklmnop
Character

UTF-8      0jklmnop
Character
```

Any Unicode character in the range 0080 to 07FF hex is converted to a two-byte UTF-8 character as follows:

```
Unicode    00000fgh ijklmnop
Character

UTF-8      110fghij 10klmnop
Character
```

Any Unicode character in the range 0800 to D7FF and DC00 to FFFF hex is converted to a three-byte UTF-8 character as follows:

```
Unicode    abcdefgh ijklmnop
Character

UTF-8      1110abcd 10efghij 10klmnop
Character
```

Any Unicode surrogate pair starting with a high surrogate in the range D800 to DBFF hex is converted to a four-byte UTF-8 character as follows:

```
Unicode    110110ab cdefghij 110111kl mnopqrst
Characters

UTF-8      11110uvw 10xyefgh 10ijklmn 10opqrst
Character

           where uvwxy = abcd + 1
```

The first six bits of the second Unicode character are ignored.

The second-operand location is considered exhausted when it does not contain at least two remaining bytes or at least four remaining bytes when the first two bytes are a Unicode high surrogate. The first-operand location is considered exhausted when it does not contain at least the one, two, three, or four remaining bytes required to contain the UTF-8 character resulting from the



LzLabs GmbH. Ex. 1008-372

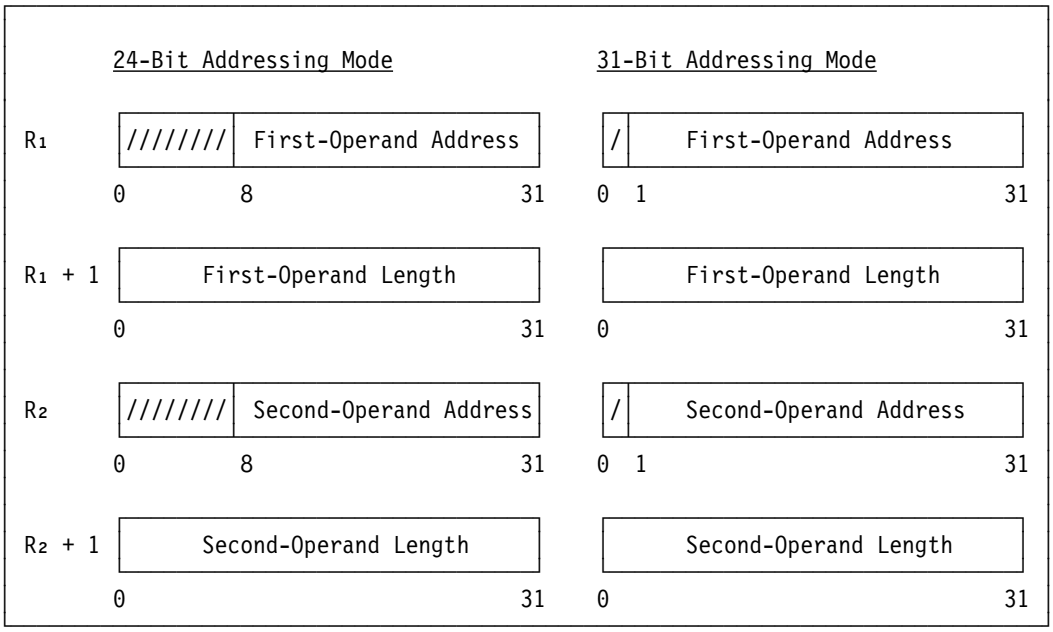



*Figure 7-61. Register Contents for CONVERT UNICODE TO UTF-8*

conversion of the next second-operand character or surrogate pair.

When the second-operand location is exhausted, condition code 0 is set. When the first-operand location is exhausted, condition code 1 is set, except that condition code 0 is set if the second-operand location also is exhausted. When a CPU-determined number of characters have been converted, condition code 3 is set.

When the operation is completed, the contents of general register $R_2 + 1$ are decremented by the number of bytes converted, and the contents of general register $R_2$ are incremented by the same number. Also, the contents of general register $R_1 + 1$ are decremented by the number of bytes placed at the first-operand location, and the contents of general register $R_1$ are incremented by the same number. When general registers $R_1$ and $R_2$ are updated, the bits in them that are not part of the address may be set to zeros or may remain unchanged.

When condition code 3 is set, the registers have been updated so that the instruction, when reexecuted, resumes at the next byte locations to be processed.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the $R_1$ register is the same register as the $R_2$ register, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portions of the operands to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

***Resulting Condition Code:***

0 Entire second operand processed
1 End of first operand reached
2 --
3 CPU-determined number of characters converted


LzLabs GmbH. Ex. 1008-373

***Program Exceptions:***

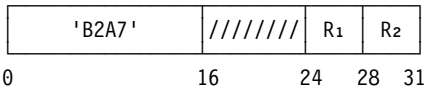
- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 1 is not installed)
- Specification

**Programming Notes:**

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the conversion. The program need not determine the number of first-operand or second-operand bytes that were processed.
2. The storage-operand references of CONVERT UNICODE TO UTF-8 may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

# CONVERT UTF-8 TO UNICODE

```
CUTFU    R1,R2      [RRE]
```




The one-, two-, three-, or four-byte UTF-8 characters of the second operand are converted to two-byte Unicode characters and placed at the first-operand location. The operation proceeds until the end of the first or second operand is reached, a CPU-determined number of characters have been converted, or an invalid UTF-8 character is encountered, whichever occurs first. The result is indicated in the condition code.

The $R_1$ and $R_2$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and the second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 0-31 of general registers $R_1 + 1$ and $R_2 + 1$, respectively. The contents of general registers $R_1 + 1$ and $R_2 + 1$ are treated as 32-bit unsigned binary integers.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of the registers constitute the address, and the contents of bit position 0 are ignored.

The contents of the registers just described are shown in Figure 7-62 on page 7-71.

The characters of the second operand are selected one by one for conversion, proceeding left to right. The bytes resulting from a conversion are placed at the first-operand location, proceeding left to right. The operation proceeds until the first-operand or second-operand location is exhausted, a CPU-determined number of second-operand characters have been converted, or an invalid UTF-8 character is encountered in the second operand.

To show the method of converting a UTF-8 character to a Unicode character, the bits of a Unicode character are identified by letters as follows:

```
Unicode Character                    111111
Bit Numbers               01234567 89012345

Identifying Bit Letters   abcdefgh ijklmnop
```

In the case of a Unicode surrogate pair, which is a character pair consisting of a character called a high surrogate followed by a character called a low surrogate, the bits are identified by letters as follows:

```
Unicode High Surrogate               111111
Bit Numbers               01234567 89012345

Identifying Bit Letters   110110ab cdefghij


Unicode Low Surrogate     11112222 22222233
Bit Numbers               67890123 45678901

Identifying Bit Letters   110111kl mnopqrst
```

When the contents of the first byte of a UTF-8 character are in the range 00 to 7F hex, the character is a one-byte character, and it is converted to a two-byte Unicode character as follows:

```
UTF-8       0jklmnop
Character

Unicode     00000000 0jklmnop
Character
```



LzLabs GmbH. Ex. 1008-374

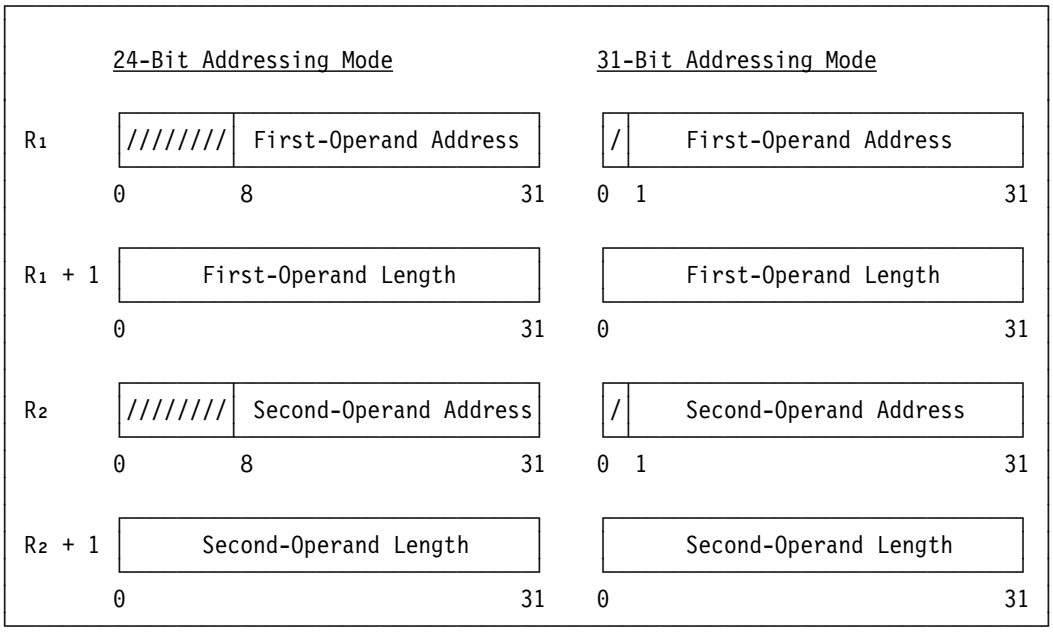



*Figure 7-62. Register Contents for CONVERT UTF-8 TO UNICODE*

When the contents of the first byte of a UTF-8 character are in the range C0 to DF hex, the character is a two-byte character, and it is converted to a two-byte Unicode character as follows:

```
UTF-8      110fghij 10klmnop
Character

Unicode    00000fgh ijklmnop
Character
```

The first two bits in the second byte of the UTF-8 character are ignored.

When the contents of the first byte of a UTF-8 character are in the range E0 to EF hex, the character is a three-byte character, and it is converted to a two-byte Unicode character as follows:

```
UTF-8      1110abcd 10efghij 10klmnop
Character

Unicode    abcdefgh ijklmnop
Character
```

The first two bits in the second and third bytes of the UTF-8 character are ignored.

When the contents of the first byte of a UTF-8 character are in the range F0 to F7 hex, the character is a four-byte character, and it is converted to two two-byte Unicode characters (a surrogate pair) as follows:

```
UTF-8      11110uvw 10xyefgh 10ijklmn 10opqrst
Character

Unicode    110110ab cdefghij 110111kl mnopqrst
Characters

where zabcd = uvwxy -1
```

The first two bits in the second, third, and fourth bytes of the UTF-8 character are ignored. The high order bit (z) produced by the subtract operation should be zero but is ignored.

The second-operand location is considered exhausted when it does not contain at least one remaining byte or when it does not contain at least the two, three, or four remaining bytes required to contain the two-, three-, or four-byte UTF-8 character indicated by the contents of the first remaining byte. The first-operand location is considered exhausted when it does not contain at least two remaining bytes or at least four remaining bytes in the case when a four-byte UTF-8 character is to be converted.

When the second-operand location is exhausted, condition code 0 is set. When the first-operand location is exhausted, condition code 1 is set, except that condition code 0 is set if the second-operand location also is exhausted. When a CPU-determined number of characters have been processed, condition code 3 is set.


LzLabs GmbH. Ex. 1008-375

When the contents of the first byte of the next UTF-8 character are in the range 80 to BF hex or F8 to FF hex, the character is invalid, and condition code 2 is set.

When the conditions for setting condition codes 1 and 2 are both met, condition code 2 is set.

When the operation is completed, the contents of general register $R_2 + 1$ are decremented by the number of bytes converted, and the contents of general register $R_2$ are incremented by the same number. Also, the contents of general register $R_1 + 1$ are decremented by the number of bytes placed at the first-operand location, and the contents of general register $R_1$ are incremented by the same number. When general registers $R_1$ and $R_2$ are updated, the bits in them that are not part of the address may be set to zeros or may remain unchanged.

When condition code 3 is set, the registers have been updated so that the instruction, when reexecuted, resumes at the next byte locations to be processed.

When condition code 2 is set, general register $R_2$ contains the address of the invalid UTF-8 character.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the $R_1$ register is the same register as the $R_2$ register, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portions of the operands to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

***Resulting Condition Code:***

0 Entire second operand processed
1 End of first operand reached
2 Invalid UTF-8 character
3 CPU-determined number of characters processed

***Program Exceptions:***

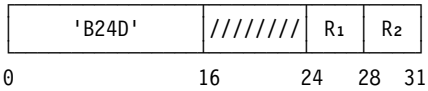
- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 1 is not installed)
- Specification

**Programming Notes:**

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the conversion. The program need not determine the number of first-operand or second-operand bytes that were processed.
2. Bits 0 and 1 of the continuation bytes of multiple-byte UTF-8 characters are not checked in order to speed up the conversion. Thus, invalid continuation bytes are not detected.
3. The storage-operand references of CONVERT UTF-8 TO UNICODE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)

## COPY ACCESS

```
CPYA      R1,R2      [RRE]
```




The contents of access register $R_2$ are placed in access register $R_1$.

***Condition Code:*** The code remains unchanged.

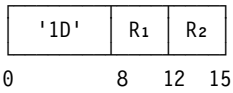
***Program Exceptions:*** None.



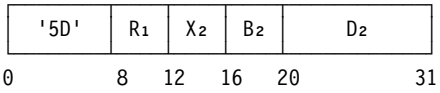
LzLabs GmbH. Ex. 1008-376

## DIVIDE

DR R1,R2 [RR]




D R1,D2(X2,B2) [RX]




The 64-bit first operand (the dividend) is divided by the 32-bit second operand (the divisor), and the 32-bit remainder and quotient are placed at the first-operand location.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The dividend is treated as a 64-bit signed binary integer. The leftmost 32 bits of the dividend are in general register $R_1$, and the rightmost 32 bits are in general register $R_1 + 1$. The divisor, remainder, and quotient are treated as 32-bit signed binary integers. The remainder is placed in general register $R_1$, and the quotient is placed in general register $R_1 + 1$.

The sign of the quotient is determined by the rules of algebra, and the remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive.

When the divisor is zero, or when the magnitudes of the dividend and divisor are such that the quotient cannot be expressed by a 32-bit signed binary integer, a fixed-point-divide exception is recognized. This includes the case of division of zero by zero.

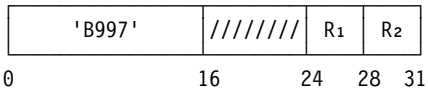
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

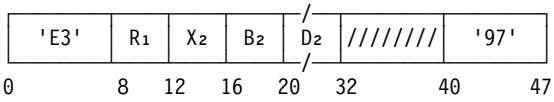
- Access (fetch, operand 2 of D only)
- Fixed-point divide
- Specification

## DIVIDE LOGICAL

DLR R1,R2 [RRE]




DL R1,D2(X2,B2) [RXE]




The 64-bit first operand (the dividend) is divided by the 32-bit second operand (the divisor), and the 32-bit remainder and quotient are placed at the first-operand location.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The dividend is treated as a 64-bit unsigned binary integer. The leftmost 32 bits of the dividend are in general register $R_1$, and the rightmost 32 bits are in general register $R_1 + 1$.

The divisor, remainder, and quotient are treated as 32-bit unsigned binary integers. The remainder is placed in general register $R_1$, and the quotient is placed in general register $R_1 + 1$.

When the divisor is zero, or when the magnitudes of the dividend and divisor are such that the quotient cannot be expressed as a 32-bit unsigned binary, a fixed-point-divide exception is recognized. This includes the case of division of zero by zero.

***Condition Code:*** The code remains unchanged.

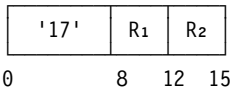
***Program Exceptions:***

- Access (fetch, operand 2 of DL only)
- Fixed-point divide
- Operation (if z/Architecture is not installed)
- Specification

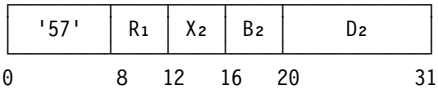

LzLabs GmbH. Ex. 1008-377

# EXCLUSIVE OR

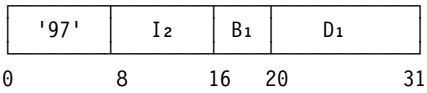
XR R1,R2 [RR]



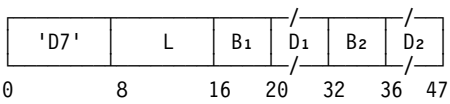

X R1,D2(X2,B2) [RX]




XI D1(B1),I2 [SI]




XC D1(L,B1),D2(B2) [SS]




The EXCLUSIVE OR of the first and second operands is placed at the first-operand location.

The connective EXCLUSIVE OR is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the bits in the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

For EXCLUSIVE OR (XC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

For EXCLUSIVE OR (XI), the first operand is one byte in length, and only one byte is stored.

***Resulting Condition Code:***

0 Result zero
1 Result not zero
2 --
3 --

***Program Exceptions:***

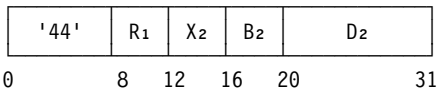
- Access (fetch, operand 2, X and XC; fetch and store, operand 1, XI and XC)

**Programming Notes:**

1. An example of the use of the EXCLUSIVE OR instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. EXCLUSIVE OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary switches.
3. A field EXCLUSIVE-ORed with itself becomes all zeros.
4. For EXCLUSIVE OR (XR), the sequence A EXCLUSIVE-OR B, B EXCLUSIVE-OR A, A EXCLUSIVE-OR B results in the exchange of the contents of A and B without the use of an additional general register.
5. Accesses to the first operand of EXCLUSIVE OR (XI) and EXCLUSIVE OR (XC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, EXCLUSIVE OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in “Multiprogramming and Multiprocessing Examples” in Appendix A, “Number Representation and Instruction-Use Examples.”

# EXECUTE

EX R1,D2(X2,B2) [RX]




The single instruction at the second-operand address is modified by the contents of general register R1, and the resulting instruction, called the target instruction, is executed.

When the R1 field is not zero, bits 8-15 of the instruction designated by the second-operand address are ORed with bits 24-31 of general reg-



LzLabs GmbH. Ex. 1008-378

ister $R_1$. The ORing does not change either the contents of general register $R_1$ or the instruction in storage, and it is effective only for the interpretation of the instruction to be executed. When the $R_1$ field is zero, no ORing takes place.

The target instruction may be two, four, or six bytes in length. The execution and exception handling of the target instruction are exactly as if the target instruction were obtained in normal sequential operation, except for the instruction address and the instruction-length code.

The instruction address in the current PSW is increased by the length of EXECUTE. This updated address and the instruction-length code of EXECUTE are used, for example, as part of the link information when the target instruction is BRANCH AND LINK. When the target instruction is a successful branching instruction, the instruction address in the current PSW is replaced by the branch address specified by the target instruction.

When the target instruction is in turn EXECUTE, an execute exception is recognized.

The effective address of EXECUTE must be even; otherwise, a specification exception is recognized. When the target instruction is two or three halfwords in length but can be executed without fetching its second or third halfword, it is unpredictable whether access exceptions are recognized for the unused halfwords. Access exceptions are not recognized for the second-operand address when the address is odd.

The second-operand address of EXECUTE is an instruction address rather than a logical address; thus, the target instruction is fetched from the primary address space when in the primary-space, secondary-space, or access-register mode.

***Condition Code:*** The code may be set by the target instruction.

***Program Exceptions:***

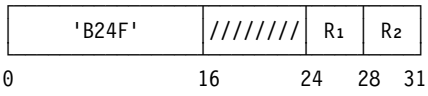
- Access (fetch, target instruction)
- Execute
- Specification

**Programming Notes:**

1. An example of the use of the EXECUTE instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. The ORing of eight bits from the general register with the designated instruction permits the indirect specification of the length, index, mask, immediate-data, register, or extended-op-code field.
3. The fetching of the target instruction is considered to be an instruction fetch for purposes of program-event recording and for purposes of reporting access exceptions.
4. An access or specification exception may be caused by EXECUTE or by the target instruction.
5. When an interruptible instruction is made the target of EXECUTE, the program normally should not designate any register updated by the interruptible instruction as the $R_1$, $X_2$, or $B_2$ register for EXECUTE. Otherwise, on resumption of execution after an interruption, or if the instruction is refetched without an interruption, the updated values of these registers will be used in the execution of EXECUTE. Similarly, the program should normally not let the destination field in storage of an interruptible instruction include the location of EXECUTE, since the new contents of the location may be interpreted when resuming execution.

# EXTRACT ACCESS

```
EAR     R1,R2     [RRE]
```




The contents of access register $R_2$ are placed in general register $R_1$.

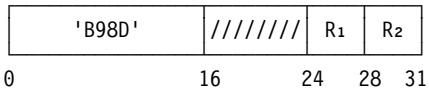
***Condition Code:*** The code remains unchanged.

***Program Exceptions:*** None.


LzLabs GmbH. Ex. 1008-379

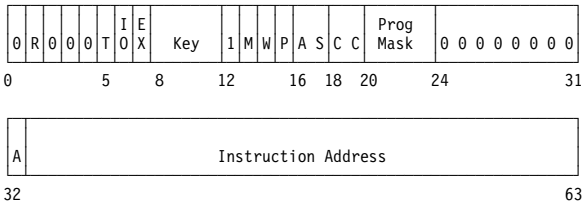
## EXTRACT PSW

EPSW R1,R2 [RRE]




Bits 0-31 of the current PSW are placed in the first operand. Subsequently, bit 32 of the current PSW is placed in bit position 0 of the second operand, and bits 1-31 of the second operand are set to zeros. The action associated with the second operand is not performed if the R2 field is zero.

The PSW has the following format:




***Condition Code:*** The code remains unchanged.

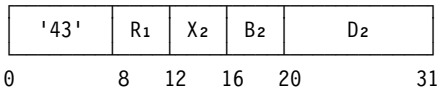
***Program Exceptions:***

- Operation (if z/Architecture is not installed)

**Programming Note:** When general register R2 is loaded, bit positions 1-31 are loaded with all zeros instead of with an instruction address.

## INSERT CHARACTER

IC R1,D2(X2,B2) [RX]




The byte at the second-operand location is inserted into bit positions 24-31 of general register R1. The remaining bits in the register remain unchanged.

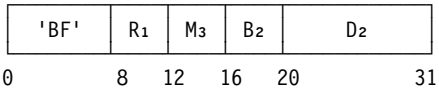
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2)

## INSERT CHARACTERS UNDER MASK

ICM R1,M3,D2(B2) [RS]




Bytes from contiguous locations beginning at the second-operand address are inserted into general register R1 under control of a mask.

The contents of the M3 field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of general register R1. The byte positions corresponding to ones in the mask are filled, left to right, with bytes from successive storage locations beginning at the second-operand address. When the mask is not zero, the length of the second operand is equal to the number of ones in the mask. The bytes in the general register corresponding to zeros in the mask remain unchanged.

The resulting condition code is based on the mask and on the value of the bits inserted. When the mask is zero or when all inserted bits are zeros, the condition code is set to 0. When the inserted bits are not all zeros, the code is set according to the leftmost bit of the storage operand: if this bit is one, the code is set to 1; if this bit is zero, the code is set to 2.

When the mask is not zero, exceptions associated with storage-operand access are recognized only for the number of bytes specified by the mask. When the mask is zero, access exceptions are recognized for one byte at the second-operand address.

***Resulting Condition Code:***

0 All inserted bits zeros, or mask bits all zeros
1 Leftmost inserted bit one
2 Leftmost inserted bit zero, and not all inserted bits zeros
3 --

***Program Exceptions:***

- Access (fetch, operand 2)



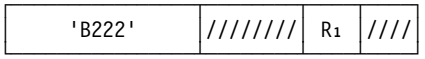
LzLabs GmbH. Ex. 1008-380

**Programming Notes:**

1. Examples of the use of the INSERT CHARACTERS UNDER MASK instruction are given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. The condition code for INSERT CHARACTERS UNDER MASK is defined such that, when the mask is 1111, the instruction causes the same condition code to be set as for LOAD AND TEST. Thus, the instruction may be used as a storage-to-register load-and-test operation.
3. INSERT CHARACTERS UNDER MASK with a mask of 1111 or 0001 performs a function similar to that of a LOAD (L) or INSERT CHARACTER (IC) instruction, respectively, with the exception of the condition-code setting. However, the performance of INSERT CHARACTERS UNDER MASK may be slower.

## INSERT PROGRAM MASK

```
IPM     R1                [RRE]
```



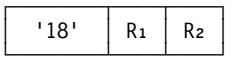

The condition code and program mask from the current PSW are inserted into bit positions 2 and 3 and 4-7, respectively, of general register $R_1$. Bits 0 and 1 of the register are set to zeros; bits 8-31 are left unchanged.

***Condition Code:*** The code remains unchanged.

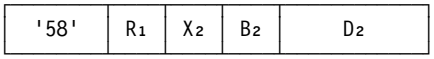
***Program Exceptions:*** None.

## LOAD

```
LR      R1,R2      [RR]
```




```
L       R1,D2(X2,B2)      [RX]
```




The second operand is placed unchanged at the first-operand location.

***Condition Code:*** The code remains unchanged.

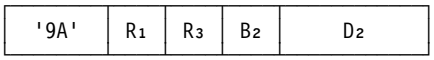
***Program Exceptions:***

- Access (fetch, operand 2 of L only)

**Programming Note:** An example of the use of the LOAD instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”

## LOAD ACCESS MULTIPLE

```
LAM     R1,R3,D2(B2)      [RS]
```




The set of access registers starting with access register $R_1$ and ending with access register $R_3$ is loaded from the locations designated by the second-operand address.

The storage area from which the contents of the access registers are obtained starts at the location designated by the second-operand address and continues through as many storage words as the number of access registers specified. The access registers are loaded in ascending order of their register numbers, starting with access register $R_1$ and continuing up to and including access register $R_3$, with access register 0 following access register 15.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

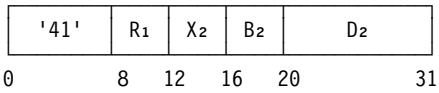
***Program Exceptions:***

- Access (fetch, operand 2)
- Specification


LzLabs GmbH. Ex. 1008-381

## LOAD ADDRESS

```
LA      R1,D2(X2,B2)      [RX]
```




The address specified by the $X_2$, $B_2$, and $D_2$ fields is placed in general register $R_1$. The address computation follows the rules for address arithmetic.

In the 24-bit addressing mode, the address is placed in bit positions 8-31, and bits 0-7 are set to zeros. In the 31-bit addressing mode, the address is placed in bit positions 1-31, and bit 0 is set to zero.

No storage references for operands take place, and the address is not inspected for access exceptions.

***Condition Code:*** The code remains unchanged.

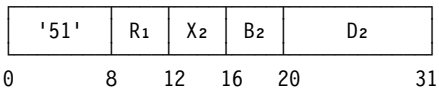
***Program Exceptions:*** None.

**Programming Notes:**

1. An example of the use of the LOAD ADDRESS instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. LOAD ADDRESS may be used to increment the rightmost bits of a general register, other than register 0, by the contents of the $D_2$ field of the instruction. The register to be incremented should be designated by $R_1$ and by either $X_2$ (with $B_2$ set to zero) or $B_2$ (with $X_2$ set to zero). The instruction updates 24 bits in the 24-bit addressing mode and updates 31 bits in the 31-bit addressing mode.

## LOAD ADDRESS EXTENDED

```
LAE     R1,D2(X2,B2)      [RX]
```




The address specified by the $X_2$, $B_2$, and $D_2$ fields is placed in general register $R_1$. Access register $R_1$ is loaded with a value that depends on the current value of the address-space-control bits, bits 16 and 17 of the PSW. If the address-space-control bits are 01 binary, the value placed in the access register also depends on whether the $B_2$ field is zero or nonzero.

The address computation follows the rules for address arithmetic. In the 24-bit addressing mode, the address is placed in bit positions 8-31 of general register $R_1$, and bits 0-7 are set to zeros. In the 31-bit addressing mode, the address is placed in bit positions 1-31 of general register $R_1$, and bit 0 is set to zero.

The value placed in access register $R_1$ is as shown in the following table:

| PSW Bits 16 and 17 | Value Placed in Access Register $R_1$ |
|---|---|
| 00 | 00000000 hex (zeros in bit positions 0-31) |
| 10 | 00000001 hex (zeros in bit positions 0-30 and one in bit position 31) |
| 01 | If $B_2$ field is zero: 00000000 hex (zeros in bit positions 0-31)<br>If $B_2$ field is nonzero: Contents of access register $B_2$ |
| 11 | 00000002 hex (zeros in bit positions 0-29 and 31, and one in bit position 30) |

However, when PSW bits 16 and 17 are 01 binary and the $B_2$ field is nonzero, bit positions 0-6 of access register $B_2$ must contain all zeros; otherwise, the results in general register $R_1$ and access register $R_1$ are unpredictable.

No storage references for operands take place, and the address is not inspected for access exceptions.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:*** None.

**Programming Notes:**

1. When DAT is on, the different values of the address-space-control bits correspond to translation modes as follows:



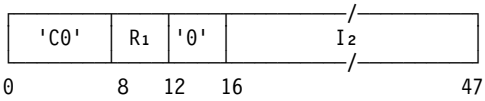
LzLabs GmbH. Ex. 1008-382

| PSW Bits 16 and 17 | Translation Mode |
|---|---|
| 00 | Primary-space mode |
| 10 | Secondary-space mode |
| 01 | Access-register mode |
| 11 | Home-space mode |

2. In the access-register mode, the value 00000000 hex in an access register designates the primary address space, and the value 00000001 hex designates the secondary address space. The value 00000002 hex designates the home address space if the control program assigns entry 2 of the dispatchable-unit access list as designating the home address space and places a zero access-list-entry sequence number (ALESN) in that entry.

## LOAD ADDRESS RELATIVE LONG

```
LARL     R1,I2     [RIL]
```




The address specified by the $I_2$ field is placed in general register $R_1$. The address computation follows the rules for the branch address of BRANCH RELATIVE ON CONDITION LONG and BRANCH RELATIVE AND SAVE LONG.

In the 24-bit addressing mode, the address is placed in bit positions 8-31, and bits 0-7 are set to zeros. In the 31-bit addressing mode, the address is placed in bit positions 1-31, and bit 0 is set to zero.

The contents of the $I_2$ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the computed address.

No storage references for operands take place, and the address is not inspected for access exceptions.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

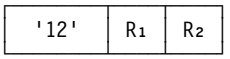
- Operation (if z/Architecture is not installed)

**Programming Notes:**

1. Only even addresses (halfword addresses) can be generated. If an odd address is desired, LOAD ADDRESS can be used to add one to an address formed by LOAD ADDRESS RELATIVE LONG.
2. When LOAD ADDRESS RELATIVE LONG is the target of EXECUTE, the address produced is relative to the location of the LOAD ADDRESS RELATIVE LONG instruction, not of the EXECUTE instruction. This is consistent with the operation of the relative-branch instructions.

## LOAD AND TEST

```
LTR      R1,R2     [RR]
```




The second operand is placed unchanged at the first-operand location, and the sign and magnitude of the second operand, treated as a 32-bit signed binary integer, are indicated in the condition code.

***Resulting Condition Code:***

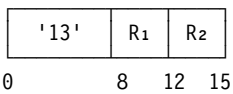
0 Result zero
1 Result less than zero
2 Result greater than zero
3 --

***Program Exceptions:*** None.

**Programming Note:** When the $R_1$ and $R_2$ fields designate the same register, the operation is equivalent to a test without data movement.

## LOAD COMPLEMENT

```
LCR      R1,R2     [RR]
```




The two's complement of the second operand is placed at the first-operand location. The second


LzLabs GmbH. Ex. 1008-383

operand and result are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

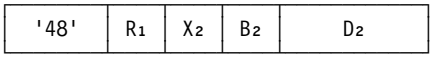
***Program Exceptions:***

- Fixed-point overflow

**Programming Note:** The operation complements all numbers. Zero and the maximum negative number remain unchanged. An overflow condition occurs when the maximum negative number is complemented.

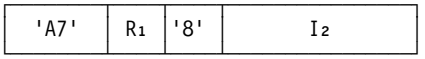
## LOAD HALFWORD

```
LH      R1,D2(X2,B2)      [RX]
```




## LOAD HALFWORD IMMEDIATE

```
LHI     R1,I2     [RI]
```




The second operand is sign extended and placed at the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The second operand is extended to 32 bits by setting each of the 16 left-most bit positions equal to the sign bit of the two-byte operand.

***Condition Code:*** The code remains unchanged.

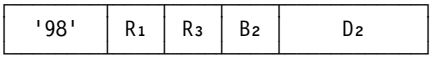
***Program Exceptions:***

- Access (fetch, operand 2 of LH only)
- Operation (LHI if the immediate-and-relative-instruction facility is not installed)

**Programming Note:** An example of the use of the LOAD HALFWORD instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”

## LOAD MULTIPLE

```
LM      R1,R3,D2(B2)      [RS]
```




The set of general registers starting with general register $R_1$ and ending with general register $R_3$ is loaded from storage beginning at the location designated by the second-operand address and continuing through as many locations as needed.

The general registers are loaded in the ascending order of their register numbers, starting with general register $R_1$ and continuing up to and including general register $R_3$, with general register 0 following general register 15.

***Condition Code:*** The code remains unchanged.

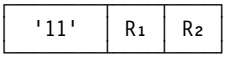
***Program Exceptions:***

- Access (fetch, operand 2)

**Programming Note:** All combinations of register numbers specified by $R_1$ and $R_3$ are valid. When the register numbers are equal, only four bytes are transmitted. When the number specified by $R_3$ is less than the number specified by $R_1$, the register numbers wrap around from 15 to 0.

## LOAD NEGATIVE

```
LNR     R1,R2     [RR]
```




The two's complement of the absolute value of the second operand is placed at the first-operand location. The second operand and result are treated as 32-bit signed binary integers.



LzLabs GmbH. Ex. 1008-384

***Resulting Condition Code:***

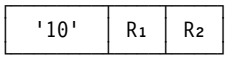
0 Result zero
1 Result less than zero
2 --
3 --

***Program Exceptions:*** None.

**Programming Note:** The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged.

## LOAD POSITIVE

LPR R1,R2 [RR]




The absolute value of the second operand is placed at the first-operand location. The second operand and the result are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

***Resulting Condition Code:***

0 Result zero; no overflow
1 --
2 Result greater than zero; no overflow
3 Overflow

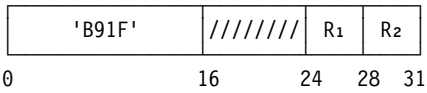
***Program Exceptions:***

- Fixed-point overflow

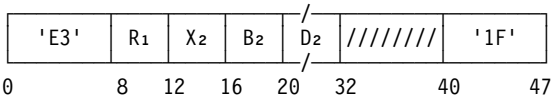
**Programming Note:** The operation complements negative numbers; positive numbers and zero remain unchanged. An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged.

## LOAD REVERSED

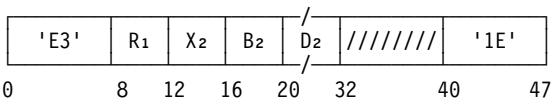
LRVR R1,R2 [RRE]




LRVH R1,D2(X2,B2) [RXE]




LRV R1,D2(X2,B2) [RXE]




The second operand is placed at the first-operand location with the left-to-right sequence of the bytes reversed.

For LOAD REVERSED (LRVH), the second operand is two bytes, the result is placed in bit positions 16-31 of general register R1, and bits 0-15 of the register remain unchanged. For LOAD REVERSED (LRVR, LRV), the second operand is four bytes.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2 of LRVH and LRV only)
- Operation (if z/Architecture is not installed)

**Programming Notes:**

1. The instruction can be used to convert two or four bytes from a “little-endian” format to a “big-endian” format, or vice versa. In the big-endian format, the bytes in a left-to-right sequence are in the order most significant to least significant. In the little-endian format, the bytes are in the order least significant to most significant. For example, the bytes ABCD in the big-endian format are DCBA in the little-endian format.
2. LOAD REVERSED (LRVR) can be used with a two-byte value already in a register as shown in the following example. In the


LzLabs GmbH. Ex. 1008-385

example, the two bytes of interest are in bit positions 16-31 of the R1 register.

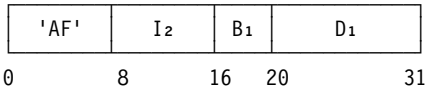
```
LRVR   R1,R1
SRA    R1,16
```

The LOAD REVERSED instruction places the two bytes of interest in bit positions 0-15 of the register, with the order of the bytes reversed. The SHIFT RIGHT SINGLE (SRA) instruction shifts the two bytes to bit positions 16-31 of the register and extends them on their left, in bit positions 0-15, with their sign bit. The instruction SHIFT RIGHT SINGLE LOGICAL (SRL) should be used, instead, if the two bytes of interest are unsigned.

3. The storage-operand references of LOAD REVERSED may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

## MONITOR CALL

MC  $D_1(B_1),I_2$  [SI]




A program interruption is caused if the appropriate monitor-mask bit in control register 8 is one.

The monitor-mask bits are in bit positions 16-31 of control register 8, which correspond to monitor classes 0-15, respectively.

Bit positions 12-15 in the $I_2$ field contain a binary number specifying one of 16 monitoring classes. When the monitor-mask bit corresponding to the class specified by the $I_2$ field is one, a monitor-event program interruption occurs. The contents of the $I_2$ field are stored at location 149, with zeros stored at location 148. Bit 9 of the program-interruption code is set to one.

The first-operand address is not used to address data; instead, the address specified by the $B_1$ and $D_1$ fields forms the monitor code, which is placed in the word at location 156. Address computation follows the rules of address arithmetic; in the 24-bit addressing mode, bits 0-7 are set to zeros; in the 31-bit addressing mode, bit 0 is set to zero.

When the monitor-mask bit corresponding to the class specified by bits 12-15 of the instruction is zero, no interruption occurs, and the instruction is executed as a no-operation.

Bit positions 8-11 of the instruction must contain zeros; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Monitor event
- Specification

**Programming Notes:**

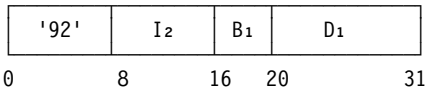
1. MONITOR CALL provides the capability for passing control to a monitoring program when selected points are reached in the monitored program. This is accomplished by implanting MONITOR CALL instructions at the desired points in the monitored program. This function may be useful in performing various measurement functions; specifically, tracing information can be generated indicating which programs were executed, counting information can be generated indicating how often particular programs were used, and timing information can be generated indicating the amount of time a particular program required for execution.
2. The monitor masks provide a means of disallowing all monitor-event program interruptions or allowing monitor-event program interruptions for all or selected classes.
3. The monitor code provides a means of associating descriptive information, in addition to the class number, with each MONITOR CALL. Without the use of a base register, up to 4,096 distinct monitor codes can be associated with a monitoring interruption. With the base register designated by a nonzero value in the $B_1$ field, each monitoring interruption can be identified by a 24-bit code in the 24-bit addressing mode or a 31-bit code in the 31-bit addressing mode.



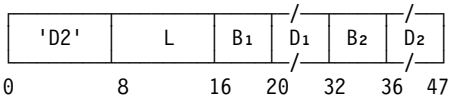
LzLabs GmbH. Ex. 1008-386

## MOVE

MVI D1(B1),I2 [SI]




MVC D1(L,B1),D2(B2) [SS]




The second operand is placed at the first-operand location.

For MOVE (MVC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand byte.

For MOVE (MVI), the first operand is one byte in length, and only one byte is stored.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

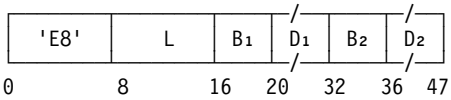
- Access (fetch, operand 2 of MVC; store, operand 1, MVI and MVC)

**Programming Notes:**

1. Examples of the use of the MOVE instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. It is possible to propagate one byte through an entire field by having the first operand start one byte to the right of the second operand.

## MOVE INVERSE

MVCIN D1(L,B1),D2(B2) [SS]




The second operand is placed at the first-operand location with the left-to-right sequence of the bytes inverted.

The first-operand address designates the leftmost byte of the first operand. The second-operand address designates the rightmost byte of the second operand. Both operands have the same length.

The result is obtained as if the second operand were processed from right to left and the first operand from left to right. The second operand may wrap around from location 0 to location $2^{24}$ - 1 in the 24-bit addressing mode, or, in the 31-bit addressing mode, to location $2^{31}$ - 1. The first operand may, in the 24-bit addressing mode, wrap around from location $2^{24}$ - 1 to location 0, or, in the 31-bit addressing mode, from location $2^{31}$ - 1 to location 0.

When the operands overlap by more than one byte, the contents of the overlapped portion of the result field are unpredictable.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

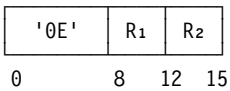
- Access (fetch, operand 2; store, operand 1)

**Programming Notes:**

1. An example of the use of the MOVE INVERSE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The contents of each byte moved remain unchanged.
3. MOVE INVERSE is the only SS-format instruction for which the second-operand address designates the rightmost, instead of the leftmost, byte of the second operand.
4. The storage-operand references for MOVE INVERSE may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

## MOVE LONG

MVCL R1,R2 [RR]




The second operand is placed at the first-operand location, provided overlapping of operand


LzLabs GmbH. Ex. 1008-387

locations would not affect the final contents of the first-operand location. The remaining rightmost byte positions, if any, of the first-operand location are filled with padding bytes.

The $R_1$ and $R_2$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively. The number of bytes in the first-operand and second-operand locations is specified by unsigned binary integers in bit positions 8-31 of general registers $R_1 + 1$ and $R_2 + 1$, respectively. Bit positions 0-7 of register $R_2 + 1$ contain the padding byte. The contents of bit positions 0-7 of register $R_1 + 1$ are ignored.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 8-31 of registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of registers $R_1$ and $R_2$ constitute the address, and the contents of bit position 0 are ignored.

The contents of the registers just described are shown in Figure 7-63 on page 7-85.

The result is obtained as if the movement starts at the left end of both fields and proceeds to the right, byte by byte. The operation is ended when the number of bytes specified by bits 8-31 of general register $R_1 + 1$ have been moved into the first-operand location. If the second operand is shorter than the first operand, the remaining rightmost bytes of the first-operand location are filled with the padding byte.

As part of the execution of the instruction, the values of the two length fields are compared for the setting of the condition code, and a check is made for destructive overlap of the operands. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it, assuming the inspection for overlap is performed by the use of logical operand addresses. When the operands overlap destructively, no movement takes place, and condition code 3 is set.

Operands do not overlap destructively, and movement is performed, if the leftmost byte of the first operand does not coincide with any of the second-operand bytes participating in the operation other than the leftmost byte of the second operand. When an operand wraps around from location $2^{24} - 1$ (or $2^{31} - 1$) to location 0, operand bytes in locations up to and including $2^{24} - 1$ (or $2^{31} - 1$) are considered to be to the left of bytes in locations from 0 up.

In the 24-bit addressing mode, wraparound is from location $2^{24} - 1$ to location 0; in the 31-bit addressing mode, wraparound is from location $2^{31} - 1$ to location 0.

In the access-register mode, the contents of access register $R_1$ and access register $R_2$ are compared. If the $R_1$ or $R_2$ field is zero, 32 zeros are used rather than the contents of access register 0. If all 32 bits of the compared values are equal, then the destructive overlap test is made. If all 32 bits of the compared values are not equal, destructive overlap is declared not to exist. If, for this case, the operands actually overlap in real storage, it is unpredictable whether the result reflects the overlap condition.

When the length specified by bit positions 8-31 of general register $R_1 + 1$ is zero, no movement takes place, and condition code 0 or 1 is set to indicate the relative values of the lengths.

The execution of the instruction is interruptible. When an interruption occurs other than one that follows termination, the contents of general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes moved, and the contents of general registers $R_1$ and $R_2$ are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. The leftmost bits which are not part of the address in general registers $R_1$ and $R_2$ are set to zeros; the contents of bit positions 0-7 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged; and the condition code is unpredictable. If the operation is interrupted during padding, the length field in general register $R_2 + 1$ is 0, the address in general register $R_2$ is incremented by the original contents of general register $R_2 + 1$, and general



LzLabs GmbH. Ex. 1008-388

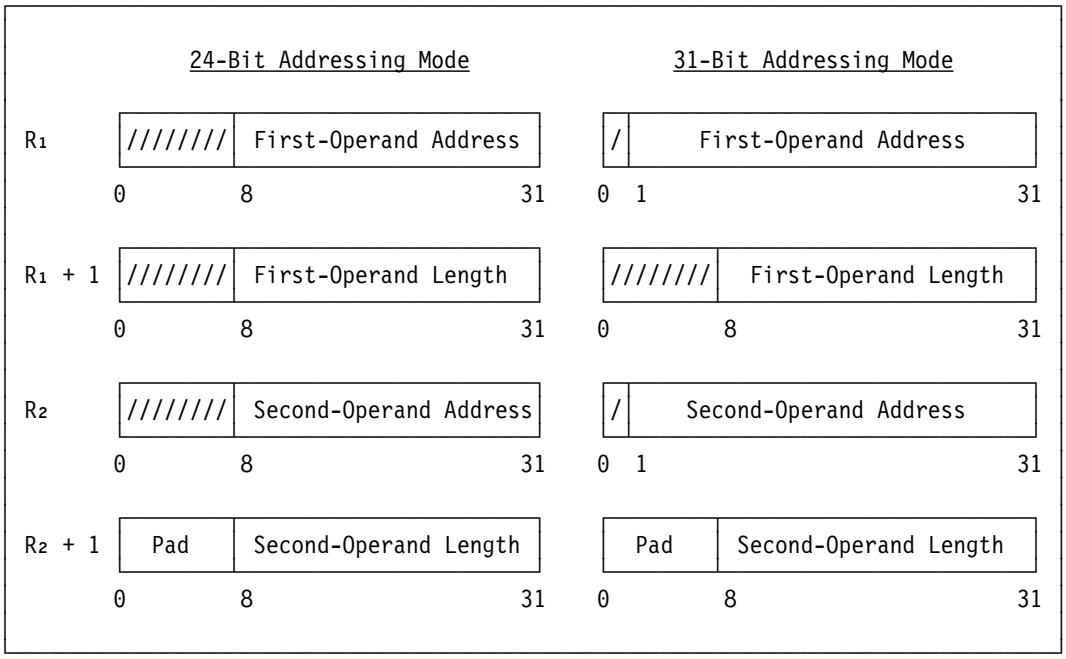



*Figure 7-63. Register Contents for MOVE LONG*

registers $R_1$ and $R_1 + 1$ reflect the extent of the padding operation.

When the first-operand location includes the location of the instruction or of EXECUTE, the instruction may be refetched from storage and reinterpreted even in the absence of an interruption during execution. The exact point in the execution at which such a refetch occurs is unpredictable.

Padding byte values of B0 hex and B8 hex may be used during the nonpadding part of the operation by some models, in certain cases, as an indication of whether the movement should be performed bypassing the cache or using the cache, respectively. Thus, a padding byte of B0 hex indicates no intention to reference the destination area after the move, and a padding byte of B8 hex indicates an intention to reference the destination area.

For the nonpadding part of the operation, accesses to the operands for MOVE LONG are single-access references. These accesses do not necessarily appear to occur in a left-to-right direction as observed by other CPUs and by channel programs, unless the padding byte is B1 hex. During the nonpadding part of the operation, operands appear to be accessed doubleword concurrent as observed by other CPUs, provided that both operands start on doubleword boundaries, are an integral number of doublewords in length, and do not overlap.

As observed by other CPUs and by channel programs, that portion of the first operand which is filled with the padding byte is not necessarily stored into in a left-to-right direction and may appear to be stored into more than once.

At the completion of the operation, the length in general register $R_1 + 1$ is decremented by the number of bytes stored at the first-operand location, and the address in general register $R_1$ is incremented by the same amount. The length in general register $R_2 + 1$ is decremented by the number of bytes moved out of the second-operand location, and the address in general register $R_2$ is incremented by the same amount. The leftmost bits which are not part of the address in general registers $R_1$ and $R_2$ are set to zeros, even when one or both of the original length values are zeros or when condition code 3 is set. The contents of bit positions 0-7 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged.

When condition code 3 is set, no exceptions associated with operand access are recognized. When the length of an operand is zero, no access


LzLabs GmbH. Ex. 1008-389

exceptions for that operand are recognized. Similarly, when the second operand is longer than the first operand, access exceptions are not recognized for the part of the second-operand field that is in excess of the first-operand field. For operands longer than 2K bytes, access exceptions are not recognized for locations more than 2K bytes beyond the current location being processed. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Also, when the $R_1$ field is odd, PER storage-alteration events are not recognized, and no change bits are set.

***Resulting Condition Code:***

0 Operand lengths equal; no destructive overlap
1 First-operand length low; no destructive overlap
2 First-operand length high; no destructive overlap
3 No movement performed because of destructive overlap

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Specification

**Programming Notes:**

1. An example of the use of the MOVE LONG instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

2. MOVE LONG may be used for clearing storage by setting the padding byte to zero and the second-operand length to zero. On most models, this is the fastest instruction for clearing storage areas in excess of 256 bytes. However, the stores associated with this clearing may be multiple-access stores and should not be used to clear an area if the possibility exists that another CPU or a channel program will attempt to access and use the area as soon as it appears to be zero. For more details, see "Storage-Operand Consistency" on page 5-87.

3. The program should avoid specification of a length for either operand which would result in an addressing exception. Addressing (and also protection) exceptions may result in termination of the entire operation, not just the current unit of operation. The termination may be such that the contents of all result fields are unpredictable; in the case of MOVE LONG, this includes the condition code and the two even-odd general-register pairs, as well as the first-operand location in main storage. The following are situations that have actually occurred on one or more models:

   a. When a protection exception occurs on a 4K-byte block of a first operand which is several blocks in length, stores to the protected block are suppressed. However, the move continues into the subsequent blocks of the first operand, which are not protected. Similarly, an addressing exception on a block does not necessarily suppress processing of subsequent blocks which are available.

   b. Some models may update the general registers only when an external, I/O, repressible machine-check, or restart interruption occurs, or when a program interruption occurs for which it is required to nullify or suppress a unit of operation. Thus, if, after a move into several blocks of the first operand, an addressing or protection exception occurs, the general registers may remain unchanged.

4. When the first-operand length is zero, the operation consists in setting the condition code and setting the leftmost bits of general registers $R_1$ and $R_2$ to zero.

5. When the contents of the $R_1$ and $R_2$ fields are the same, the contents of the designated registers are incremented or decremented only by the number of bytes moved, not by twice the number of bytes moved. Condition code 0 is set.

6. The following is a detailed description of those cases in which movement takes place, that is, where destructive overlap does not exist.

   In the access-register mode, the contents of the access registers used are called the effective space designations. When the effective space designations are not equal, destructive overlap is declared not to exist and movement occurs. When the effective space designations are the same or when not in the access-register mode, then the following cases apply.



LzLabs GmbH. Ex. 1008-390

Depending on whether the second operand wraps around from location $2^{24} - 1$ to location 0, or, in the 31-bit addressing mode, from location $2^{31} - 1$ to location 0, movement takes place in the following cases:

a. When the second operand does not wrap around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, *or* if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.

b. When the second operand wraps around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, *and* if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.

The rightmost second-operand byte is determined by using the smaller of the first-operand and second-operand lengths.

When the second-operand length is one or zero, destructive overlap cannot exist.

7. Special precautions should be taken if MOVE LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.

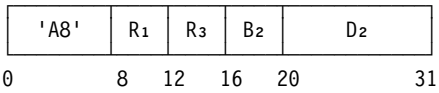
8. Since the execution of MOVE LONG is interruptible, the instruction cannot be used for situations where the program must rely on uninterrupted execution of the instruction. Similarly, the program should normally not let the first operand of MOVE LONG include the location of the instruction or of EXECUTE because the new contents of the location may be interpreted for a resumption after an interruption, or the instruction may be refetched without an interruption.

9. Further programming notes concerning interruptible instructions are included in “Interruptible Instructions” in Chapter 5, “Program Execution.”

10. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

## MOVE LONG EXTENDED

```
MVCLE   R1,R3,D2(B2)      [RS]
```




All or part of the third operand is placed at the first-operand location. The remaining rightmost byte positions, if any, of the first-operand location are filled with padding bytes. The operation proceeds until the end of the first-operand location is reached or a CPU-determined number of bytes have been placed at the first-operand location, whichever occurs first. The result is indicated in the condition code.

The $R_1$ and $R_3$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and third operand is designated by the contents of general registers $R_1$ and $R_3$, respectively. The number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 0-31 of general registers $R_1 + 1$ and $R_3 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers.

The handling of the addresses in general registers $R_1$ and $R_3$ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_3$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general registers $R_1$ and $R_3$ constitute the address, and the contents of bit position 0 are ignored.

The second-operand address is not used to address data; instead, the rightmost eight bits of the second-operand address, bits 24-31, are the padding byte. Bits 0-23 of the second-operand address are ignored.

The contents of the registers and address just described are shown in Figure 7-64 on page 7-88.


LzLabs GmbH. Ex. 1008-391

The result is obtained as if the movement starts at the left end of both fields and proceeds to the right, byte by byte. The operation is ended when the number of bytes specified in general register $R_1 + 1$ have been placed at the first-operand location or when a CPU-determined number of bytes have been placed, whichever occurs first. If the third operand is shorter than the first operand, the remaining rightmost bytes of the first-operand location are filled with the padding byte.

When the operation is completed because the end of the first operand has been reached, the condition code is set to 0 if the two operand lengths are equal, it is set to 1 if the first-operand length is less than the third-operand length, or it is set to 2 if the first-operand length is greater than the third-operand length. When the operation is completed because a CPU-determined number of bytes have been moved without reaching the end of the first operand, condition code 3 is set.

No test is made for destructive overlap, and the results in the first-operand location are unpredictable when destructive overlap exists. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it.

Operands do not overlap destructively if the leftmost byte of the first operand does not coincide with any of the third-operand bytes participating in the operation other than the leftmost byte of the third operand. When an operand wraps around from location $2^{24} - 1$ (or $2^{31} - 1$) to location 0, operand bytes in locations up to and including $2^{24} - 1$ (or $2^{31} - 1$) are considered to be to the left of bytes in locations from 0 up.

In the 24-bit addressing mode, wraparound is from location $2^{24} - 1$ to location 0; and, in the 31-bit addressing mode, wraparound is from location $2^{31} - 1$ to location 0.

When the length specified in general register $R_1 + 1$ is zero, no movement takes place, and condition code 0 or 1 is set to indicate the relative values of the lengths.

Padding byte values of B0 hex and B8 hex may be used during the nonpadding part of the operation by some models, in certain cases, as an indi-

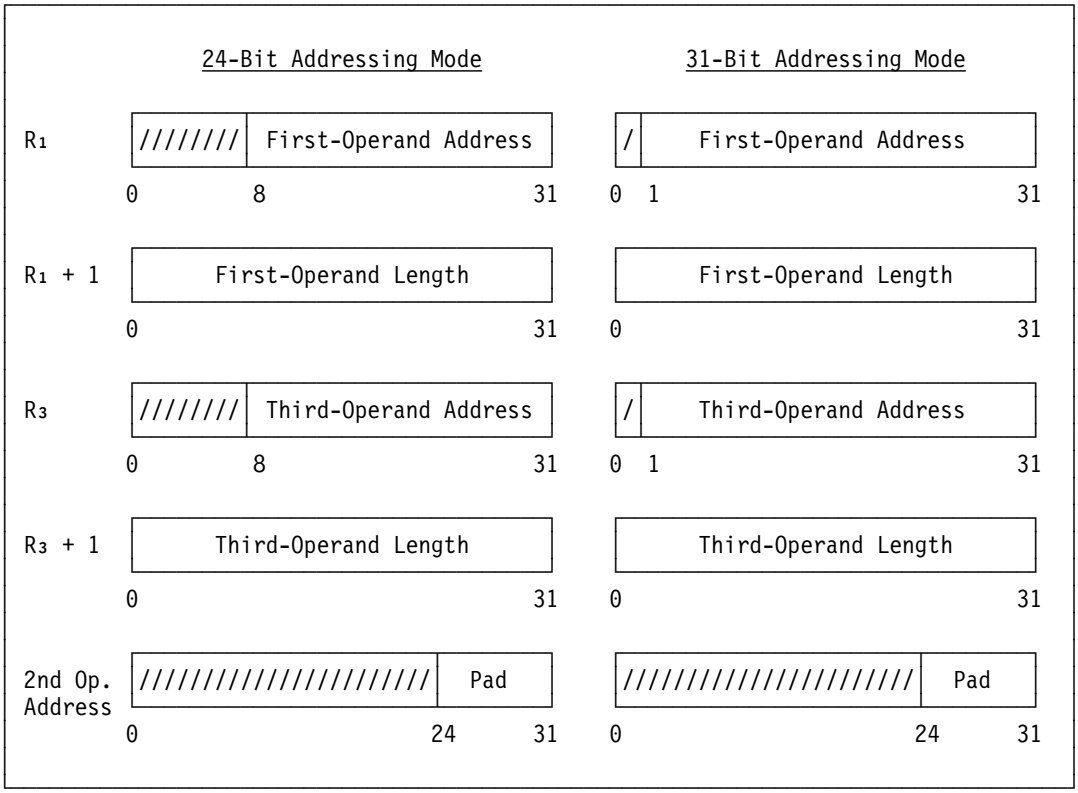



*Figure 7-64. Register Contents and Second-Operand Address for MOVE LONG EXTENDED*



LzLabs GmbH. Ex. 1008-392

cation of whether the movement should be performed bypassing the cache or using the cache, respectively. Thus, a padding byte of B0 hex indicates no intention to reference the destination area after the move, and a padding byte of B8 hex indicates an intention to reference the destination area.

For the nonpadding part of the operation, accesses to the operands for MOVE LONG are single-access references. These accesses do not necessarily appear to occur in a left-to-right direction as observed by other CPUs and by channel programs, unless the padding byte is B1 hex. During the nonpadding part of the operation, operands appear to be accessed doubleword concurrent as observed by other CPUs, provided that both operands start on doubleword boundaries, are an integral number of doublewords in length, and do not overlap.

As observed by other CPUs and by channel programs, that portion of the first operand which is filled with the padding byte is not necessarily stored into in a left-to-right direction and may appear to be stored into more than once.

At the completion of the operation, the length in general register $R_1 + 1$ is decremented by the number of bytes stored at the first-operand location, and the address in general register $R_1$ is incremented by the same amount. The length in general register $R_3 + 1$ is decremented by the number of bytes moved out of the third-operand location, and the address in general register $R_3$ is incremented by the same amount.

If the operation is completed because a CPU-determined number of bytes have been moved without reaching the end of the first operand, the lengths in general registers $R_1 + 1$ and $R_3 + 1$ are decremented by the number of bytes moved, and the addresses in general registers $R_1$ and $R_3$ are incremented by the same number, so that the instruction, when reexecuted, resumes at the next byte to be moved. If the operation is completed during padding, the length field in general register $R_3 + 1$ is zero, the address in general register $R_3$ is incremented by the original contents of general register $R_3 + 1$, and general registers $R_1$ and $R_1 + 1$ reflect the extent of the padding operation.

The padding byte may be formed from $D_2(B_2)$ multiple times during the execution of the instruction, and the registers designated by $R_1$ and $R_3$ may be updated multiple times. Therefore, if $B_2$ equals $R_1$, $R_1 + 1$, $R_3$, or $R_3 + 1$ and is subject to change during the execution of the instruction, the results are unpredictable.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed. The maximum amount is approximately 4K bytes of either operand.

At the completion of the operation, the leftmost bits which are not part of the address in general registers $R_1$ and $R_3$ may be set to zeros or may remain unchanged from their original values, even when one or both of the original length values are zeros.

When the length of an operand is zero, no access exceptions for that operand are recognized. Similarly, when the third operand is longer than the first operand, access exceptions are not recognized for the part of the third-operand field that is in excess of the first-operand field. For operands longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the current location being processed. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Also, when the $R_1$ field is odd, PER storage-alteration events are not recognized, and no change bits are set.

***Resulting Condition Code:***

0 All bytes moved, operand lengths equal
1 All bytes moved, first-operand length low
2 All bytes moved, first-operand length high
3 CPU-determined number of bytes moved without reaching end of first operand

***Program Exceptions:***

- Access (fetch, operand 3; store, operand 1)
- Operation (if the compare-and-move-extended facility is not installed)
- Specification

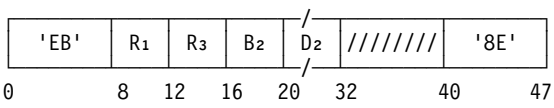

LzLabs GmbH. Ex. 1008-393

**Programming Notes:**

1. MOVE LONG EXTENDED is intended for use in place of MOVE LONG when the operand lengths are specified as 32-bit binary integers and a test for destructive overlap is not required. MOVE LONG EXTENDED sets condition code 3 in cases in which MOVE LONG would be interrupted.
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the movement. The program need not determine the number of bytes that were moved.
3. The function of not processing more than approximately 4K bytes of either operand is intended to permit software polling of a flag that may be set by a program on another CPU during long operations.
4. MOVE LONG EXTENDED may be used for clearing storage by setting the padding byte to zero and the third-operand length to zero. However, the stores associated with this clearing may be multiple-access stores and should not be used to clear an area if the possibility exists that another CPU or a channel program will attempt to access and use the area as soon as it appears to be zero. For more details, see "Storage-Operand Consistency" on page 5-87.
5. When the contents of the $R_1$ and $R_3$ fields are the same, the contents of the designated registers are incremented or decremented only by the number of bytes moved, not by twice the number of bytes moved. The condition code is finally set to 0 after possible settings to 3.
6. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

## MOVE LONG UNICODE

```
MVCLU    R1,R3,D2(B2)      [RSE]
```




All or part of the third operand is placed at the first-operand location. The remaining rightmost two-byte character positions, if any, of the first-operand location are filled with two-byte padding characters. The operation proceeds until the end of the first-operand location is reached or a CPU-determined number of characters have been placed at the first-operand location, whichever occurs first. The result is indicated in the condition code.

The $R_1$ and $R_3$ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost character of the first operand and third operand is designated by the contents of general registers $R_1$ and $R_3$, respectively. The number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 0-31 of general registers $R_1 + 1$ and $R_3 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers.

The contents of general registers $R_1 + 1$ and $R_3 + 1$ must specify an even number of bytes; otherwise, a specification exception is recognized.

The handling of the addresses in general registers $R_1$ and $R_3$ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_3$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of the registers constitute the address, and the contents of bit position 0 are ignored.

The second-operand address is not used to address data; instead, the rightmost 16 bits of the second-operand address, bits 16-31, are the two-byte padding character. Bits 0-15 of the second-operand address are ignored.

The contents of the registers and address just described are shown in Figure 7-65 on page 7-91.

The result is obtained as if the movement starts at the left end of both fields and proceeds to the right, character by character. The operation is ended when the number of characters specified by the contents of general register $R_1 + 1$ have been placed at the first-operand location or when a CPU-determined number of characters have been



LzLabs GmbH. Ex. 1008-394

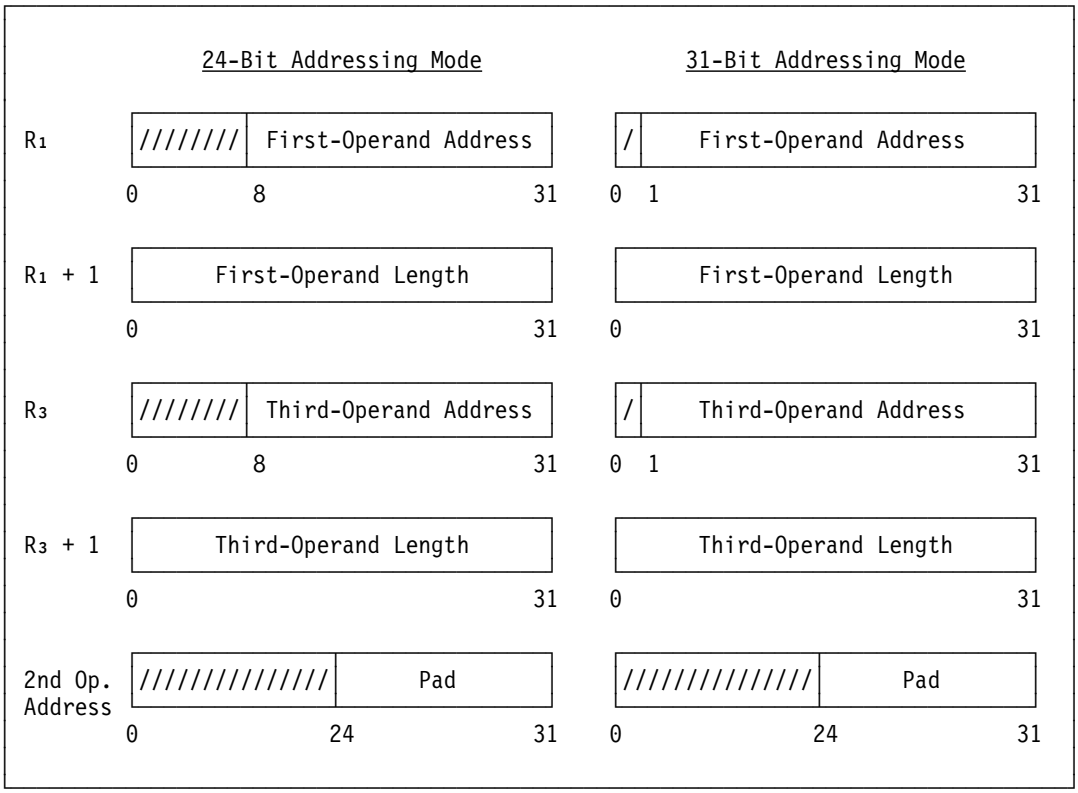



*Figure 7-65. Register Contents and Second-Operand Address for MOVE LONG UNICODE*

placed, whichever occurs first. If the third operand is shorter than the first operand, the remaining rightmost character positions of the first-operand location are filled with the two-byte padding character.

When the operation is completed because the end of the first operand has been reached, the condition code is set to 0 if the two operand lengths are equal, it is set to 1 if the first-operand length is less than the third-operand length, or it is set to 2 if the first-operand length is greater than the third-operand length. When the operation is completed because a CPU-determined number of characters have been moved without reaching the end of the first operand, condition code 3 is set.

No test is made for destructive overlap, and the results in the first-operand location are unpredictable when destructive overlap exists. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it.

Operands do not overlap destructively if the leftmost character of the first operand does not coincide with any of the third-operand characters participating in the operation other than the leftmost character of the third operand. When an operand wraps around from location $2^{24} - 1$ (or $2^{31} - 1$) to location 0, operand characters in locations up to and including $2^{24} - 1$ (or $2^{31} - 1$) are considered to be to the left of characters in locations from 0 up.

In the 24-bit addressing mode, wraparound is from location $2^{24} - 1$ to location 0; and, in the 31-bit addressing mode, wraparound is from location $2^{31} - 1$ to location 0.

When the length specified in general register $R_1 + 1$ is zero, no movement takes place, and condition code 0 or 1 is set to indicate the relative values of the lengths.

For the nonpadding part of the operation, accesses to the operands for MOVE LONG UNICODE are single-access references. These accesses do not necessarily appear to occur in a left-to-right direction as observed by other CPUs and by channel programs. During the nonpadding part of the operation, operands appear to be


LzLabs GmbH. Ex. 1008-395

accessed doubleword concurrent as observed by other CPUs, provided that both operands start on doubleword boundaries, are an integral number of doublewords in length, and do not overlap.

As observed by other CPUs and by channel programs, that portion of the first operand which is filled with the two-byte padding character is not necessarily stored into in a left-to-right direction and may appear to be stored into more than once.

At the completion of the operation, the length in general register $R_1 + 1$ is decremented by 2 times the number of characters stored at the first-operand location, and the address in general register $R_1$ is incremented by the same amount. The length in general register $R_3 + 1$ is decremented by 2 times the number of characters moved out of the third-operand location, and the address in general register $R_3$ is incremented by the same amount.

If the operation is completed because a CPU-determined number of characters have been moved without reaching the end of the first operand, the lengths in general registers $R_1 + 1$ and $R_3 + 1$ are decremented by 2 times the number of characters moved, and the addresses in general registers $R_1$ and $R_3$ are incremented by the same number, so that the instruction, when reexecuted, resumes at the next character to be moved. If the operation is completed during padding, the length field in general register $R_3 + 1$ is zero, the address in general register $R_3$ is incremented by 2 times the number of characters moved from operand 3, and general registers $R_1$ and $R_1 + 1$ reflect the extent of the padding operation.

The two-byte padding character may be formed from $D_2(B_2)$ multiple times during the execution of the instruction, and the registers designated by $R_1$ and $R_3$ may be updated multiple times. Therefore, if $B_2$ equals $R_1$, $R_1 + 1$, $R_3$, or $R_3 + 1$ and is subject to change during the execution of the instruction, the results are unpredictable.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

At the completion of the operation, the leftmost bits which are not part of the address in general registers $R_1$ and $R_3$ may be set to zeros or may remain unchanged from their original values, including the case when one or both of the original length values are zeros.

When the length of an operand is zero, no access exceptions for that operand are recognized. Similarly, when the third operand is longer than the first operand, access exceptions are not recognized for the part of the third-operand field that is in excess of the first-operand field. For operands longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the current location being processed. Access exceptions are not recognized for an operand if the R field or length associated with that operand is odd. Also, when the $R_1$ field or length is odd, PER storage-alteration events are not recognized, and no change bits are set.

***Resulting Condition Code:***

0 All characters moved, operand lengths equal
1 All characters moved, first-operand length low
2 All characters moved, first-operand length high
3 CPU-determined number of characters moved without reaching end of first operand

***Program Exceptions:***

- Access (fetch, operand 3; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

**Programming Notes:**

1. MOVE LONG UNICODE is intended for use in place of MOVE LONG or MOVE LONG EXTENDED when the padding character is two bytes. The character may be a Unicode character or any other double-byte character. MOVE LONG UNICODE sets condition code 3 in cases in which MOVE LONG would be interrupted.
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the movement. The program need not determine the number of characters that were moved.
3. MOVE LONG UNICODE may be used for filling storage with padding characters by



LzLabs GmbH. Ex. 1008-396

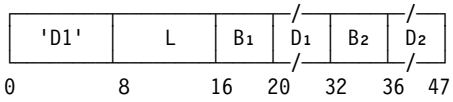
placing the padding character in the second-operand address and setting the third-operand length to zero. However, the stores associated with this clearing may be multiple-access stores and should not be used to clear an area if the possibility exists that another CPU or a channel program will attempt to access and use the area as soon as it appears to be zero. For more details, see "Storage-Operand Consistency" on page 5-87.

4. When the contents of the $R_1$ and $R_3$ fields are the same, the contents of the designated registers are incremented or decremented only by 2 times the number of characters moved, not by 4 times the number of characters moved. The condition code is finally set to 0 after possible settings to 3.
5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
6. If padding with a Unicode space character is required (or any character whose representation is less than or equal to FFF hex), the character may be represented in the displacement field of the instruction, for example:

```
MVCLU   6,8,X'020'
```

## MOVE NUMERICS

```
MVN     D1(L,B1),D2(B2)          [SS]
```




The rightmost four bits of each byte in the second operand are placed in the rightmost bit positions of the corresponding bytes in the first operand. The leftmost four bits of each byte in the first operand remain unchanged.

Each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

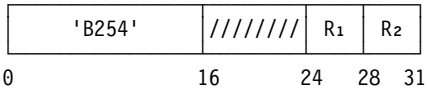
- Access (fetch, operand 2; fetch and store, operand 1)

**Programming Notes:**

1. An example of the use of the MOVE NUMERICS instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. MOVE NUMERICS moves the numeric portion of a decimal-data field that is in the zoned format. The zoned-decimal format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.
3. Accesses to the first operand of MOVE NUMERICS consist in fetching the rightmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in "Multiprogramming and Multiprocessing Examples" in Appendix A, "Number Representation and Instruction-Use Examples."

## MOVE PAGE (Facility 1)

```
MVPG    R1,R2    [RRE]
```




This definition applies if move-page facility 1 is installed. The MOVE PAGE instruction of move-page facility 2 is defined in Chapter 10, "Control Instructions."

The first operand is replaced by the second operand. The first and second operands both are 4K bytes on 4K-byte boundaries. The results are indicated in the condition code.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively.


LzLabs GmbH. Ex. 1008-397

The handling of the addresses in general registers $R_1$ and $R_2$ depends on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 8-19 of a general register, with 12 rightmost zeros appended, are the address, and bits 0-7 and 20-31 in the register are ignored. In the 31-bit addressing mode, the contents of bit positions 1-19 of a general register, with 12 rightmost zeros appended, are the address, and bits 0 and 20-31 in the register are ignored.

Bits 16-23 of general register 0 must be 00000001 binary; otherwise, a specification exception is recognized. Bits 0-15 and 24-31 of general register 0 are ignored.

The contents of the registers just described are shown in Figure 7-66.

When DAT is on and the page-invalid bit is one in the page-table entry for an operand, additional address translation is performed to determine whether the operand is valid in expanded storage. As a result, the replacement of the first operand by the second operand may be performed by moving data from main storage to main storage, from main storage to expanded storage, or from expanded storage to main storage, depending on whether and where the operands are valid. When 4K bytes have been moved, condition code 0 is set.

Certain conditions prevent data movement from occurring and cause a nonzero condition code to be set. Data movement is prevented, and condition code 1 is set, if (1) the second operand is valid in either main storage or expanded storage, but the first operand is invalid in both main storage and expanded storage; (2) both operands are valid in expanded storage; or (3) data movement between main storage and expanded storage is due to occur but the translation path for the expanded-storage operand is locked or the expanded-storage block containing that operand either is not available or causes an expanded-storage data error. When condition code 1 is set because of an expanded-storage data error, the contents of the first-operand location are unpredictable. Data movement is prevented, and condition code 2 is set, if the second operand is invalid in both main storage and expanded storage.

When one operand is invalid in both main storage and expanded storage and an access exception can be recognized for the other operand, it is unpredictable whether a nonzero condition code is set or the access exception is recognized.

The case when the page-table entry for an operand is outside the page table is treated as a page-translation-exception condition.

When data is moved to or from expanded storage, access-list-controlled, page, and key-controlled protection apply, and it is unpredictable whether

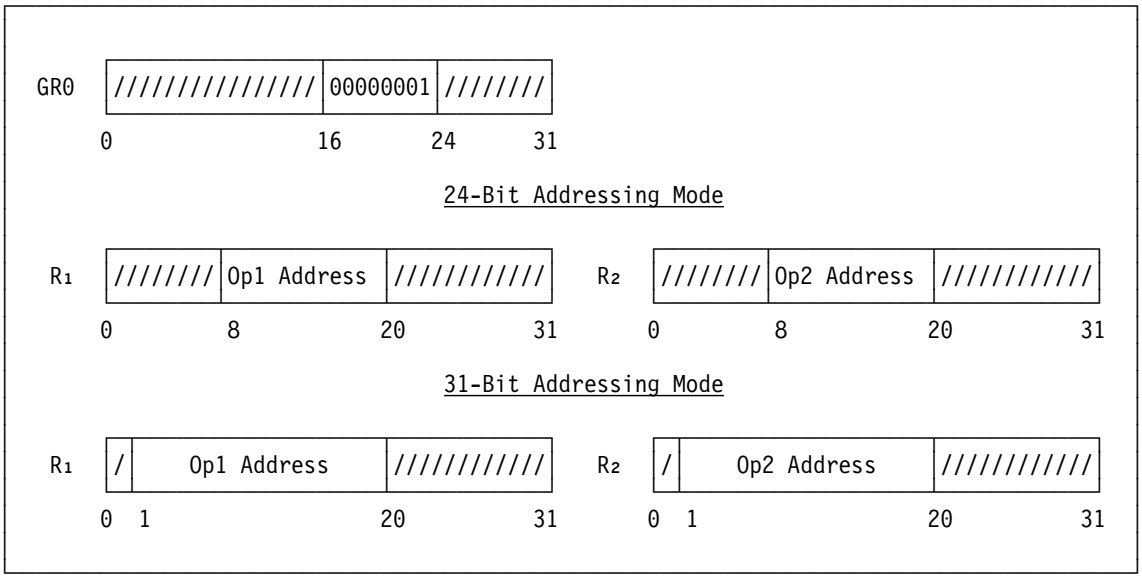



*Figure 7-66. Register Contents for MOVE PAGE of Move-Page Facility 1*



LzLabs GmbH. Ex. 1008-398

low-address protection applies. The protection mechanisms apply to main storage in the normal way.

When the first operand is valid in main storage and the second operand is valid in expanded storage, but the expanded-storage block containing the second operand is unavailable, a storage-alteration PER event may be recognized, and the change bit may be set, for the first operand even though the first-operand location remains unchanged.

**Operation in a Multiple-CPU Configuration**

The references to main storage and to expanded storage are not necessarily single-access references and are not necessarily performed in a left-to-right direction, as observed by other CPUs and by channel programs.

If two or more CPUs move data to or from expanded storage at approximately the same instant in time, depending on the model, the operations may be performed one at a time, or the operations may be performed concurrently. Concurrent operation may occur even if the instructions address the same expanded-storage block.

When two or more CPUs move data to the same expanded-storage block concurrently, the resulting values in the expanded-storage block for each group of bytes transferred may be from any of the instructions being executed simultaneously. The number of bytes transferred as a group is unpredictable.

Similarly, for concurrent movement to and from the same expanded-storage block, the resulting values for each group of bytes moved from expanded storage may be either the old or the new values from the expanded-storage block.

When data movement is due to occur between main storage and expanded storage, the translation path being used for the expanded-storage operand is set to the locked state. When this data movement is completed successfully, or when condition code 1 is due to be set because the movement cannot be completed successfully, the translation path is set to the unlocked state.

***Resulting Condition Code:***

0 Data moved
1 First operand invalid and second operand valid, both operands valid in expanded storage, translation path locked, expanded-storage block unavailable, or expanded-storage data error
2 Second operand invalid
3 --

***Program Exceptions:***

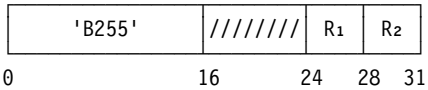
- Access (fetch, operand 2; store, operand 1, except low-address protection for operand in expanded storage is unpredictable)
- Specification

**Programming Notes:**

1. Since an expanded-storage location may be accessed by means of more than one translation path or even without translation, the locked state of a translation path does not necessarily prevent concurrent accesses to the location by different processes. To ensure predictable results when data is in either main storage or expanded storage, the program must use a programmed lock to prevent different processes from performing concurrent store accesses or concurrent fetch and store accesses to the same location.
2. Monitoring for PER storage-alteration events is done using logical addresses. Thus, it applies to the operands of MOVE PAGE regardless of whether the operands are in main storage or expanded storage.

## MOVE STRING

```
MVST     R1,R2     [RRE]
```




All or part of the second operand is placed in the first-operand location. The operation proceeds until the end of the second operand is reached or a CPU-determined number of bytes have been moved, whichever occurs first. The CPU-determined number is at least one. The result is indicated in the condition code.

The location of the leftmost byte of the first operand and second operand is designated by the


LzLabs GmbH. Ex. 1008-399

contents of general registers $R_1$ and $R_2$, respectively.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit position 0 are ignored.

The end of the second operand is indicated by an ending character in the last byte position of the operand. The ending character to be used to determine the end of the second operand is specified in bit positions 24-31 of general register 0. Bit positions 0-23 of general register 0 are reserved for possible future extensions and must contain all zeros; otherwise, a specification exception is recognized.

The operation proceeds left to right and ends as soon as the second-operand ending character has been moved or a CPU-determined number of second-operand bytes have been moved, whichever occurs first. The CPU-determined number is at least one. When the ending character is in the first byte position of the second operand, only the ending character is moved. When the ending character has been moved, condition code 1 is set. When a CPU-determined number of second-operand bytes not including an ending character have been moved, condition code 3 is set. Destructive overlap is not recognized. If the second operand is used as a source after it has been used as a destination, the results are unpredictable to the extent that an ending character in the second operand may not be recognized.

When condition code 1 is set, the address of the ending character in the first operand is placed in general register $R_1$, and the contents of general register $R_2$ remain unchanged. When condition code 3 is set, the address of the next byte to be processed in the first and second operands is placed in general registers $R_1$ and $R_2$, respectively. Whenever an address is placed in a general register, bits 0-7 of the register, in the 24-bit mode, or bit 0, in the 31-bit mode, are set to zeros.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

Access exceptions for the first and second operands are recognized only for that portion of the operand that is necessarily used in the operation.

The storage-operand-consistency rules are the same as for the MOVE (MVC) instruction, except that destructive overlap is not recognized.

***Resulting Condition Code:***

0 --
1 Entire second operand moved; general register $R_1$ updated with address of ending character in first operand; general register $R_2$ unchanged
2 --
3 CPU-determined number of bytes moved; general registers $R_1$ and $R_2$ updated with addresses of next bytes

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Operation (if the string-instruction facility is not installed)
- Specification

**Programming Notes:**

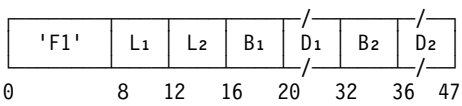
1. An example of the use of the MOVE STRING instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the data movement. The program need not determine the number of bytes that were moved.
3. $R_1$ or $R_2$ may be zero, in which case general register 0 is treated as containing an address and also the ending character.
4. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.



LzLabs GmbH. Ex. 1008-400

## MOVE WITH OFFSET

```
MVO    D1(L1,B1),D2(L2,B2)      [SS]
```




The second operand is placed to the left of and adjacent to the rightmost four bits of the first operand.

The rightmost four bits of the first operand are attached as the rightmost bits to the second operand, the second-operand bits are offset by four bit positions, and the result is placed at the first-operand location.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand is too short to contain all of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if the operands were processed one byte at a time, as if each result byte were stored immediately after fetching the necessary operand bytes, and as if the left digit of each second-operand byte were to remain available for the next result byte and need not be refetched.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

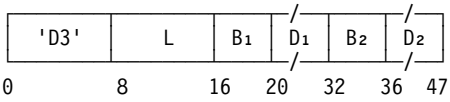
- Access (fetch, operand 2; fetch and store, operand 1)

**Programming Notes:**

1. An example of the use of the MOVE WITH OFFSET instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. MOVE WITH OFFSET may be used to shift packed decimal data by an odd number of digit positions. The packed-decimal format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes. In many cases, however, SHIFT AND ROUND DECIMAL may be more convenient to use.
3. Access to the rightmost byte of the first operand of MOVE WITH OFFSET consists in fetching the rightmost four bits and subsequently storing the updated value of this byte. These fetch and store accesses to the rightmost byte of the first operand do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in "Multiprogramming and Multiprocessing Examples" in Appendix A, "Number Representation and Instruction-Use Examples."
4. The storage-operand references for MOVE WITH OFFSET may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

## MOVE ZONES

```
MVZ    D1(L,B1),D2(B2)          [SS]
```




The leftmost four bits of each byte in the second operand are placed in the leftmost four bit positions of the corresponding bytes in the first operand. The rightmost four bits of each byte in the first operand remain unchanged.

Each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2; fetch and store, operand 1)

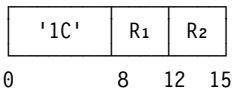

LzLabs GmbH. Ex. 1008-401

**Programming Notes:**

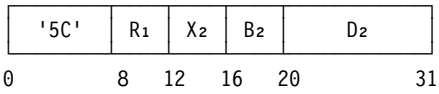
1. An example of the use of the MOVE ZONES instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. MOVE ZONES moves the zoned portion of a decimal field in the zoned format. The zoned format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.
3. Accesses to the first operand of MOVE ZONES consist in fetching the leftmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for the OR (OI) instruction in "Multiprogramming and Multiprocessing Examples" in Appendix A, "Number Representation and Instruction-Use Examples."

## MULTIPLY

```
MR      R1,R2      [RR]
```




```
M       R1,D2(X2,B2)      [RX]
```




The 32-bit first operand (the multiplicand), which is the second word at the first-operand location, is multiplied by the 32-bit second-operand (the multiplier), and the 64-bit product is placed at the first-operand location.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

Both the multiplicand and multiplier are treated as 32-bit signed binary integers. The multiplicand is in general register $R_1 + 1$. The contents of general register $R_1$ are ignored. The product is a 64-bit signed binary integer, which replaces the contents of the even-odd pair of general registers designated by $R_1$. An overflow cannot occur.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

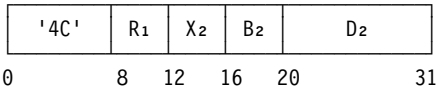
- Access (fetch, operand 2 of M only)
- Specification

**Programming Notes:**

1. An example of the use of the MULTIPLY instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The significant part of the product usually occupies 62 bit positions or fewer. Only when two maximum negative numbers are multiplied are 63 significant product bits formed.

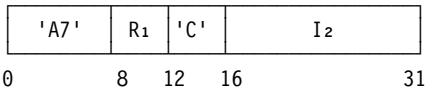
## MULTIPLY HALFWORD

```
MH      R1,D2(X2,B2)      [RX]
```




## MULTIPLY HALFWORD IMMEDIATE

```
MHI     R1,I2     [RI]
```




The 32-bit first operand (multiplicand) is multiplied by the 16-bit second operand (multiplier), and the rightmost 32 bits of the product are placed at the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer.



LzLabs GmbH. Ex. 1008-402

The multiplicand is treated as a 32-bit signed binary integer and is replaced by the rightmost 32 bits of the signed-binary-integer product. The bits to the left of the 32 rightmost bits of the product are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

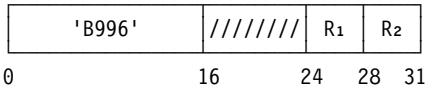
- Access (fetch, operand 2 of MH only)
- Operation (MHI if the immediate-and-relative-instruction facility is not installed)

**Programming Notes:**

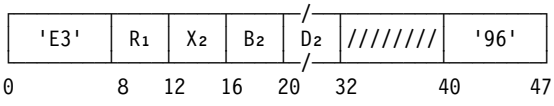
1. An example of the use of the MULTIPLY HALFWORD instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The significant part of the product usually occupies 46 bit positions or fewer. Only when two maximum negative numbers are multiplied are 47 significant product bits formed. Since the rightmost 32 bits of the product are placed unchanged at the first-operand location, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow. For a negative product, the 32 bits placed in register $R_1$ are the rightmost part of the product in two's-complement notation.

## MULTIPLY LOGICAL

MLR R1,R2 [RRE]




ML R1,D2(X2,B2) [RXE]




The 32-bit first operand (the multiplicand) is multiplied by the 32-bit second operand (the multiplier), and the 64-bit product is placed at the first-operand location.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

Both the multiplicand and the multiplier are treated as 32-bit unsigned binary integers. The multiplicand is in general register $R_1 + 1$. The contents of general register $R_1$ are ignored. The product is a 64-bit unsigned binary integer. Bits 0-31 of the product replace the contents of general register $R_1$, and bits 32-63 of the product replace the contents of general register $R_1 + 1$. An overflow cannot occur.

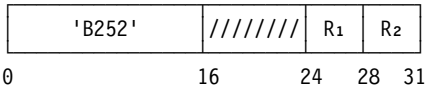
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

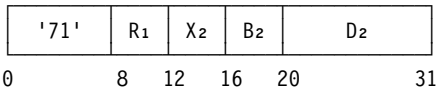
- Access (fetch, operand 2 of ML only)
- Operation (if z/Architecture is not installed)
- Specification

## MULTIPLY SINGLE

MSR R1,R2 [RRE]




MS R1,D2(X2,B2) [RX]




The first operand (multiplicand) is multiplied by the second operand (multiplier), and the rightmost 32 bits of the product are placed at the first-operand location.

Both the multiplicand and multiplier are treated as 32-bit signed binary integers. The multiplicand is taken from general register $R_1$ and is replaced by the rightmost 32 bits of the signed-binary-integer product. The bits to the left of the 32 rightmost bits of the product are not tested for significance; no overflow indication is given.


LzLabs GmbH. Ex. 1008-403

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

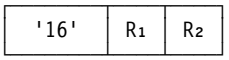
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

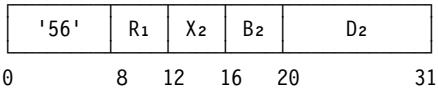
- Access (fetch, operand 2 of MS only)
- Operation (if the immediate-and-relative-instruction facility is not installed)

# OR

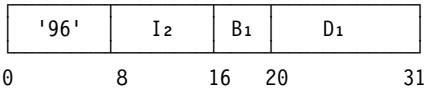
```
OR      R1,R2       [RR]
```



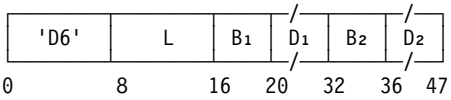

```
O       R1,D2(X2,B2)      [RX]
```




```
OI      D1(B1),I2         [SI]
```




```
OC      D1(L,B1),D2(B2)           [SS]
```




The OR of the first and second operands is placed at the first-operand location.

The connective OR is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero.

For OR (OC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

For OR (OI), the first operand is one byte in length, and only one byte is stored.

***Resulting Condition Code:***

0 Result zero
1 Result not zero
2 --
3 --

***Program Exceptions:***

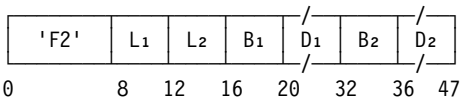
- Access (fetch, operand 2, O and OC; fetch and store, operand 1, OI and OC)

**Programming Notes:**

1. Examples of the use of the OR instruction are given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. OR may be used to set a bit to one.
3. Accesses to the first operand of OR (OI) and OR (OC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown in “Multiprogramming and Multiprocessing Examples” in Appendix A, “Number Representation and Instruction-Use Examples.”

# PACK

```
PACK    D1(L1,B1),D2(L2,B2)       [SS]
```




The format of the second operand is changed from zoned to packed, and the result is placed at the first-operand location. The zoned and packed formats are described in Chapter 8, “Decimal Instructions.”

The second operand is treated as having the zoned format. The numeric bits of each byte are treated as a digit. The zone bits are ignored,



LzLabs GmbH. Ex. 1008-404

except the zone bits in the rightmost byte, which are treated as a sign.

The sign and digits are moved unchanged to the first operand and are not checked for valid codes. The sign is placed in the rightmost four bit positions of the rightmost byte of the result field, and the digits are placed adjacent to the sign and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if each result byte were stored immediately after fetching the necessary operand bytes. Two second-operand bytes are needed for each result byte, except for the rightmost byte of the result field, which requires only the rightmost second-operand byte.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

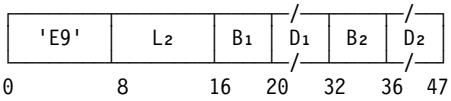
- Access (fetch, operand 2; store, operand 1)

**Programming Notes:**

1. An example of the use of the PACK instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. PACK may be used to interchange the two hexadecimal digits in one byte by specifying a zero in the $L_1$ and $L_2$ fields and the same address for both operands.
3. To remove the zone bits of all bytes of a field, including the rightmost byte, both operands should be extended on the right with a dummy byte, which subsequently should be ignored in the result field.
4. The storage-operand references for PACK may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)

## PACK ASCII

```
PKA      D1(B1),D2(L2,B2)   [SS]
```




The format of the second operand is changed from ASCII to packed, and the result is placed at the first-operand location. The packed format is described in Chapter 8, “Decimal Instructions.”

The second-operand bytes are treated as containing decimal digits, having the binary encoding 0000-1001 for 0-9, in their rightmost four bit positions. The leftmost four bit positions of a byte are ignored. The second operand is considered to be positive.

The implied positive sign (1100 binary) and the source digits are placed at the first-operand location. The source digits are moved unchanged and are not checked for valid codes. The sign is placed in the rightmost four bit positions of the rightmost byte of the result field, and the digits are placed adjacent to the sign and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros.

The length of the first operand is 16 bytes.

The length of the second operand is designated by the contents of the $L_2$ field. The second-operand length must not exceed 32 bytes ($L_2$ must be less than or equal to 31); otherwise, a specification exception is recognized.

When the length of the second operand is 32 bytes, the leftmost byte is ignored.

The results are unpredictable if the first and second operands overlap in any way.

As observed by other CPUs and by channel programs, the first-operand location is not necessarily stored into in any particular order.

***Condition Code:*** The code remains unchanged.


LzLabs GmbH. Ex. 1008-405

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

**Programming Notes:**

1. Although PACK ASCII is primarily intended to change the format of ASCII decimal digits, its use is not restricted to ASCII since the left-most four bits of each byte are ignored.

2. The following example illustrates the use of the instruction to pack ASCII digits:

```
ASDIGITS  DS     0CL31
          DC     X'3132333435'
          DC     X'3637383930'
          DC     X'3132333435'
          DC     X'3637383930'
          DC     X'3132333435'
          DC     X'3637383930'
          DC     X'31'
PKDIGITS  DS     PL16
          ...
          PKA    PKDIGITS,ASDIGITS(31)
```

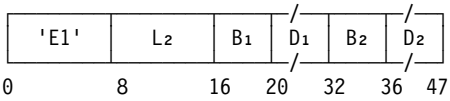
3. The instruction can also be used to pack EBCDIC digits, which is especially useful when the length of the second operand is greater than the 16-byte second-operand limit of PACK.

```
EBDIGITS  DS     0CL31
          DC     X'F1F2F3F4F5'
          DC     X'F6F7F8F9F0'
          DC     X'F1F2F3F4F5'
          DC     X'F6F7F8F9F0'
          DC     X'F1F2F3F4F5'
          DC     X'F6F7F8F9F0'
          DC     X'F1'
PKDIGITS  DS     PL16
          ...
          PKA    PKDIGITS,EBDIGITS(31)
```

4. The storage-operand references for PACK ASCII may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

# PACK UNICODE

```
PKU      D1(B1),D2(L2,B2)   [SS]
```




The format of the second operand is changed from Unicode to packed, and the result is placed at the first-operand location. The packed format is described in Chapter 8, "Decimal Instructions."

The two-byte second-operand characters are treated as Unicode Basic Latin characters containing decimal digits, having the binary encoding 0000-1001 for 0-9, in their rightmost four bit positions. The leftmost 12 bit positions of a character are ignored. The second operand is considered to be positive.

The implied positive sign (1100 binary) and the source digits are placed at the first-operand location. The source digits are moved unchanged and are not checked for valid codes. The sign is placed in the rightmost four bit positions of the rightmost byte of the result field, and the digits are placed adjacent to the sign and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros.

The length of the first operand is 16 bytes.

The byte length of the second operand is designated by the contents of the $L_2$ field. The second-operand length must not exceed 32 characters or 64 bytes, and the byte length must be even ($L_2$ must be less than or equal to 63 and must be odd); otherwise, a specification exception is recognized.

When the length of the second operand is 32 characters (64 bytes), the leftmost character is ignored.

The results are unpredictable if the first and second operands overlap in any way.



LzLabs GmbH. Ex. 1008-406

As observed by other CPUs and by channel programs, the first-operand location is not necessarily stored into in any particular order.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

**Programming Notes:**

1. The following example illustrates the use of PACK UNICODE to pack European numbers:

```
UNDIGITS  DS     0CL62
          DC     X'0031003200330034003 5'
          DC     X'0036003700380039003 0'
          DC     X'0031003200330034003 5'
          DC     X'0036003700380039003 0'
          DC     X'0031003200330034003 5'
          DC     X'0036003700380039003 0'
          DC     X'0031'
PKDIGITS  DS     PL16
          ...
          PKU    PKDIGITS,UNDIGITS(62)
```

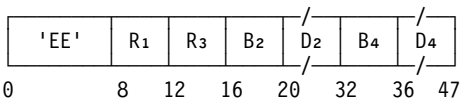
2. Because the leftmost 12 bits of each character are ignored, those Unicode decimal digits where the digit zero has four rightmost zero bits can also be packed by the instruction. For example, for Thai digits:

```
UNDIGITS  DS     0CL62
          DC     X'0E510E520E530E540E55'
          DC     X'0E560E570E580E590E50'
          DC     X'0E510E520E530E540E55'
          DC     X'0E560E570E580E590E50'
          DC     X'0E510E520E530E540E55'
          DC     X'0E560E570E580E590E50'
          DC     X'0E51'
PKDIGITS  DS     PL16
          ...
          PKU    PKDIGITS,UNDIGITS(62)
```

3. The storage-operand references for PACK UNICODE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)

## PERFORM LOCKED OPERATION

```
PLO     R1,D2(B2),R3,D4(B4)      [SS]
```




After the lock specified in general register 1 has been obtained, the operation specified by the function code in general register 0 is performed, and then the lock is released. However, as observed by other CPUs: (1) storage operands, including fields in a parameter list that may be used, may be fetched and tested for store-type access exceptions before the lock is obtained, and (2) operands may be stored in the parameter list after the lock has been released. If an operand not in the parameter list is fetched before the lock is obtained, it is fetched again after the lock has been obtained.

The function code can specify any of six operations: compare and load, compare and swap, double compare and swap, compare and swap and store, compare and swap and double store, or compare and swap and triple store.

A test bit in general register 0 specifies, when one, that a lock is not to be obtained and none of the six operations is to be performed but, instead, the validity of the function code is to be tested. This will be useful if additional function codes for additional operations are assigned in the future. This definition is written as if the test bit is zero except when stated otherwise.

If compare and load is specified, the first-operand comparison value and the second operand are compared. If they are equal, the fourth operand is placed in the third-operand location. If the comparison indicates inequality, the second operand is placed in the first-operand-comparison-value location as a new first-operand comparison value.

If compare and swap is specified, the first-operand comparison value and the second operand are compared. If they are equal, the first-operand replacement value is stored at the second-operand location. If the comparison indicates inequality, the second operand is placed in the first-operand-comparison-value location as a new first-operand comparison value.


LzLabs GmbH. Ex. 1008-407

If double compare and swap is specified, the first-operand comparison value and the second operand are compared. If they are equal, the third-operand comparison value and the fourth operand are compared. If both comparisons indicate equality, the first-operand and third-operand replacement values are stored at the second-operand location and fourth-operand location, respectively. If the first comparison indicates inequality, the second operand is placed in the first-operand-comparison-value location as a new first-operand comparison value. If the first comparison indicates equality but the second does not, the fourth operand is placed in the third-operand-comparison-value location as a new third-operand comparison value.

If compare and swap and store, double store, or triple store is specified, the first-operand comparison value and the second operand are compared. If they are equal, the first-operand replacement value is stored at the second-operand location, and the third operand is stored at the fourth-operand location. Then, if the operation is the double-store or triple-store operation, the fifth operand is stored at the sixth-operand location, and, if it is the triple-store operation, the seventh operand is stored at the eighth-operand location. If the first-operand comparison indicates inequality, the second operand is placed in the first-operand-comparison-value location as a new first-operand comparison value.

After any of the six operations, the result of the comparison or comparisons is indicated in the condition code.

The function code (FC) is in bit positions 24-31 of general register 0. The function code specifies not only the operation to be performed but also the length of the operands. A function code that is a multiple of 4 specifies a 32-bit length, and a function code that is one more than a multiple of 4 specifies a 64-bit length. Figure 7-67 shows the function codes, operation names, and operand lengths, and also symbols that may be used to refer to the operations in discussions. For example, PLO.DCS may be used to mean PERFORM LOCKED OPERATION with function code 8. In the symbols, the letter “G” indicates a 64-bit operand length.

The CPU can perform all of the operations specified by the function codes listed in Figure 7-67.

| Function Code | Operation | Operand Length (Bits) | Function Symbol |
|---|---|---|---|
| 0 | Compare and load | 32 | CL |
| 1 | Compare and load | 64 | CLG |
| 4 | Compare and swap | 32 | CS |
| 5 | Compare and swap | 64 | CSG |
| 8 | Double compare and swap | 32 | DCS |
| 9 | Double compare and swap | 64 | DCSG |
| 12 | Compare and swap and store | 32 | CSST |
| 13 | Compare and swap and store | 64 | CSSTG |
| 16 | Compare and swap and double store | 32 | CSDST |
| 17 | Compare and swap and double store | 64 | CSDSTG |
| 20 | Compare and swap and triple store | 32 | CSTST |
| 21 | Compare and swap and triple store | 64 | CSTSTG |

*Figure 7-67. PERFORM LOCKED OPERATION Function Codes and Operations*

Function codes specifying operations that the CPU can perform are called valid. Function codes that have not been assigned to operations or that specify operations that the CPU cannot perform because the operations are not implemented (installed) are called invalid.

Bit 23 of general register 0 is the test bit (T). When bit 23 is zero, the function code in general register 0 must be valid; otherwise, a specification exception is recognized. When bit 23 is one, the condition code is set to 0 if the function code is valid or to 3 if the function code is invalid, and no other operation is performed.

Bits 0-22 of general register 0 must be all zeros; otherwise, a specification exception is recognized. When bit 23 of the register is one, this is the only exception that can be recognized.

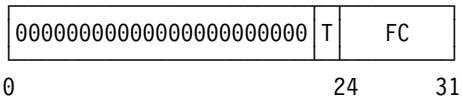


LzLabs GmbH. Ex. 1008-408

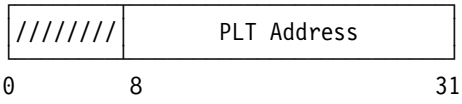
The lock to be used is represented by a program lock token (PLT) whose logical address is specified in general register 1. In the 24-bit addressing mode, the PLT address is bits 8-31 of general register 1, and bits 0-7 of the register are ignored. In the 31-bit addressing mode, the PLT address is bits 1-31 of the register, and bit 0 of the register is ignored.

The contents of general registers 0 and 1 described above are as follows:

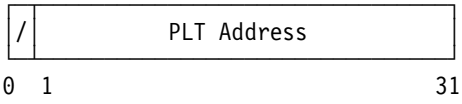
GR 0




GR 1 in 24-Bit Addressing Mode




GR 1 in 31-Bit Addressing Mode




For function codes 0, 4, 8, 12, 16, and 20, the first-operand comparison value is in general register $R_1$. For function codes 4, 8, 12, 16, and 20, the first-operand replacement value is in general register $R_1 + 1$, and $R_1$ designates an even-odd pair of registers and must designate an even-numbered register; otherwise, a specification exception is recognized. For function code 0, $R_1$ can be even or odd.

For function codes 0 and 12, the third operand is in general register $R_3$, and $R_3$ can be even or odd.

For function code 8, the third-operand comparison value is in general register $R_3$, the third-operand replacement value is in general register $R_3 + 1$, and $R_3$ designates an even-odd pair of registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

For all function codes, the $B_2$ and $D_2$ fields of the instruction specify the second-operand address.

For function codes 0, 8, and 12, the $B_4$ and $D_4$ fields of the instruction specify the fourth-operand address.

For function codes 1, 5, 9, 13, 16, 17, 20, and 21, the $B_4$ and $D_4$ fields of the instruction specify the address of a parameter list that is used by the instruction, and this address is not called the fourth-operand address. The parameter list contains odd-numbered operands, including comparison and replacement values, and addresses of even-numbered operands other than the second operand. In the access-register mode, the parameter list also contains access-list-entry tokens (ALETs) associated with the even-numbered-operand addresses.

In the access-register mode, for function codes that cause use of a parameter list containing an ALET, $R_3$ must not be zero; otherwise, a specification exception is recognized.

The rules about $R_1$ and $R_3$, and the use of the address specified by $B_4$ and $D_4$, are summarized in Figure 7-68.

| Function Code | Operation | $R_1$ | $R_3$ | $D_4(B_4)$ |
|---|---|---|---|---|
| 0 | Compare and load | EO | EO | Op4a |
| 1 | Compare and load | - | NZ | PLa |
| 4 | Compare and swap | E | - | - |
| 5 | Compare and swap | - | - | PLa |
| 8 | Double compare and swap | E | E | Op4a |
| 9 | Double compare and swap | - | NZ | PLa |
| 12 | Compare and swap and store | E | EO | Op4a |
| 13 | Compare and swap and store | - | NZ | PLa |
| 16 | Compare and swap and double store | E | NZ | PLa |
| 17 | Compare and swap and double store | - | NZ | PLa |
| 20 | Compare and swap and triple store | E | NZ | PLa |
| 21 | Compare and swap and triple store | - | NZ | PLa |

**Explanation:**

| | |
|---|---|
| - | Ignored. |
| E | Must be even. |
| EO | Can be even or odd. |
| NZ | Must be nonzero in the access-register mode. Ignored otherwise. |
| Op4a | $D_4(B_4)$ is operand-4 address. |
| PLa | $D_4(B_4)$ is parameter-list address. |

*Figure 7-68. Register Rules and $D_4(B_4)$ Usage for PERFORM LOCKED OPERATION*



LzLabs GmbH. Ex. 1008-409

| Function Code | Operation | Op1c | Op1r | Op2a | Op3 or Op3c | Op3 or Op3r | Op4a | Op5 and Op6a | Op7 and Op8a | PLa |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Compare and load | $R_1$ | - | $D_2(B_2)$ | $R_3$ | | $D_4(B_4)$ | - | - | - |
| 1 | Compare and load | PL | - | $D_2(B_2)$ | PL | | PL | - | - | $D_4(B_4)$ |
| 4 | Compare and swap | $R_1$ | $R_1+1$ | $D_2(B_2)$ | - | | - | - | - | - |
| 5 | Compare and swap | PL | PL | $D_2(B_2)$ | - | | - | - | - | $D_4(B_4)$ |
| 8 | Double compare and swap | $R_1$ | $R_1+1$ | $D_2(B_2)$ | $R_3$ | $R_3+1$ | $D_4(B_4)$ | - | - | - |
| 9 | Double compare and swap | PL | PL | $D_2(B_2)$ | PL | PL | PL | - | - | $D_4(B_4)$ |
| 12 | Compare and swap and store | $R_1$ | $R_1+1$ | $D_2(B_2)$ | $R_3$ | | $D_4(B_4)$ | - | - | - |
| 13 | Compare and swap and store | PL | PL | $D_2(B_2)$ | PL | | PL | - | - | $D_4(B_4)$ |
| 16 | Compare and swap and double store | $R_1$ | $R_1+1$ | $D_2(B_2)$ | PL | | PL | PL | - | $D_4(B_4)$ |
| 17 | Compare and swap and double store | PL | PL | $D_2(B_2)$ | PL | | PL | PL | - | $D_4(B_4)$ |
| 20 | Compare and swap and triple store | $R_1$ | $R_1+1$ | $D_2(B_2)$ | PL | | PL | PL | PL | $D_4(B_4)$ |
| 21 | Compare and swap and triple store | PL | PL | $D_2(B_2)$ | PL | | PL | PL | PL | $D_4(B_4)$ |

**Explanation:**

```
-     Operand, value, or address is not used in the operation.
OpNc  Operand-N comparison value.
OpNr  Operand-N replacement value.
OpNa  Operand-N address.
PL    Operand, value, or address is in the parameter list.
PLa   Parameter-list address.
```

*Figure 7-69. Operand and Address Locations for PERFORM LOCKED OPERATION*

Figure 7-69 on page 7-106 shows the locations of the operands (including operand comparison and replacement values), operand addresses, and parameter-list address used by the instruction.

Operand addresses in a parameter list, if used, are in words in the list. In the 24-bit addressing mode, an operand address is bits 8-31 of a word, and bits 0-7 of the word are ignored. In the 31-bit addressing mode, an operand address is bits 1-31 of a word, and bit 0 of the word is ignored.

In the access-register mode, access register 1 specifies the address space containing the program lock token (PLT), access register $B_2$ specifies the address space containing the second operand, and access register $B_4$ specifies the address space containing a fourth operand or a parameter list as shown in Figure 7-69. Also, for an operand whose address is in the parameter list, an access-list-entry token (ALET) is in the list along with the address and is used in the access-register mode to specify the address space containing the operand.

In the access-register mode, if an access exception or PER storage-alteration event is recognized for an operand whose address is in the parameter list, the associated ALET in the parameter list is loaded into access register $R_3$ when the exception or event is recognized. Then, during the resulting program interruption, if a value is due to be stored



LzLabs GmbH. Ex. 1008-410

as the exception access identification at real location 160 or the PER access identification at real location 161, $R_3$ is stored. If the instruction execution is completed without the recognition of an exception or event, the contents of access register $R_3$ are unpredictable. When not in the access-register mode, or when a parameter list containing an ALET is not used, the contents of access register $R_3$ remain unchanged.

The even-numbered (2, 4, 6, and 8) storage operands must be designated on an integral boundary, which is a word boundary for function codes that are a multiple of 4 or a doubleword boundary for function codes that are one more than a multiple of 4. A parameter list, if used, must be designated on a doubleword boundary. Otherwise, a specification exception is recognized. The program-lock-token (PLT) address in general register 1 does not have a boundary-alignment requirement.

All unused fields in a parameter list, including bits 0-7 of a word containing an address in the 24-bit addressing mode or bit 0 of a word containing an address in the 31-bit addressing mode, should contain all zeros; otherwise, the program may not operate compatibly in the future.

A serialization operation is performed immediately after the lock is obtained and again immediately before it is released. However, values fetched from the parameter list before the lock is obtained are not necessarily refetched. A serialization operation is not performed if the test bit, bit 23 of general register 0, is one.

In the following figures showing the parameter lists for the different function codes, the offsets shown on the left are byte values.

**Function Codes 0 and 1 (Compare and Load)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-69 on page 7-106.

The parameter list used for function code 1 has the following format:

Parameter List for Function Code 1

| | | |
|---|---|---|
| 0 | | |
| 8 | Operand-1 Comparison Value | |
| 16 | | |
| 24 | | |
| 32 | | |
| 40 | Operand 3 | |
| 48 | | |
| 56 | | |
| 64 | | Operand-4 ALET |
| 72 | | Operand-4 Adr. |

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the third operand is replaced by the fourth operand, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

**Function Codes 4 and 5 (Compare and Swap)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-69 on page 7-106.

The parameter list used for function code 5 has the following format:

Parameter List for Function Code 5

| | |
|---|---|
| 0 | |
| 8 | Operand-1 Comparison Value |
| 16 | |
| 24 | Operand-1 Replacement Value |

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the first-operand replacement value is stored at


LzLabs GmbH. Ex. 1008-411

the second-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

**Function Codes 8 and 9 (Double Compare and Swap)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-69 on page 7-106.

The parameter list used for function code 9 has the following format:

Parameter List for Function Code 9

| | | |
|---|---|---|
| 0 | | |
| 8 | Operand-1 Comparison Value | |
| 16 | | |
| 24 | Operand-1 Replacement Value | |
| 32 | | |
| 40 | Operand-3 Comparison Value | |
| 48 | | |
| 56 | Operand-3 Replacement Value | |
| 64 | | Operand-4 ALET |
| 72 | | Operand-4 Adr. |

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the third-operand comparison value is compared to the fourth operand. When the third-operand comparison value is equal to the fourth operand (after the first-operand comparison value has been found equal to the second operand), the first-operand replacement value is stored at the second-operand location, the third-operand replacement value is stored at the fourth-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

When the third-operand comparison value is not equal to the fourth operand (after the first-operand comparison value has been found equal to the second operand), the third-operand comparison value is replaced by the fourth operand, and condition code 2 is set.



LzLabs GmbH. Ex. 1008-412

**Function Codes 12 and 13 (Compare and Swap and Store)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-69 on page 7-106.

The parameter list used for function code 13 has the following format:

Parameter List for Function Code 13

| | | |
|---|---|---|
| 0 | | |
| 8 | Operand-1 Comparison Value | |
| 16 | | |
| 24 | Operand-1 Replacement Value | |
| 32 | | |
| 40 | | |
| 48 | | |
| 56 | Operand 3 | |
| 64 | | Operand-4 ALET |
| 72 | | Operand-4 Adr. |

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the first-operand replacement value is stored at the second-operand location, the third operand is stored at the fourth-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

**Function Codes 16 and 17 (Compare and Swap and Double Store)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-69 on page 7-106.

The parameter list used for function code 16 has the following format:

Parameter List for Function Code 16

| | | |
|---|---|---|
| 0 | | |
| 8 | | |
| 16 | | |
| 24 | | |
| 32 | | |
| 40 | | |
| 48 | | |
| 56 | | Operand 3 |
| 64 | | Operand-4 ALET |
| 72 | | Operand-4 Adr. |
| 80 | | |
| 88 | | Operand 5 |
| 96 | | Operand-6 ALET |
| 104 | | Operand-6 Adr. |



LzLabs GmbH. Ex. 1008-413

The parameter list used for function code 17 has the following format:

Parameter List for Function Code 17

| | | |
|---|---|---|
| 0 | | |
| 8 | Operand-1 Comparison Value | |
| 16 | | |
| 24 | Operand-1 Replacement Value | |
| 32 | | |
| 40 | | |
| 48 | | |
| 56 | Operand 3 | |
| 64 | | Operand-4 ALET |
| 72 | | Operand-4 Adr. |
| 80 | | |
| 88 | Operand 5 | |
| 96 | | Operand-6 ALET |
| 104 | | Operand-6 Adr. |

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the first-operand replacement value is stored at the second-operand location, the third operand is stored at the fourth-operand location, the fifth operand is stored at the sixth-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

**Function Codes 20 and 21 (Compare and Swap and Triple Store)**

The locations of the operands and addresses used by the instruction are as shown in Figure 7-69 on page 7-106.

The parameter list used for function code 20 has the following format:

Parameter List for Function Code 20

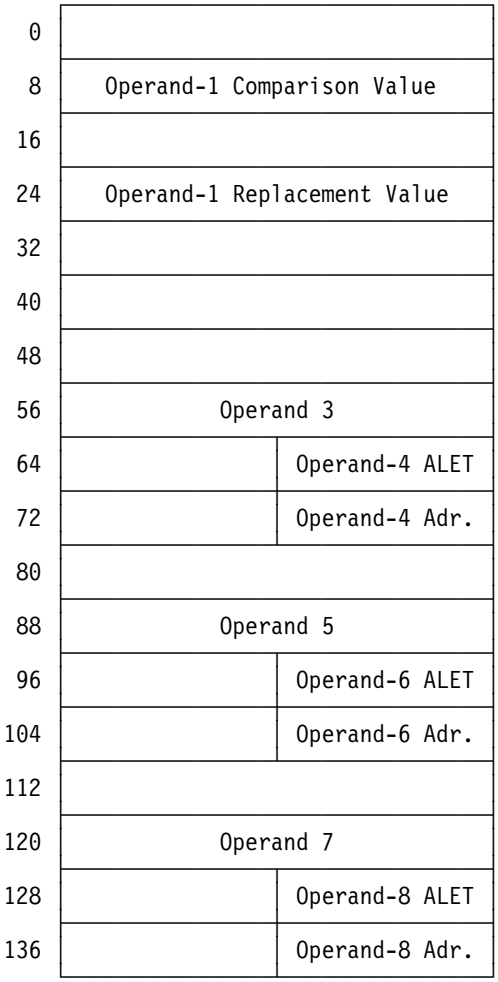
| | | |
|---|---|---|
| 0 | | |
| 8 | | |
| 16 | | |
| 24 | | |
| 32 | | |
| 40 | | |
| 48 | | |
| 56 | | Operand 3 |
| 64 | | Operand-4 ALET |
| 72 | | Operand-4 Adr. |
| 80 | | |
| 88 | | Operand 5 |
| 96 | | Operand-6 ALET |
| 104 | | Operand-6 Adr. |
| 112 | | |
| 120 | | Operand 7 |
| 128 | | Operand-8 ALET |
| 136 | | Operand-8 Adr. |



LzLabs GmbH. Ex. 1008-414

The parameter list used for function code 21 has the following format:

Parameter List for Function Code 21




The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the first-operand replacement value is stored at the second-operand location, the third operand is stored at the fourth-operand location, the fifth operand is stored at the sixth-operand location, the seventh operand is stored at the eighth-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

**Locking**

A lock is obtained at the beginning of the operation and released at the end of the operation. The lock obtained is represented by a program lock token (PLT) whose logical address is specified in general register 1 as already described.

A PLT is a value produced by a model-dependent transformation of the PLT logical address. Depending on the model, the PLT may be derived directly from the PLT logical address or, when DAT is on, from the real address that results from transformation of the PLT logical address by DAT. If DAT is used, access-register translation (ART) precedes DAT in the access-register mode.

A PLT selects one of a model-dependent number of locks within the configuration. Programs being executed by different CPUs can be assured of specifying the same lock only by specifying PLT logical addresses that are the same and that can be transformed to the same real address by the different CPUs.

Since a model may or may not use ART and DAT when forming a PLT, access-exception conditions that can be encountered during ART and DAT may or may not be recognized as exceptions. There is no accessing of a location designated by a PLT, but an addressing exception may be recognized for the location. A protection exception is not recognized for any reason during processing of a PLT logical address.

The CPU can hold one lock at a time.

When PERFORM LOCKED OPERATION is executed by this CPU and is to use a lock that is already held by another CPU due to the execution of a PERFORM LOCKED OPERATION instruction by the other CPU, the execution by this CPU is delayed until the lock is no longer held. An excessive delay can be caused only by a machine malfunction and is a machine-check condition.

The order in which multiple requests for the same lock are satisfied is undefined.

A nonrecoverable failure of a CPU while holding a lock may result in a machine check, entry into the check-stop state, or system check stop. The machine check is processing backup if all operands are undamaged or processing damage if


LzLabs GmbH. Ex. 1008-415

register operands are damaged. If a machine check or the check-stop state is the result, either no storage operands have been changed or else all storage operands that were due to be changed have been correctly changed, and, in either case, the lock has been released. If the storage operands are not in either their correct original state or their correct final state, the result is system check stop.

### Storage-Operand References

The accesses to the even-numbered storage operands appear to be word concurrent, as observed by other CPUs, for function codes that are a multiple of 4 or doubleword concurrent for function codes that are one more than a multiple of 4. The accesses to the doublewords in the parameter list appear to be doubleword concurrent, as observed by other CPUs, regardless of the function code.

As observed by other CPUs, all storage operands may be tested for access exceptions before a lock is obtained. (A channel program cannot observe a lock.)

As observed by other CPUs, in all operations except the compare-and-swap operation (which does not have a fourth operand), the fourth operand is accessed while the lock is held only if a comparison of the first-operand comparison value to the second operand while the lock is held has indicated equality. In these operations, the fourth operand is accessed before the lock is held only if a comparison of the first-operand comparison value to the second operand has indicated equality and only if, when DAT is on, an INVALIDATE PAGE TABLE ENTRY instruction executed by another CPU after the fetch of the second operand will not be the cause of a page-translation exception recognized for the fourth operand, which it will if it sets to one the page-invalid bit in the page-table entry for the fourth operand when this CPU does not have a TLB entry corresponding to that page-table entry. In the compare-and-swap-and-double-store and compare-and-swap-and-triple-store operations, the sixth operand, and also the eighth operand in the triple-store operation, are treated the same as the fourth operand described above. The reason for this specification about INVALIDATE PAGE TABLE ENTRY is given in programming note 6 on page 7-113.

When a comparison is made between an operand comparison value and an operand before the lock is obtained and indicates inequality, the lock still is obtained. The condition code is set only as a result of a comparison made while the lock is held. When condition code 1 or 2 is set, the first-operand comparison value or third-operand comparison value is replaced only by means of a fetch of the second operand or fourth operand, respectively, made while the lock is held, as observed by other CPUs.

In those cases when a store is performed to the second-operand location and one or more of the fourth-, sixth-, and eighth-operand locations, the store to the second-operand location is always performed last, as observed by other CPUs and by channel programs.

Stores into the parameter list may be performed while the lock is held or after it has been released.

A serialization operation is performed immediately after the lock is obtained and again immediately before it is released. However, values fetched from the parameter list before the lock is obtained are not necessarily refetched. A serialization operation is not performed if the test bit, bit 23 of general register 0, is one.

Access exceptions may be recognized for parameter-list locations even when the locations are not required in the operation. The locations are those beginning at offset 0 and extending up through the last location defined for the function code used.

For the compare-and-load and compare-and-swap operations, the operation is suppressed on all addressing and protection exceptions.

When a nonrecoverable failure of a CPU while holding a lock results in a machine check or entry into the check-stop state, either no storage operands have been changed or else all storage operands that were due to be changed have been correctly changed. The latter may be accomplished by repeating stores that were performed successfully before the failure. Therefore, there may be two single-access store references (possibly the store part of an update reference and then a store reference) to the store-type operands, with the first value stored equal to the second value stored.



LzLabs GmbH. Ex. 1008-416

***Resulting Condition Code:***

When test bit is zero:

0 All comparisons equal; replacement value or values stored or loaded
1 First-operand comparison not equal; first-operand comparison value replaced
2 -- (all operations except double compare and swap)
2 First-operand comparison equal but third-operand comparison not equal; third-operand comparison value replaced (double compare and swap)
3 --

When test bit is one:

0 Function code valid
1 --
2 --
3 Function code invalid

***Program Exceptions:***

- Access (for all function codes, fetch, except addressing and protection for PLT location, program lock token, model-dependent; for all function codes, fetch and store, operand 2; for function codes 1, 5, 9, 13, 17, and 21, fetch and store, parameter list; for function codes 16 and 20, fetch, parameter list; for function codes 0 and 1, fetch, operand 4; for function codes 8 and 9, fetch and store, operand 4; for function codes 12, 13, 16, 17, 20, and 21, store, operand 4; for function codes 16, 17, 20, and 21, store, operand 6; for function codes 20 and 21, store, operand 8)
- Operation (if the perform-locked-operation facility is not installed)
- Specification

**Programming Notes:**

1. An example of the use of the PERFORM LOCKED OPERATION instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”

2. When the contents of storage locations are changed by PERFORM LOCKED OPERATION instructions that are executed concurrently by different CPUs and that use the same lock, the changes to operands not in the parameter list will be completed by one of the CPUs before they are begun by the other CPU, depending on which CPU first obtains the lock.

3. The compare-and-swap functions of PERFORM LOCKED OPERATION are not performed by means of interlocked-update references. Concurrent store references by another CPU to the storage operands, even if they are interlocked-update references, will interfere unpredictably, in terms of the resulting register and storage contents, with the intended operation of PERFORM LOCKED OPERATION. All changes to the contents of the storage locations must be made by PERFORM LOCKED OPERATION instructions that use the same lock, if predictable storage results are to be obtained.

4. Because a nonrecoverable failure of a CPU while executing PERFORM LOCKED OPERATION may cause two stores of the same value to a store-type operand, a concurrent store made by another CPU to the same operand but not by executing PERFORM LOCKED OPERATION may be lost.

5. When programs in different address spaces are using the same lock when DAT is on, the programs must ensure that they are using PLT logical addresses that are the same and that will be translated to the same real address regardless of the address space in which a translation occurs. Otherwise, the programs may in fact use different locks.

6. The section “Storage-Operand References” on page 7-112 contains a specification concerning the INVALIDATE PAGE TABLE ENTRY (IPTE) instruction. The need for the specification is shown by the following example that is possible without the specification.

   a. CPU 1 begins to execute a PERFORM LOCKED OPERATION instruction with function code 8, which is referred to as PLO.DCS. Operand 2 is a location, Qtail, containing the address (the first-operand comparison value) of the last element, element X, on a queue, and operand 4 is a location in that element containing the address (0, the third-operand comparison value) of the next (nonexisting) element on the queue. The purpose of the PLO instruction is to enqueue an element by placing the address of the element (the


LzLabs GmbH. Ex. 1008-417

first-operand and third-operand replacement values) in both operand 2 and operand 4. With the lock not held, the PLO instruction fetches operand 2 and compares it, with an equal result, to the first-operand comparison value.

b. CPU 2 completely executes a PLO.DCS instruction to dequeue element X, which is the only element on the queue, from the queue. The PLO instruction stores 0 in Qtail and also in Qhead, which is a location containing the address of the first element on the queue. The program on CPU 2 processes the dequeued element and then invokes the freemain service of the control program to deallocate the storage containing the element. The freemain service uses IPTE to set the page-invalid bit to one in the page-table entry for the page containing element X. The IPTE instruction immediately sets the page-invalid bit to one, and then it waits for the signal that all other CPUs have cleared their TLBs of entries corresponding to the page.

c. CPU 1 attempts to fetch operand 4. CPU 1 does not have a TLB entry for the operand-4 page. CPU 1 signals CPU 2 that the CPU 2 IPTE instruction may proceed.

d. CPU 2 completes its IPTE instruction. The program on CPU 2 sets a software bit in the page-table entry to one to indicate that the page has been freemained and that, therefore, a reference to the page should result in presentation by the control program of an addressing exception to the program making the reference.

e. CPU 1 attempts to do DAT for operand 4 and sees that the page-invalid bit is one. CPU 1 performs a program interruption indicating a page-translation exception. The exception handler sees that the software bit indicating freemained is one, and it presents an addressing exception to the CPU 1 program, which causes an abend of the program.

If CPU 1 had had a TLB entry for the page, its PLO instruction would not have been interrupted, and the comparison of the first-operand comparison value to the second operand while the lock was held would indicate that CPU 2 had changed the second operand. The PLO instruction would set condition code 1. If CPU 1 did not have a TLB entry but IPTE could not set the page-invalid bit to one while CPU 1 was executing an instruction, CPU 1 could successfully translate the operand-4 address and, again, discover while the lock was held that operand 2 had changed. The case when operand 2 points to element X but the freemained bit for the element-X page is one is a programming error.

7. Figure 7-70 on page 7-115 summarizes the results of the operation.



LzLabs GmbH. Ex. 1008-418

| Op1c=Op2 | Op3c=Op4 | Cond Code | Action |
|---|---|---|---|
| Function Codes 0 and 1 (Compare and Load) | | | |
| No | - | 1 | Op2 ⟶ Op1c |
| Yes | - | 0 | Op4 ⟶ Op3 |
| Function Codes 4 and 5 (Compare and Swap) | | | |
| No | - | 1 | Op2 ⟶ Op1c |
| Yes | - | 0 | Op1r ⟶ Op2 |
| Function Codes 8 and 9 (Double Compare and Swap) | | | |
| No | - | 1 | Op2 ⟶ Op1c |
| Yes | No | 2 | Op4 ⟶ Op3c |
| Yes | Yes | 0 | Op1r ⟶ Op2 Op3r ⟶ Op4 |
| Function Codes 12 and 13 (Compare and Swap and Store) | | | |
| No | - | 1 | Op2 ⟶ Op1c |
| Yes | - | 0 | Op1r ⟶ Op2 Op3 ⟶ Op4 |
| Function Codes 16 and 17 (Compare and Swap and Double Store) | | | |
| No | - | 1 | Op2 ⟶ Op1c |
| Yes | - | 0 | Op1r ⟶ Op2 Op3 ⟶ Op4<br>Op5 ⟶ Op6 |
| Function Codes 20 and 21 (Compare and Swap and Triple Store) | | | |
| No | - | 1 | Op2 ⟶ Op1c |
| Yes | - | 0 | Op1r ⟶ Op2 Op3 ⟶ Op4<br>Op5 ⟶ Op6<br>Op7 ⟶ Op8 |

**Explanation:**

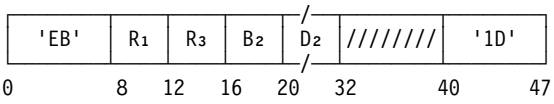
| | |
|---|---|
| - | Not applicable. |
| OpNc | Operand-N comparison value. |
| OpNr | Operand-N replacement value. |

*Figure 7-70. Summary of PERFORM LOCKED OPERATION Results*


LzLabs GmbH. Ex. 1008-419

## ROTATE LEFT SINGLE LOGICAL

```
RLL     R1,R3,D2(B2)          [RSE]
```




The third operand is rotated left the number of bits specified by the second-operand address, and the result is placed at the first-operand location. The third operand remains unchanged in general register $R_3$.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be rotated. The remainder of the address is ignored.

All 32 bits of the third operand participate in a left shift. Each bit shifted out of the leftmost bit position of the operand reenters in the rightmost bit position of the operand.

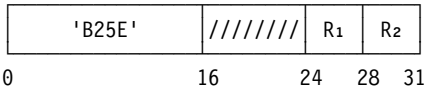
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Operation (if z/Architecture is not installed)

## SEARCH STRING

```
SRST      R1,R2       [RRE]
```




The second operand is searched until a specified character is found, the end of the second operand is reached, or a CPU-determined number of bytes have been searched, whichever occurs first. The CPU-determined number is at least 256. The result is indicated in the condition code.

The location of the leftmost byte of the second operand is designated by the contents of general register $R_2$. The location of the first byte after the second operand is designated by the contents of general register $R_1$.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of general register $R_1$ and $R_2$ constitute the address, and the contents of bit position 0 are ignored.

In the access-register mode, the address space containing the second operand is specified only by means of access register $R_2$. The contents of access register $R_1$ are ignored.

The character for which the search occurs is specified in bit positions 24-31 of general register 0. Bit positions 0-23 of general register 0 are reserved for possible future extensions and must contain all zeros; otherwise, a specification exception is recognized.

The operation proceeds left to right and ends as soon as the specified character has been found in the second operand, the address of the next second-operand byte to be examined equals the address in general register $R_1$, or a CPU-determined number of second-operand bytes have been examined, whichever occurs first. The CPU-determined number is at least 256. When the specified character is found, condition code 1 is set. When the address of the next second-operand byte to be examined equals the address in general register $R_1$, condition code 2 is set. When a CPU-determined number of second-operand bytes have been examined, condition code 3 is set. When the CPU-determined number of second-operand bytes have been examined and the address of the next second-operand byte is in general register $R_1$, it is unpredictable whether condition code 2 or 3 is set.

When condition code 1 is set, the address of the specified character found in the second operand is placed in general register $R_1$, and the contents of general register $R_2$ remain unchanged. When condition code 3 is set, the address of the next byte to be processed in the second operand is placed in general register $R_2$, and the contents of general register $R_1$ remain unchanged. When condition code 2 is set, the contents of general registers $R_1$ and $R_2$ remain unchanged. Whenever an address is placed in a general register, bits 0-7 of the register, in the 24-bit mode, or bit 0, in the 31-bit mode, are set to zeros.



LzLabs GmbH. Ex. 1008-420

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

Access exceptions for the second operand are recognized only for that portion of the operand that is necessarily examined.

The storage-operand-consistency rules are the same as for the COMPARE LOGICAL LONG instruction.

***Resulting Condition Code:***

0 --
1 Specified character found; general register $R_1$ updated with address of character; general register $R_2$ unchanged
2 Specified character not found in entire second operand; general registers $R_1$ and $R_2$ unchanged
3 CPU-determined number of bytes searched; general register $R_1$ unchanged; general register $R_2$ updated with address of next byte

***Program Exceptions:***

- Access (fetch, operand 2)
- Operation (if the string-instruction facility is not installed)
- Specification

**Programming Notes:**

1. Examples of the use of the SEARCH STRING instruction are given in Appendix A, "Number Representation and Instruction-Use Examples"

2. When condition code 3 is set, the program can simply branch back to the instruction to continue the search. The program need not determine the number of bytes that were searched.

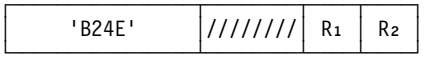
3. When the address in general register $R_1$ equals the address in general register $R_2$, condition code 2 is set immediately, and access exceptions are not recognized. When the address in general register $R_1$ is less than the address in general register $R_2$, condition code 2 can be set only if the operand wraps around from the top of storage to location 0.

4. $R_1$ or $R_2$ may be zero, in which case general register 0 is treated as containing an address and also the specified character.

5. When it is desired to search a string of unknown length for its ending character, and assuming that (1) the string does not start below location 256 (or below location 1 if the ending character is 00 hex), (2) the string does not wrap around to location 0, and (3) the specified character in general register 0 need not be preserved, then $R_1$ can be zero in order to have SEARCH STRING use only two general registers instead of three.

## SET ACCESS

```
SAR     R1,R2     [RRE]
```



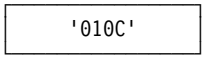

The contents of general register $R_2$ are placed in access register $R_1$.

***Condition Code:*** The code remains unchanged.

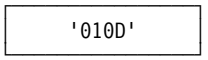
***Program Exceptions:*** None.

## SET ADDRESSING MODE

```
SAM24     [E]
```




```
SAM31     [E]
```




The addressing mode is set by setting the addressing-mode bit, bit 32 of the current PSW, as follows:

| Instruction | PSW Bit 32 | Resulting Addressing Mode |
|---|---|---|
| SAM24 | 0 | 24-bit |
| SAM31 | 1 | 31-bit |


LzLabs GmbH. Ex. 1008-421

The instruction address in the PSW is updated under the control of the new addressing mode, as follows. The value 2 (the instruction length) is added to the contents of bit positions 33-63 of the PSW, or the value 4 is added if the instruction is the target of EXECUTE. In either case, a carry out of bit position 33 is ignored. Then bits 33-39 of the PSW are set to zeros if the new addressing mode is the 24-bit mode.

The instruction is completed only if the new addressing mode and the unupdated instruction address in the PSW are a valid combination. When the new addressing mode is to be the 24-bit mode, bits 33-39 of the unupdated PSW must be all zeros; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

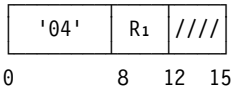
***Program Exceptions:***

- Operation (if z/Architecture is not installed)
- Specification (SAM24 only)

**Programming Note:** Checking the unupdated instruction address prevents completion in a major case: the instruction is located at address $2^{24}$ or above and the new addressing mode is to be the 24-bit mode. In this case, if the instruction were completed, the updating of the instruction address under the control of the new addressing mode would cause one or more leftmost bits of the address to be set to zeros, which would cause the next instruction to be fetched from other than the next sequential location. This action is sometimes called a "wild branch." A wild branch still can occur if the instruction is located at $2^{24}$ - 2, or at $2^{24}$ - 4 if EXECUTE is used.

## SET PROGRAM MASK

```
SPM    R1          [RR]
```




The first operand is used to set the condition code and the program mask of the current PSW.

Bits 2 and 3 of general register $R_1$ replace the condition code, and bits 4-7 replace the program mask. Bits 0, 1, and 8-31 of general register $R_1$ are ignored.

***Condition Code:*** The code is set as specified by bits 2 and 3 of general register $R_1$.

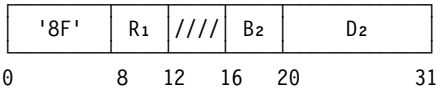
***Program Exceptions:*** None.

**Programming Notes:**

1. Bits 2-7 of the general register may have been loaded from the PSW by execution of BRANCH AND LINK in the 24-bit addressing mode or by execution of INSERT PROGRAM MASK in either the 24-bit or 31-bit addressing mode.
2. SET PROGRAM MASK permits setting of the condition code and the mask bits in either the problem state or the supervisor state.
3. The program should take into consideration that the setting of the program mask can have a significant effect on subsequent execution of the program. Not only do the four mask bits control whether the corresponding interruptions occur, but the exponent-underflow and significance masks also determine the result which is obtained.

## SHIFT LEFT DOUBLE

```
SLDA   R1,D2(B2)          [RS]
```




The 63-bit numeric part of the signed first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 64-bit signed binary integer. The sign position of the even-numbered register remains unchanged. The leftmost bit position of the odd-numbered register contains a numeric bit, which participates in the



LzLabs GmbH. Ex. 1008-422

shift in the same manner as the other numeric bits. Zeros are supplied to the vacated bit positions on the right.

If one or more bits unlike the sign bit are shifted out of bit position 1 of the even-numbered register, an overflow occurs, and condition code 3 is set. If the fixed-point-overflow mask bit is one, a program interruption for fixed-point overflow occurs.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

***Program Exceptions:***

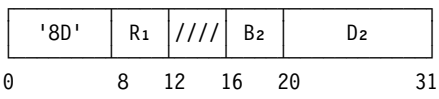
- Fixed-point overflow
- Specification

**Programming Notes:**

1. An example of the use of the SHIFT LEFT DOUBLE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The eight shift instructions provide the following three pairs of alternatives: left or right, single or double, and signed or logical. The signed shifts differ from the logical shifts in that, in the signed shifts, overflow is recognized, the condition code is set, and the leftmost bit participates as a sign.
3. A zero shift amount in the two signed double-shift operations provides a double-length sign and magnitude test.
4. The base register participating in the generation of the second-operand address permits indirect specification of the shift amount by means of placement of the shift amount in the base register. A zero in the $B_2$ field indicates the absence of indirect shift specification.

## SHIFT LEFT DOUBLE LOGICAL

SLDL $R_1$,$D_2$($B_2$) [RS]




The 64-bit first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 0 of the even-numbered register are not inspected and are lost. Zeros are supplied to the vacated bit positions on the right.

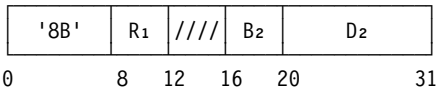
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Specification

## SHIFT LEFT SINGLE

SLA $R_1$,$D_2$($B_2$) [RS]




The 31-bit numeric part of the signed first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 32-bit signed binary integer. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the left shift. Zeros are supplied to the vacated bit positions on the right.

If one or more bits unlike the sign bit are shifted out of bit position 1, an overflow occurs, and condition code 3 is set. If the fixed-point-overflow


LzLabs GmbH. Ex. 1008-423

mask bit is one, a program interruption for fixed-point overflow occurs.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

***Program Exceptions:***

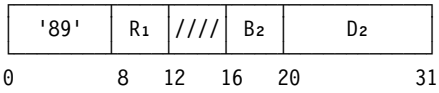
- Fixed-point overflow

**Programming Notes:**

1. An example of the use of the SHIFT LEFT SINGLE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. For numbers with a value greater than or equal to $-2^{30}$ and less than $2^{30}$, a left shift of one bit position is equivalent to multiplying the number by 2.
3. Shift amounts from 31 to 63 cause the entire numeric part to be shifted out of the register, leaving a result of the maximum negative number or zero, depending on whether or not the initial contents were negative.

## SHIFT LEFT SINGLE LOGICAL

SLL R1,D2(B2) [RS]




The 32-bit first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

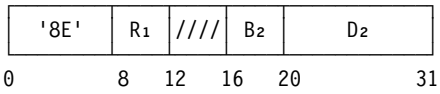
All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 0 are not inspected and are lost. Zeros are supplied to the vacated bit positions on the right.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:*** None.

## SHIFT RIGHT DOUBLE

SRDA R1,D2(B2) [RS]




The 63-bit numeric part of the signed first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 64-bit signed binary integer. The sign position of the even-numbered register remains unchanged. The left-most bit position of the odd-numbered register contains a numeric bit, which participates in the shift in the same manner as the other numeric bits. Bits shifted out of bit position 31 of the odd-numbered register are not inspected and are lost. Bits equal to the sign are supplied to the vacated bit positions on the left.

***Resulting Condition Code:***

0 Result zero
1 Result less than zero
2 Result greater than zero
3 --

***Program Exceptions:***

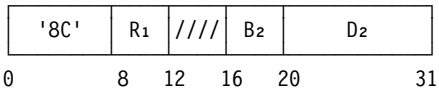
- Specification



LzLabs GmbH. Ex. 1008-424

## SHIFT RIGHT DOUBLE LOGICAL

SRDL R1,D2(B2) [RS]




The 64-bit first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 31 of the odd-numbered register are not inspected and are lost. Zeros are supplied to the vacated bit positions on the left.

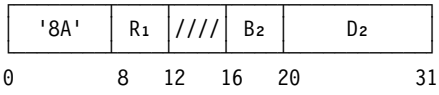
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Specification

## SHIFT RIGHT SINGLE

SRA R1,D2(B2) [RS]




The 31-bit numeric part of the signed first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 32-bit signed binary integer. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the right shift. Bits shifted out of bit position 31 are not inspected and are lost. Bits equal to the sign are supplied to the vacated bit positions on the left.

***Resulting Condition Code:***

0 Result zero
1 Result less than zero
2 Result greater than zero
3 --

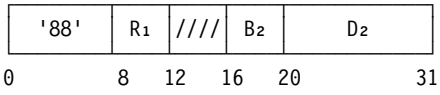
***Program Exceptions:*** None.

**Programming Notes:**

1. A right shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the result is equivalent to dividing the number by 2. When an odd number is shifted right one position, the result is equivalent to dividing the *next lower* number by 2. For example, +5 shifted right by one bit position yields +2, whereas -5 yields -3.
2. Shift amounts from 31 to 63 cause the entire numeric part to be shifted out of the register, leaving a result of -1 or zero, depending on whether or not the initial contents were negative.

## SHIFT RIGHT SINGLE LOGICAL

SRL R1,D2(B2) [RS]




The 32-bit first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 31 are not


LzLabs GmbH. Ex. 1008-425

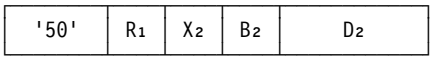
inspected and are lost. Zeros are supplied to the vacated bit positions on the left.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:*** None.

## STORE

```
ST      R1,D2(X2,B2)      [RX]
```




The first operand is placed unchanged at the second-operand location.

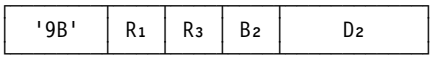
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)

## STORE ACCESS MULTIPLE

```
STAM    R1,R3,D2(B2)      [RS]
```




The contents of the set of access registers starting with access register $R_1$ and ending with access register $R_3$ are stored at the locations designated by the second-operand address.

The storage area where the contents of the access registers are placed starts at the location designated by the second-operand address and continues through as many storage words as the number of access registers specified. The contents of the access registers are stored in ascending order of their register numbers, starting with access register $R_1$ and continuing up to and including access register $R_3$, with access register 0 following access register 15. The contents of the access registers remain unchanged.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

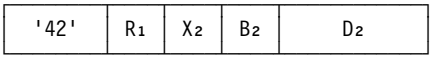
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Specification

## STORE CHARACTER

```
STC     R1,D2(X2,B2)      [RX]
```




Bits 24-31 of general register $R_1$ are placed unchanged at the second-operand location. The second operand is one byte in length.

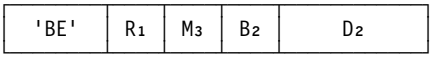
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)

## STORE CHARACTERS UNDER MASK

```
STCM    R1,M3,D2(B2)      [RS]
```




Bytes selected from general register $R_1$ under control of a mask are placed at contiguous byte locations beginning at the second-operand address.

The contents of the $M_3$ field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of general register $R_1$. The bytes corresponding to ones in the mask are placed in the same order at successive and contiguous storage locations beginning at the second-operand address. When the mask is not zero, the length of the second operand is equal to the number of ones in the mask. The contents of the general register remain unchanged.

When the mask is not zero, exceptions associated with storage-operand accesses are recognized only for the number of bytes specified by the mask.



LzLabs GmbH. Ex. 1008-426

When the mask is zero, the single byte designated by the second-operand address remains unchanged; however, on some models, the contents may be fetched and subsequently stored back unchanged at the same storage location. This update appears to be an interlocked-update reference as observed by other CPUs.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

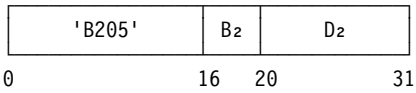
- Access (store, operand 2)

**Programming Notes:**

1. An example of the use of the STORE CHARACTERS UNDER MASK instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. STORE CHARACTERS UNDER MASK with a mask of 0111 may be used to store a three-byte address, for example, in modifying the address in a CCW.
3. STORE CHARACTERS UNDER MASK with a mask of 1111, 0011, or 0001 performs the same function as STORE, STORE HALFWORD, or STORE CHARACTER, respectively. However, on most models, the performance of STORE CHARACTERS UNDER MASK is slower.
4. Using STORE CHARACTERS UNDER MASK with a zero mask should be avoided since this instruction, depending on the model, may perform a fetch and store of the single byte designated by the second-operand address. This reference is not interlocked against accesses by channel programs. In addition, it may cause any of the following to occur for the byte designated by the second-operand address: a PER storage-alteration event may be recognized; access exceptions may be recognized; and, provided no access exceptions exist, the change bit may be set to one. Because the contents of storage remain unchanged, the change bit may or may not be one when a PER storage-alteration event is recognized.

# STORE CLOCK

STCK $D_2(B_2)$ [S]




The current value of bits 0-63 of the TOD clock is stored in the eight-byte field designated by the second-operand address, provided the clock is in the set, stopped, or not-set state.

When the clock is stopped, zeros are stored in positions to the right of the rightmost bit position that is incremented when the clock is running. When the value of a running clock is stored, nonzero values may be stored in positions to the right of the rightmost incremented bit; this is to ensure that a unique value is stored.

Zeros are stored at the operand location when the clock is in the error state or the not-operational state.

The quality of the clock value stored by the instruction is indicated by the resultant condition-code setting.

A serialization function is performed before the value of the clock is fetched and again after the value is placed in storage.

***Resulting Condition Code:***

0 Clock in set state
1 Clock in not-set state
2 Clock in error state
3 Clock in stopped state or not-operational state

***Program Exceptions:***

- Access (store, operand 2)

**Programming Notes:**

1. Bit position 31 of the clock is incremented every 1.048576 seconds; hence, for timing applications involving human responses, the leftmost clock word may provide sufficient resolution.
2. Condition code 0 normally indicates that the clock has been set by the control program. Accordingly, the value may be used in elapsed-time measurements and as a valid

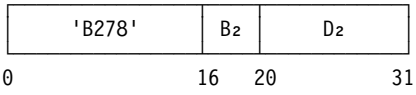

LzLabs GmbH. Ex. 1008-427

time-of-day and calendar indication. Condition code 1 indicates that the clock value is the elapsed time since the power for the clock was turned on. In this case, the value may be used in elapsed-time measurements but is not a valid time-of-day indication. Condition codes 2 and 3 mean that the value provided by STORE CLOCK cannot be used for time measurement or indication.

3. If a problem program written for ESA/390 is to be executed also on a system in the System/370 mode, then the program should take into account that, in the System/370 mode, the value stored when the condition code is 2 is not necessarily zero.

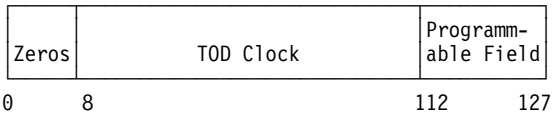
## STORE CLOCK EXTENDED

```
STCKE  D2(B2)            [S]
```




The current value of bits 0-103 of the TOD clock is stored in byte positions 1-13 of the sixteen-byte field designated by the second-operand address, provided the clock is in the set, stopped, or not-set state. Zeros are stored in byte position 0. The TOD programmable field, bits 16-31 of the TOD programmable register, is stored in byte positions 14 and 15.

The operand just described has the following format:




When the clock is stopped, zeros are stored in the clock value in positions to the right of the right-most bit position that is incremented when the clock is running. The programmable field still is stored.

When the value of a running clock is stored, the value in bit positions 64-103 of the clock (bit positions 72-111 of the storage operand) is always nonzero; this ensures that values stored by STORE CLOCK EXTENDED are unique when compared with values stored by STORE CLOCK and extended with zeros.

Zeros are stored at the operand location when the clock is in the error state or the not-operational state.

The quality of the clock value stored by the instruction is indicated by the resultant condition-code setting.

A serialization function is performed before the value of the clock is fetched and again after the value is placed in storage.

***Resulting Condition Code:***

0 Clock in set state
1 Clock in not-set state
2 Clock in error state
3 Clock in stopped state or not-operational state

***Program Exceptions:***

- Access (store, operand 2)
- Operation (if the extended-TOD-clock facility is not installed)

**Programming Notes:**

1. Condition code 0 normally indicates that the clock has been set by the control program. Accordingly, the value may be used in elapsed-time measurements and as a valid time-of-day and calendar indication. Condition code 1 indicates that the clock value is the elapsed time since the power for the clock was turned on. In this case, the value may be used in elapsed-time measurements but is not a valid time-of-day indication. Condition codes 2 and 3 mean that the value provided by STORE CLOCK EXTENDED cannot be used for time measurement or indication.
2. Programming notes beginning on page 4-33 show hex values related to the TOD clock as it is stored by the STORE CLOCK instruction. Notes 3-5, below, are repetitions of those notes except with the text and hex values adjusted so they apply to bits 0-71 of the value stored by STORE CLOCK EXTENDED.
3. The following chart shows the time interval between instants at which various bits of the TOD-clock value stored by STORE CLOCK EXTENDED are stepped. This time value may also be considered as the weighted time value that the bit, when one, represents. The



LzLabs GmbH. Ex. 1008-428

bit numbers are those of the STORE CLOCK EXTENDED operand.

| STCKE Bit | Stepping Interval | | | |
|---|---|---|---|---|
| | Days | Hours | Min. | Seconds |
| 59 | | | | 0.000 001 |
| 55 | | | | 0.000 016 |
| 51 | | | | 0.000 256 |
| 47 | | | | 0.004 096 |
| 43 | | | | 0.065 536 |
| 39 | | | | 1.048 576 |
| 35 | | | | 16.777 216 |
| 31 | | | 4 | 28.435 456 |
| 27 | | 1 | 11 | 34.967 296 |
| 23 | | 19 | 5 | 19.476 736 |
| 19 | 12 | 17 | 25 | 11.627 776 |
| 15 | 203 | 14 | 43 | 6.044 416 |
| 11 | 3257 | 19 | 29 | 36.710 656 |

4. The following chart shows the setting of bits 0-63 of the STORE CLOCK EXTENDED operand for 00:00:00 (0 am), UTC time, for several dates: January 1, 1900, January 1, 1972, and for that instant in time just after each of the 22 leap seconds that have occurred through January, 1999. Each of these leap seconds was inserted in the UTC time scale beginning at 23:59:60 UTC of the day previous to the one listed and ending at 00:00:00 UTC of the day listed.

| Year | Mth | Day | Leap Sec. | STCKE Value (Hex) Bits 0-63 |
|---|---|---|---|---|
| 1900 | 1 | 1 | | 0000 0000 0000 0000 |
| 1972 | 1 | 1 | | 0081 26D6 0E46 0000 |
| 1972 | 7 | 1 | 1 | 0082 0BA9 811E 2400 |
| 1973 | 1 | 1 | 2 | 0082 F300 AEE2 4800 |
| 1974 | 1 | 1 | 3 | 0084 BDE9 7114 6C00 |
| 1975 | 1 | 1 | 4 | 0086 88D2 3346 9000 |
| 1976 | 1 | 1 | 5 | 0088 53BA F578 B400 |
| 1977 | 1 | 1 | 6 | 008A 1FE5 9520 D800 |
| 1978 | 1 | 1 | 7 | 008B EACE 5752 FC00 |
| 1979 | 1 | 1 | 8 | 008D B5B7 1985 2000 |
| 1980 | 1 | 1 | 9 | 008F 809F DBB7 4400 |
| 1981 | 7 | 1 | 10 | 0092 305C 0FCD 6800 |
| 1982 | 7 | 1 | 11 | 0093 FB44 D1FF 8C00 |
| 1983 | 7 | 1 | 12 | 0095 C62D 9431 B000 |
| 1985 | 7 | 1 | 13 | 0099 5D40 F517 D400 |
| 1988 | 1 | 1 | 14 | 009D DA69 A557 F800 |
| 1990 | 1 | 1 | 15 | 00A1 717D 063E 1C00 |
| 1991 | 1 | 1 | 16 | 00A3 3C65 C870 4000 |
| 1992 | 7 | 1 | 17 | 00A5 EC21 FC86 6400 |
| 1993 | 7 | 1 | 18 | 00A7 B70A BEB8 8800 |
| 1994 | 7 | 1 | 19 | 00A9 81F3 80EA AC00 |
| 1996 | 1 | 1 | 20 | 00AC 3433 6FEC D000 |
| 1997 | 7 | 1 | 21 | 00AE E3EF A402 F400 |
| 1999 | 1 | 1 | 22 | 00B1 962F 9305 1800 |

5. The stepping value of TOD-clock bit position 63, if implemented, is $2^{-12}$ microseconds, or approximately 244 picoseconds. This value is called a clock unit.

   The following chart shows various time intervals in clock units expressed in hexadecimal notation. The chart shows the values stored in bit positions 0-71 of the STORE CLOCK EXTENDED operand. Bit 71 of the operand represents a clock unit.

| Interval | Clock Units (Hex) Bits 0-71 |
|---|---|
| 1 microsecond | 0010 00 |
| 1 millisecond | 3E80 00 |
| 1 second | 00F4 2400 00 |
| 1 minute | 3938 7000 00 |
| 1 hour | 000D 693A 4000 00 |
| 1 day | 0141 DD76 0000 00 |
| 365 days | 0001 CAE8 C13E 0000 00 |
| 366 days | 0001 CC2A 9EB4 0000 00 |
| 1,461 days* | 0007 2CE4 E26E 0000 00 |

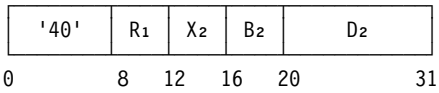
* Number of days in four years, including a leap year. Note that the year 1900 was not a leap year. Thus, the four-year span starting in 1900 has only 1,460 days.



LzLabs GmbH. Ex. 1008-429

## STORE HALFWORD

STH R1,D2(X2,B2) [RX]




Bits 16-31 of general register R1 are placed unchanged at the second-operand location. The second operand is two bytes in length.

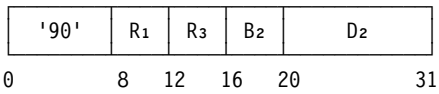
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)

## STORE MULTIPLE

STM R1,R3,D2(B2) [RS]




The contents of the set of general registers starting with general register R1 and ending with general register R3 are placed in the storage area beginning at the location designated by the second-operand address and continuing through as many locations as needed.

The general registers are stored in the ascending order of their register numbers, starting with general register R1 and continuing up to and including general register R3, with general register 0 following general register 15.

***Condition Code:*** The code remains unchanged.

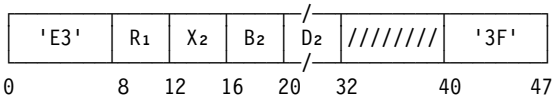
***Program Exceptions:***

- Access (store, operand 2)

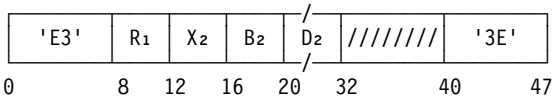
**Programming Note:** An example of the use of the STORE MULTIPLE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

## STORE REVERSED

STRVH R1,D2(X2,B2) [RXE]




STRV R1,D2(X2,B2) [RXE]




The first operand is placed at the second-operand location with the left-to-right sequence of the bytes reversed.

For STORE REVERSED (STRVH), the first operand is two bytes in bit positions 16-31 of general register R1. For STORE REVERSED (STRV), the first operand is four bytes.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Operation (if z/Architecture is not installed)

**Programming Notes:**

1. The instruction can be used to convert two or four bytes from a "little-endian" format to a "big-endian" format, or vice versa. In the big-endian format, the bytes in a left-to-right sequence are in the order most significant to least significant. In the little-endian format, the bytes are in the order least significant to most significant. For example, the bytes ABCD in the big-endian format are DCBA in the little-endian format.
2. The storage-operand references of STORE REVERSED may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)



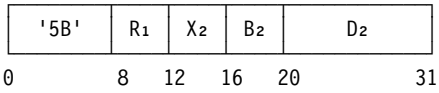
LzLabs GmbH. Ex. 1008-430

# SUBTRACT

SR R1,R2 [RR]




S R1,D2(X2,B2) [RX]




The second operand is subtracted from the first operand, and the difference is placed at the first-operand location. The operands and the difference are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

***Program Exceptions:***

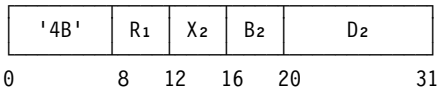
- Access (fetch, operand 2 of S only)
- Fixed-point overflow

**Programming Notes:**

1. When, in the RR format, $R_1$ and $R_2$ designate the same register, subtracting is equivalent to clearing the register.
2. Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

# SUBTRACT HALFWORD

SH R1,D2(X2,B2) [RX]




The second operand is subtracted from the first operand, and the difference is placed at the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand and the difference are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

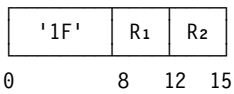
***Program Exceptions:***

- Access (fetch, operand 2)
- Fixed-point overflow

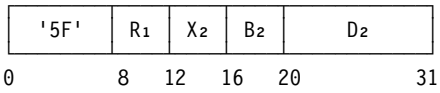
**Programming Note:** The function of a SUBTRACT HALFWORD IMMEDIATE instruction, which is an instruction not provided, can be obtained by using an ADD HALFWORD IMMEDIATE instruction with a negative $I_2$ field.

# SUBTRACT LOGICAL

SLR R1,R2 [RR]




SL R1,D2(X2,B2) [RX]





LzLabs GmbH. Ex. 1008-431

The second operand is subtracted from the first operand, and the difference is placed at the first-operand location. The operands and the difference are treated as 32-bit unsigned binary integers.

***Resulting Condition Code:***

0 --
1 Result not zero; borrow
2 Result zero; no borrow
3 Result not zero; no borrow

***Program Exceptions:***

- Access (fetch, operand 2 of SL only)

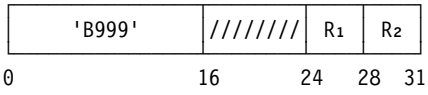
**Programming Notes:**

1. Logical subtraction is performed by adding the one's complement of the second operand and a value of one to the first operand. The use of the one's complement and the value of one instead of the two's complement of the second operand results in a carry when the second operand is zero.
2. SUBTRACT LOGICAL differs from SUBTRACT only in the meaning of the condition code and in the absence of the interruption for overflow.
3. A zero difference is always accompanied by a carry out of bit position 0.
4. The condition-code setting for SUBTRACT LOGICAL can also be interpreted as indicating the presence or absence of a carry, as follows:

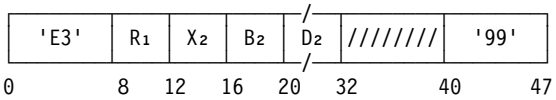
   1 Result not zero; no carry
   2 Result zero; carry
   3 Result not zero; carry

# SUBTRACT LOGICAL WITH BORROW

SLBR R1,R2 [RRE]




SLB R1,D2(X2,B2) [RXE]




The second operand and the borrow are subtracted from the first operand, and the difference is placed at the first-operand location. The operands, the borrow, and the difference are treated as 32-bit unsigned binary integers.

***Resulting Condition Code:***

0 Result zero; borrow
1 Result not zero; borrow
2 Result zero; no borrow
3 Result not zero; no borrow

***Program Exceptions:***

- Access (fetch, operand 2 of SLB only)
- Operation (if z/Architecture is not installed)

**Programming Notes:**

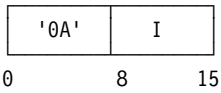
1. A borrow is represented by a zero value of bit 18 of the current PSW. Bit 18 is the leftmost bit of the two-bit condition code in the PSW. Bit 18 is set to zero by an execution of a SUBTRACT LOGICAL or SUBTRACT LOGICAL WITH BORROW instruction that produces a borrow into bit position 0 of the result.
2. Logical subtraction with borrow is performed by adding the one's complement of the second operand and bit 18 of the current PSW to the first operand. Therefore, when bit 18 is one, indicating no borrow, the addition is the same as for SUBTRACT LOGICAL.
3. Condition code zero is set for SUBTRACT LOGICAL WITH BORROW when the maximum 32-bit unsigned binary integer, $2^{32}-1$, is subtracted from zero when PSW bit 18 indicates a borrow.
4. SUBTRACT and SUBTRACT LOGICAL may provide better performance than SUBTRACT LOGICAL WITH BORROW, depending on the model.



LzLabs GmbH. Ex. 1008-432

## SUPERVISOR CALL

```
SVC    I          [RR]
```




The instruction causes a supervisor-call interruption, with the I field of the instruction providing the rightmost byte of the interruption code.

Bits 8-15 of the instruction, with eight zeros appended on the left, are placed in the supervisor-call interruption code that is stored in the course of the interruption. See "Supervisor-Call Interruption" on page 6-46.

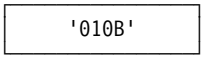
A serialization and checkpoint-synchronization function is performed.

***Condition Code:*** The code remains unchanged and is saved as part of the old PSW. A new condition code is loaded as part of the supervisor-call interruption.

***Program Exceptions:*** None.

## TEST ADDRESSING MODE

```
TAM      [E]
```




The addressing-mode bit, bit 32 of the current PSW, is tested, and the result is indicated in the condition code.

***Resulting Condition Code:***

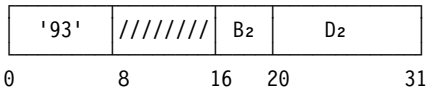
0 PSW bit 32 zero (indicating 24-bit addressing mode)
1 PSW bit 32 one (indicating 31-bit addressing mode)
2 --
3 --

***Program Exceptions:***

- Operation (if z/Architecture is not installed)

## TEST AND SET

```
TS     D2(B2)          [S]
```




The leftmost bit (bit 0) of the byte located at the second-operand address is used to set the condition code, and then the byte is set to all ones.

The byte in storage is set to all ones as it is fetched for the testing of bit 0. This update appears to be an interlocked-update reference as observed by other CPUs.

A serialization function is performed before the byte is fetched and again after the storing of all ones.

***Resulting Condition Code:***

0 Leftmost bit zero
1 Leftmost bit one
2 --
3 --

***Program Exceptions:***

- Access (fetch and store, operand 2)

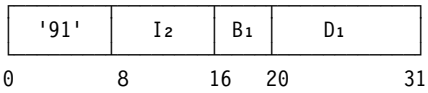
**Programming Notes:**

1. TEST AND SET may be used for controlled sharing of a common storage area by programs operating on different CPUs. This instruction is provided primarily for compatibility with programs written for System/360. The instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP provide functions which are more suitable for sharing among programs on a single CPU or for programs that may be interrupted. See the description of these instructions and the associated programming notes for details.
2. TEST AND SET does not interlock against storage accesses by channel programs. Therefore, the instruction should not be used to update a location into which a channel program may store, since the channel-program data may be lost.


LzLabs GmbH. Ex. 1008-433

## TEST UNDER MASK

```
TM     D1(B1),I2        [SI]
```




A mask is used to select bits of the first operand, and the result is indicated in the condition code.

The byte of immediate data, $I_2$, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the byte in storage designated by the first-operand address.

A mask bit of one indicates that the storage bit is to be tested. When the mask bit is zero, the storage bit is ignored. When all storage bits thus selected are zero, condition code 0 is set. Condition code 0 is also set when the mask is all zeros. When the selected bits are all ones, condition code 3 is set; otherwise, condition code 1 is set.

Access exceptions associated with the storage operand are recognized for one byte even when the mask is all zeros.

***Resulting Condition Code:***

0 Selected bits all zeros; or mask bits all zeros
1 Selected bits mixed zeros and ones
2 --
3 Selected bits all ones

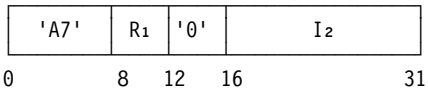
***Program Exceptions:***

- Access (fetch, operand 1)

**Programming Note:** An example of the use of the TEST UNDER MASK instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

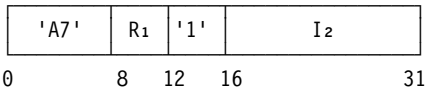
## TEST UNDER MASK HIGH

```
TMH    R1,I2      [RI]
```




## TEST UNDER MASK LOW

```
TML    R1,I2      [RI]
```




A mask is used to select bits of the first operand, and the result is indicated in the condition code.

The contents of the $I_2$ field are used as a 16-bit mask. The bits of the mask are made to correspond one for one with 16 bits of the first operand. For TEST UNDER MASK HIGH, the mask is made to correspond with bits 0-15 of the first operand. For TEST UNDER MASK LOW, the mask is made to correspond with bits 16-31 of the first operand.

A mask bit of one indicates that the first-operand bit is to be tested. When the mask bit is zero, the first-operand bit is ignored. When all first-operand bits thus selected are zero, condition code 0 is set. Condition code 0 is also set when the mask is all zeros. When the selected bits are mixed zeros and ones, condition code 1 is set if the leftmost selected bit is zero, or condition code 2 is set if the leftmost selected bit is one. When the selected bits are all ones, condition code 3 is set.

***Resulting Condition Code:***

0 Selected bits all zeros; or mask bits all zeros
1 Selected bits mixed zeros and ones, and leftmost is zero
2 Selected bits mixed zeros and ones, and leftmost is one
3 Selected bits all ones

***Program Exceptions:***

- Operation (if the immediate-and-relative-instruction facility is not installed)

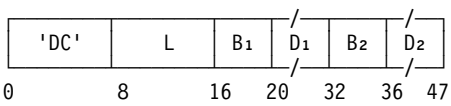
**Programming Note:** When the mask selects exactly two bits, the two selected bits effectively are loaded into the condition code.



LzLabs GmbH. Ex. 1008-434

# TRANSLATE

TR D1(L,B1),D2(B2) [SS]




The bytes of the first operand are used as eight-bit arguments to reference a list designated by the second-operand address. Each function byte selected from the list replaces the corresponding argument in the first operand.

The L field specifies the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding from left to right. Each argument byte is added to the initial second-operand address. The addition is performed following the rules for address arithmetic, with the argument byte treated as an eight-bit unsigned binary integer and extended with zeros on the left. The sum is used as the address of the function byte, which then replaces the original argument byte.

The operation proceeds until the first-operand field is exhausted. The list is not altered unless an overlap occurs.

When the operands overlap, the result is obtained as if each result byte were stored immediately after fetching the corresponding function byte.

Access exceptions are recognized only for those bytes in the second operand which are actually required.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2; fetch and store, operand 1)

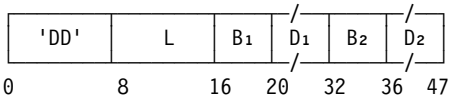
**Programming Notes:**

1. An example of the use of the TRANSLATE instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. TRANSLATE may be used to convert data from one code to another code.
3. The instruction may also be used to rearrange data. This may be accomplished by placing a pattern in the destination area, by designating the pattern as the first operand of TRANSLATE, and by designating the data that is to be rearranged as the second operand. Each byte of the pattern contains an eight-bit number specifying the byte destined for this position. Thus, when the instruction is executed, the pattern selects the bytes of the second operand in the desired order.
4. Because each eight-bit argument byte is added to the initial second-operand address to obtain the address of a function byte, the list may contain 256 bytes. In cases where it is known that not all eight-bit argument values will occur, it is possible to reduce the size of the list.
5. Significant performance degradation is possible when, with DAT on, the second-operand address of TRANSLATE designates a location that is less than 256 bytes to the left of a 4K-byte boundary. This is because the machine may perform a trial execution of the instruction to determine if the second operand actually crosses the boundary.
6. The fetch and subsequent store accesses to a particular byte in the first-operand field do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in “Multiprogramming and Multiprocessing Examples” in Appendix A, “Number Representation and Instruction-Use Examples” on page A-1.
7. The storage-operand references of TRANSLATE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)


LzLabs GmbH. Ex. 1008-435

## TRANSLATE AND TEST

```
TRT    D1(L,B1),D2(B2)          [SS]
```




The bytes of the first operand are used as eight-bit arguments to select function bytes from a list designated by the second-operand address. The first nonzero function byte is inserted in general register 2, and the related argument address in general register 1.

The L field specifies the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding from left to right. The first operand remains unchanged in storage. Calculation of the address of the function byte is performed as in the TRANSLATE instruction. The function byte retrieved from the list is inspected for a value of zero.

When the function byte is zero, the operation proceeds with the next byte of the first operand. When the first-operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting condition code 0. The contents of general registers 1 and 2 remain unchanged.

When the function byte is nonzero, the operation is completed by inserting the function byte in general register 2 and the related argument address in general register 1. This address points to the argument byte last translated. The function byte replaces bits 24-31 of general register 2, and bits 0-23 of this register remain unchanged. In the 24-bit addressing mode, the address replaces bits 8-31 of general register 1, and bits 0-7 of this register remain unchanged. In the 31-bit addressing mode, the address replaces bits 1-31 of general register 1, and bit 0 of this register is set to zero.

When the function byte is nonzero, either condition code 1 or 2 is set, depending on whether the argument byte is the rightmost byte of the first operand. Condition code 1 is set if one or more argument bytes remain to be translated. Condition code 2 is set if no more argument bytes remain.

The contents of access register 1 always remain unchanged.

Access exceptions are recognized only for those bytes in the second operand which are actually required. Access exceptions are not recognized for those bytes in the first operand which are to the right of the first byte for which a nonzero function byte is obtained.

***Resulting Condition Code:***

0 All function bytes zero
1 Nonzero function byte; first-operand field not exhausted
2 Nonzero function byte; first-operand field exhausted
3 --

***Program Exceptions:***

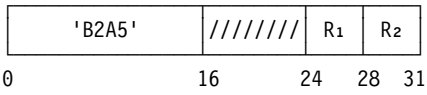
- Access (fetch, operands 1 and 2)

**Programming Notes:**

1. An example of the use of the TRANSLATE AND TEST instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. TRANSLATE AND TEST may be used to scan the first operand for characters with special meaning. The second operand, or list, is set up with all-zero function bytes for those characters to be skipped over and with nonzero function bytes for the characters to be detected.

## TRANSLATE EXTENDED

```
TRE    R1,R2     [RRE]
```




The bytes of the first operand are compared to a test byte in general register 0 and, unless an equal comparison occurs, are used as eight-bit arguments to reference a 256-byte translation table designated by the second-operand address. Each function byte selected from the second operand replaces the corresponding argument in the first operand. The operation proceeds until a first-operand byte equal to the test byte is



LzLabs GmbH. Ex. 1008-436

encountered, the end of the first operand is reached, or a CPU-determined number of bytes have been processed, whichever occurs first. The result is indicated in the condition code.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively. The number of bytes in the first-operand location is specified by bits 0-31 of general register $R_1 + 1$. The contents of general register $R_1 + 1$ are treated as a 32-bit unsigned binary integer.

The handling of the addresses in general registers $R_1$ and $R_2$ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-31 of the registers constitute the address, and the contents of bit position 0 are ignored.

The test byte is in bit positions 24-31 of general register 0, and the contents of bit positions 0-23 of this register are ignored.

The contents of the registers just described are shown in Figure 7-71 on page 7-134.

The bytes of the first operand are selected one by one for translation, proceeding left to right. Each argument byte is first compared to the test byte in general register 0. If the result is an equal comparison, the operation is completed. If the argument byte is not equal to the test byte, the argument byte is added to the initial second-operand address. The addition is performed following the rules for address arithmetic, with the argument byte treated as an eight-bit unsigned binary integer and extended with zeros on the left. The sum is used as the address of the function byte, which then replaces the original argument byte. The second operand is not altered unless an overlap occurs.

The operation proceeds until a first-operand byte equal to the test byte is encountered, the first-operand location is exhausted, or a CPU-determined number of first-operand bytes have been processed.

When a first-operand byte equal to the test byte is encountered, condition code 1 is set. When the first-operand location is exhausted without finding a byte equal to the test byte, condition code 0 is set. When a CPU-determined number of bytes have been processed, condition code 3 is set. Condition code 3 may be set even when the next byte to be processed is equal to the test byte or when the first-operand location is exhausted. In these cases, condition code 1 or 0, respectively, will be set when the instruction is executed again.

If the operation is completed with condition code 0, the contents of general register $R_1$ are incremented by the contents of general register $R_1 + 1$ and then the contents of general register $R_1 + 1$ are set to zero. If the operation is completed with condition code 1, the contents of general register $R_1 + 1$ are decremented by the number of bytes processed before the first-operand byte equal to the test byte was encountered, and the contents of general register $R_1$ are incremented by the same number, so that general register $R_1$ contains the address of the equal byte. If the operation is completed with condition code 3, the contents of general register $R_1 + 1$ are decremented by the number of bytes processed, and the contents of general register $R_1$ are incremented by the same number, so that the instruction, when reexecuted, resumes at the next byte to be processed. When general register $R_1$ is updated, the bits in it that are not part of the address may be set to zeros or may remain unchanged from their original values.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the $R_1$ register is the same register as the $R_2$ register, the results are unpredictable.

When the $R_1$ register or the $R_2$ register is zero, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.


LzLabs GmbH. Ex. 1008-437

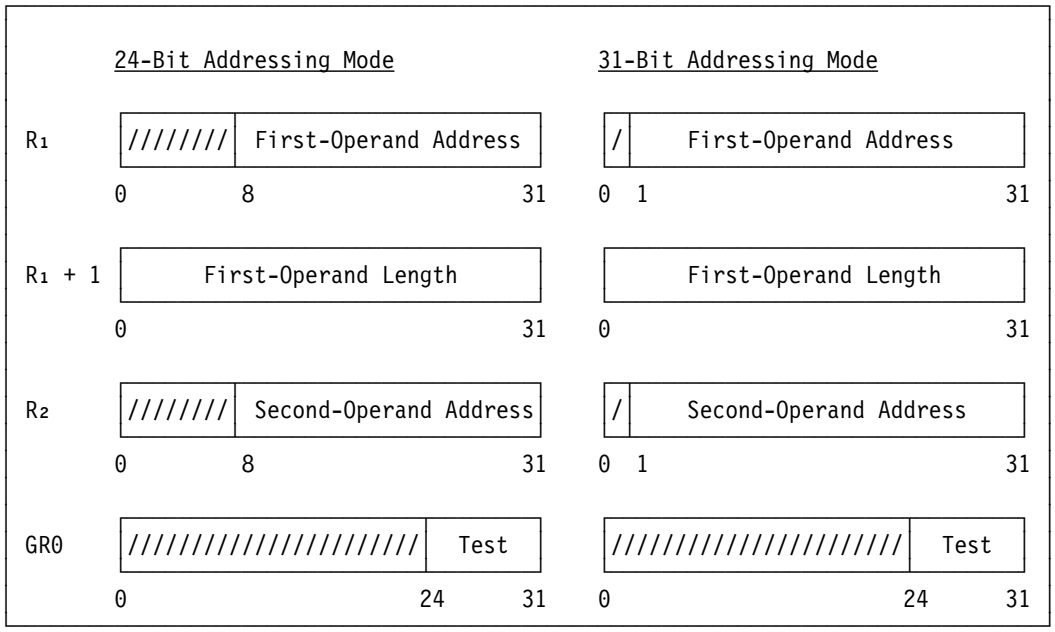



*Figure 7-71. Register Contents for TRANSLATE EXTENDED*

Access exceptions for the portion of the first operand to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

Access exceptions for all 256 bytes of the second operand may be recognized, even if not all bytes are used.

Access exceptions are not recognized if the $R_1$ field is odd. When the length of the first operand is zero, no access exceptions for the first operand are recognized.

***Resulting Condition Code:***

0 Entire first operand processed without finding a byte equal to the test byte
1 First-operand byte is equal to the test byte
2 --
3 CPU-determined number of bytes processed

***Program Exceptions:***

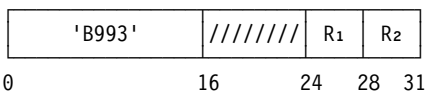
- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 1 is not installed)
- Specification

**Programming Notes:**

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the translation. The program need not determine the number of bytes that were translated.
2. The instruction can improve performance by being used in place of a TRANSLATE AND TEST instruction that locates an escape character, followed by a TRANSLATE instruction that translates the bytes preceding the escape character.
3. The storage-operand references of TRANSLATE EXTENDED may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

## TRANSLATE ONE TO ONE

```
TROO     R1,R2     [RRE]
```

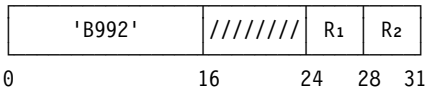





LzLabs GmbH. Ex. 1008-438

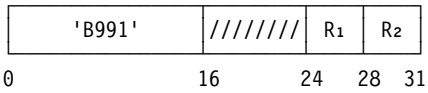
## TRANSLATE ONE TO TWO

```
TROT     R1,R2     [RRE]
```



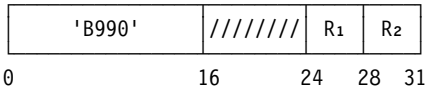

## TRANSLATE TWO TO ONE

```
TRTO     R1,R2     [RRE]
```




## TRANSLATE TWO TO TWO

```
TRTT     R1,R2     [RRE]
```




The characters of the second operand are used as arguments to select function characters from a translation table designated by the address in general register 1. Each function character selected from the translation table is compared to a test character in general register 0, and, unless an equal comparison occurs, is placed at the first-operand location. The operation proceeds until a selected function character equal to the test character is encountered, the end of the second operand is reached, or a CPU-determined number of characters have been processed, whichever occurs first. The result is indicated in the condition code.

The lengths of the operand and test characters are as follows:

- For TRANSLATE ONE TO ONE, the second-operand, first-operand, and test characters are single bytes.
- For TRANSLATE ONE TO TWO, the second-operand characters are single bytes, and the first-operand and test characters are double bytes.
- For TRANSLATE TWO TO ONE, the second-operand characters are double bytes, and the first-operand and test characters are single bytes.
- For TRANSLATE TWO TO TWO, the second-operand, first-operand, and test characters are double bytes.

For TRANSLATE ONE TO ONE and TRANSLATE TWO TO ONE, the test character is in bit positions 24-31 of general register 0. For TRANSLATE ONE TO TWO and TRANSLATE TWO TO TWO, the test character is in bit positions 16-31 of general register 0.

The $R_1$ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively. The number of bytes in the second-operand location is specified by the contents of general register $R_1 + 1$, and those contents are treated as a 32-bit unsigned binary integer. The length of the first-operand location is considered to be the same as that of the second operand for TRANSLATE ONE TO ONE and TRANSLATE TWO TO TWO, twice that for TRANSLATE ONE TO TWO, and one half that for TRANSLATE TWO TO ONE.

For TRANSLATE TWO TO ONE and TRANSLATE TWO TO TWO, the length in general register $R_1 + 1$ must be an even number of bytes; otherwise, a specification exception is recognized.

The translation table is treated as being on a doubleword boundary for TRANSLATE ONE TO ONE and TRANSLATE ONE TO TWO and on a 4K-byte boundary for TRANSLATE TWO TO ONE and TRANSLATE TWO TO TWO. The rightmost bits of the register that are not used to form the address, which are bits 29-31 in the doubleword case and bits 20-31 in the 4K-byte case, are ignored.

The handling of the addresses in general registers $R_1$, $R_2$, and 1 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 8-31 of general registers $R_1$ and $R_2$ and 8-28 or 8-19 of 1 constitute the address, and the contents of bit positions 0-7 are ignored. In the


LzLabs GmbH. Ex. 1008-439

31-bit addressing mode, the contents of bit positions 1-31 of registers $R_1$ and $R_2$ and 1-28 or 1-19 of 1 constitute the address, and the contents of bit position 0 are ignored.

The contents of the registers just described are shown in Figure 7-72.

In the access-register mode, the contents of access registers $R_1$, $R_2$, and 1 are used for accessing the first operand, second operand, and translation table, respectively.

The length of the translation table designated by the address contained in general register 1 is as follows:

- For TRANSLATE ONE TO ONE, the translation-table length is 256 bytes; each of the 256 function characters is a single byte.
- For TRANSLATE ONE TO TWO, the translation-table length is 512 bytes; each of the 256 function characters is a double byte.
- For TRANSLATE TWO TO ONE, the translation-table length is 65,536 (64K) bytes; each of the 64K function characters is a single byte.

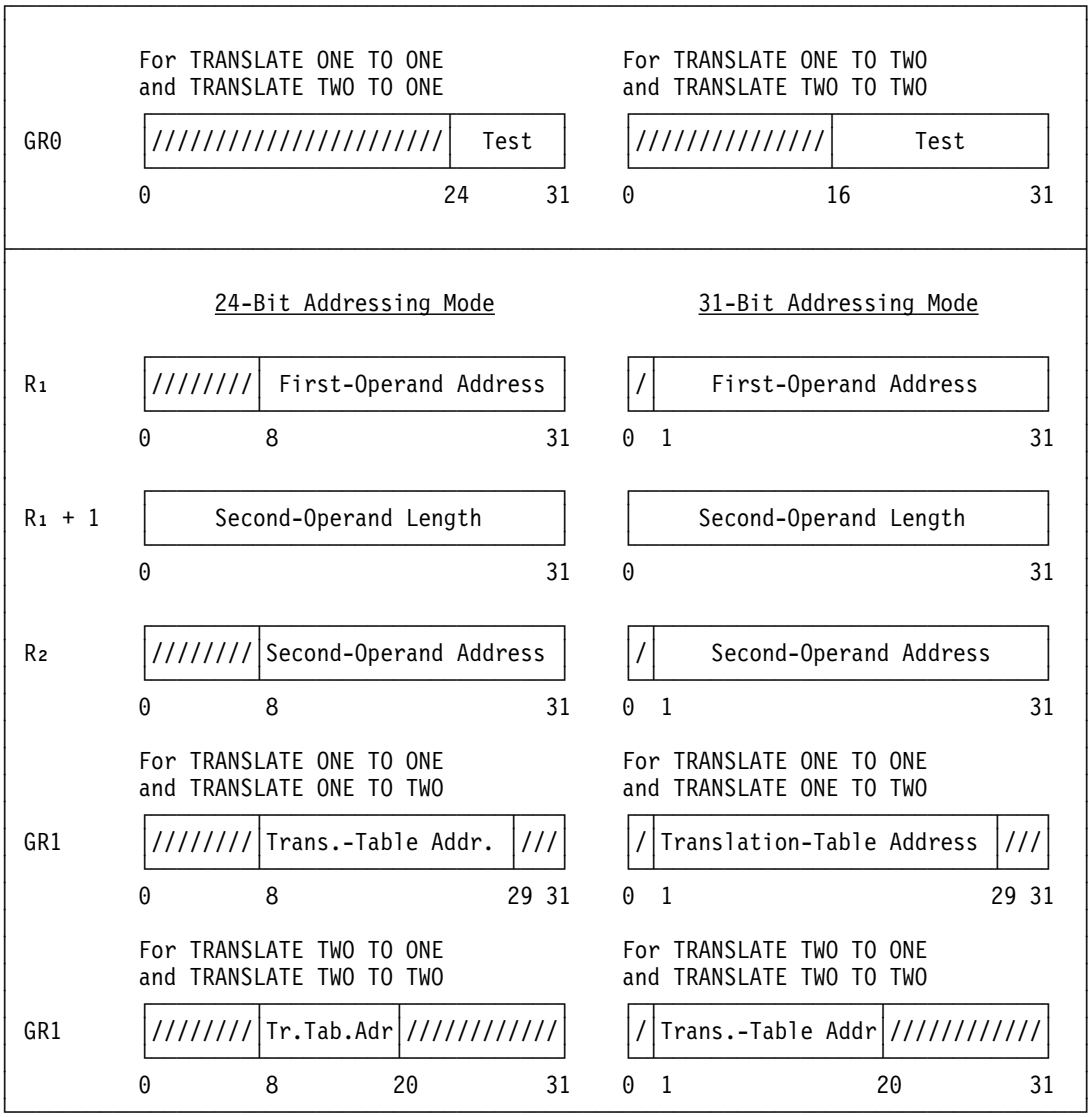



*Figure 7-72. Register Contents for TRANSLATE ONE TO ONE, TRANSLATE ONE TO TWO, TRANSLATE TWO TO ONE, and TRANSLATE TWO TO TWO*



LzLabs GmbH. Ex. 1008-440

- For TRANSLATE TWO TO TWO, the translation-table length is 131,072 (128K) bytes; each of the 64K function characters is a double byte.

The characters of the second operand are selected one by one for translation, proceeding left to right. Each argument character is added to the initial translation-table address. The addition is performed following the rules for address arithmetic, with the argument character treated as follows:

- For TRANSLATE ONE TO ONE, the argument character is treated as an eight-bit unsigned binary integer extended on the left with 24 zeros.
- For TRANSLATE ONE TO TWO, the argument character is treated as an eight-bit unsigned binary integer extended on the right with a zero and on the left with 23 zeros.
- For TRANSLATE TWO TO ONE, the argument character is treated as a 16-bit unsigned binary integer extended on the left with 16 zeros.
- For TRANSLATE TWO TO TWO, the argument character is treated as a 16-bit unsigned binary integer extended on the right with a zero and on the left with 15 zeros.

The rightmost bits of the translation-table address that are ignored (29-31 or 20-31) are treated as zeros during this addition.

The sum is used as the address of the function character.

Each function character selected as described above is first compared to the test character in general register 0. If the result is an equal comparison, the operation is completed. If the function character is not equal to the test character, the function character is placed in the next available character position in the first operand, that is, the first function character is placed at the beginning of the first-operand location, and each successive function character is placed immediately to the right of the preceding character. The second operand and the translation table are not altered unless an overlap occurs.

The operation proceeds until a selected function character equal to the test character is encountered, the second-operand location is exhausted, or a CPU-determined number of second-operand characters have been processed.

When a selected function character equal to the test character is encountered, condition code 1 is set. When the second-operand location is exhausted without finding a selected function character equal to the test character, condition code 0 is set. When a CPU-determined number of characters have been processed, condition code 3 is set. Condition code 3 may be set even when the next character to be processed results in a function character equal to the test character or when the second-operand location is exhausted. In these cases, condition code 1 or 0, respectively, will be set when the instruction is executed again.

If the operation is completed with condition code 0, the contents of general register $R_2$ are incremented by the contents of general register $R_1 + 1$, and the contents of general register $R_1$ are incremented as follows:

- For TRANSLATE ONE TO ONE and TRANSLATE TWO TO TWO, the same as for general register $R_2$.
- For TRANSLATE ONE TO TWO, by twice the amount for general register $R_2$.
- For TRANSLATE TWO TO ONE, by one half the amount for general register $R_2$.

The contents of general register $R_1 + 1$ are then set to zero.

If the operation is completed with condition code 1, the contents of general register $R_1 + 1$ are decremented by the number of second-operand bytes processed before the character that selected a function character equal to the test character was encountered, and the contents of general register $R_2$ are incremented by the same number, so that general register $R_2$ contains the address of the character that selected a function character equal to the test character. The contents of general register $R_1$ are incremented by the same, twice, or one half the number, as described above for condition code 0.

If the operation is completed with condition code 3, the contents of general register $R_1 + 1$ are decremented by the number of second-operand bytes processed, and the contents of general register $R_2$ are incremented by the same number, so


LzLabs GmbH. Ex. 1008-441

that the instruction, when reexecuted, contains the address of the next character to be processed. The contents of general register $R_1$ are incremented by the same, twice, or one half the number, as described above for condition code 0.

When general registers $R_1$ and $R_2$ are updated, the bits in them that are not part of the address may be set to zeros or may remain unchanged from their original values.

The contents of general registers 0 and 1 remain unchanged.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

During instruction execution, CPU retry may result in condition code 3 being set with possibly incorrect data having been stored in the first operand location at or to the right of the location designated by the final address in general register $R_1$. The amount of data stored depends on the operation and the point in time at which CPU retry occurred. In all cases, the storing will occur again, with correct data stored, when the instruction is executed again to continue processing the same operands.

When the $R_1$ register is the same register as the $R_2$ register, the $R_1$ or $R_2$ register is register 0, or the $R_2$ register is register 1, the results are unpredictable.

When any of the first and second operands and the translation table overlaps another of them, the results are unpredictable.

Access exceptions for the portion of the first or second operand to the right of the last character processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last character processed.

Access exceptions for all characters of the translation table may be recognized even if not all characters are used.

Access exceptions are not recognized if the $R_1$ field is odd. When the length of the second operand is zero, no access exceptions for the first or second operand are recognized, and access exceptions for the translation table may or may not be recognized.

***Resulting Condition Code:***

0 Entire second operand processed without finding a resulting function character equal to the test character
1 Second-operand character found resulting in a function character equal to the test character
2 --
3 CPU-determined number of characters processed

***Program Exceptions:***

- Access (fetch, operand 2 and translation table; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

**Programming Notes:**

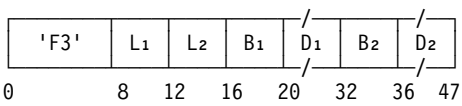
1. These instructions differ from the TRANSLATE EXTENDED instruction by having the following attributes:
   - Depending on the instruction used, the sets of argument characters and function characters each can contain single-byte or double-byte characters.
   - The test character is compared to a resulting function character instead of to an argument character.
   - The argument (source) and function (destination) operands are different operands.
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the translation. The program need not determine the number of characters that were translated.
3. The storage-operand references of these instructions may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)



LzLabs GmbH. Ex. 1008-442

## UNPACK

```
UNPK   D1(L1,B1),D2(L2,B2)      [SS]
```




The format of the second operand is changed from packed to zoned, and the result is placed at the first-operand location. The packed and zoned formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the packed format. Its digits and sign are placed unchanged in the first-operand location, using the zoned format. Zone bits with coding of 1111 are supplied for all bytes except the rightmost byte, the zone of which receives the sign of the second operand. The sign and digits are not checked for valid codes.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first-operand field is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if the operands were processed one byte at a time and as if the first result byte were stored immediately after fetching the first operand byte. The entire rightmost second-operand byte is used in forming the first result byte. For the remainder of the field, information for two result bytes is obtained from a single second-operand byte, and execution proceeds as if the leftmost four bits of the byte were to remain available for the next result byte and need not be refetched. Thus, the result is as if two result bytes were to be stored immediately after fetching a single operand byte.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

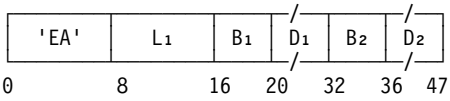
- Access (fetch, operand 2; store, operand 1)

**Programming Notes:**

1. An example of the use of the UNPACK instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. A field that is to be unpacked can be destroyed by improper overlapping. To save storage space for unpacking by overlapping the operands, the rightmost byte of the first operand must be to the right of the rightmost byte of the second operand by the number of bytes in the second operand minus 2. If only one or two bytes are to be unpacked, the rightmost bytes of the two operands may coincide.
3. The storage-operand references of UNPACK may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

## UNPACK ASCII

```
UNPKA    D1(L1,B1),D2(B2)    [SS]
```




The format of the second operand is changed from packed to ASCII, and the result is placed at the first-operand location. The packed format is described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the packed format. Its digits are converted to ASCII characters by extending them on the left with 0011 binary, and the ASCII characters are then placed at the first operand location. The digits are not checked for valid codes.

The sign of the second operand is not transferred to the first operand but is checked for validity and determines the condition code. If the sign is 1010, 1100, 1110 or 1111 binary (plus), condition code 0 is set. If the sign is 1011 or 1101 binary (minus), condition code 1 is set. If the sign is not one of the codes for plus or minus, condition code 3 is set.

The converted last digit is placed in the rightmost byte position of the result field, and the other con-


LzLabs GmbH. Ex. 1008-443

verted digits are placed adjacent to the last and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left.

The length of the second operand is 16 bytes. The second operand consists of 31 digits and a sign.

The length of the first operand is designated by the contents of the $L_1$ field. The first-operand length must not exceed 32 bytes ($L_1$ must be less than or equal to 31); otherwise, a specification exception is recognized.

If the first operand is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the length of the first operand is 32 bytes, the leftmost byte is set to ASCII zero, 30 hex.

The results are unpredictable if the first and second operands overlap in any way.

As observed by other CPUs and by channel programs, the first operand is not necessarily stored into in any particular order.

***Resulting Condition Code:***

0 Sign is plus
1 Sign is minus
2 --
3 Sign is invalid

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

**Programming Notes:**

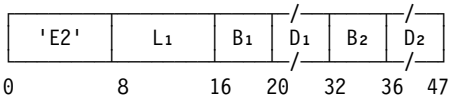
1. The following example illustrates the use of the instruction to unpack to ASCII digits:

```
ASDIGITS  DS     CL31
PKDIGITS  DS     0PL16
          DC     X'1234567890'
          DC     X'1234567890'
          DC     X'1234567890'
          DC     X'1C'
          ...
          UNPKA  ASDIGITS(31),PKDIGITS
```

2. The storage-operand references of UNPACK ASCII may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

## UNPACK UNICODE

```
UNPKU    D1(L1,B1),D2(B2)    [SS]
```




The format of the second operand is changed from packed to Unicode Basic Latin, and the result is placed at the first-operand location. The packed format is described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the packed format. Its digits are converted to two-byte Unicode characters by extending them on the left with 000000000011 binary (003 hex), and the Unicode characters are then placed at the first operand location. The digits are not checked for valid codes. The sign of the second operand is not transferred to the first operand but is checked for validity and determines the condition code. If the sign is 1010, 1100, 1110 or 1111 binary (plus), condition code 0 is set. If the sign is 1011 or 1101 binary (minus), condition code 1 is set. If the sign is not one of the codes for plus or minus, condition code 3 is set.

The converted last digit is placed in the rightmost character position of the result field, and the other converted digits are placed adjacent to the last and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left.

The length of the second operand is 16 bytes; the second operand consists of 31 digits and a sign.



LzLabs GmbH. Ex. 1008-444

The length of the first operand is designated by the contents of the $L_1$ field. The first-operand length must not exceed 32 characters or 64 bytes ($L_1$ must be less than or equal to 63 and must be odd); otherwise a specification exception is recognized.

If the first operand is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the length of the first operand is 32 characters, the leftmost character is set to Unicode Basic Latin zero, 0030 hex.

The results are unpredictable if the first and second operands overlap in any way.

As observed by other CPUs and by channel programs, the first operand is not necessarily stored into in any particular order.

***Resulting Condition Code:***

0 Sign is plus
1 Sign is minus
2 --
3 Sign is invalid

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

**Programming Notes:**

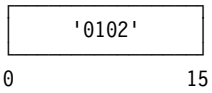
1. The following example illustrates the use of the instruction to unpack to European numbers:

```
UNDIGITS  DS     CL62
PKDIGITS  DS     0PL16
          DC     X'1234567890'
          DC     X'1234567890'
          DC     X'1234567890'
          DC     X'1C'
          ...
          UNPKU  UNDIGITS(62),PKDIGITS
```

2. The storage-operand references of UNPACK UNICODE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)

# UPDATE TREE

```
UPT       [E]
```




The doubleword nodes of a tree in storage are examined successively on a path toward the base of the tree, and the contents of general-register pair 0-1 are conditionally interchanged with the contents of the nodes so as to give a unique maximum logical value in general register 0.

General register 4 contains the base address of the tree, and general register 5 contains the index of a node whose parent node will be examined first. The base address is eight less than the address of the root node of the tree. The initial contents of general registers 4 and 5 must be a multiple of 8; otherwise, a specification exception is recognized.

In the access-register mode, access register 4 specifies the address space containing the tree.

This instruction may be interrupted between units of operation. The condition code is unpredictable if the instruction is interrupted.

A unit of operation begins by shifting the contents of general register 5 right logically one position and then setting bit 29 to zero. However, general register 5 remains unchanged if the execution of a unit of operation is nullified or suppressed. If after shifting and setting bit 29 to zero, the contents of general register 5 are zero, the instruction is completed, and condition code 1 is set; otherwise, the unit of operation continues.

Bit 0 of general register 0 is tested. If bit 0 of register 0 is one, the instruction is completed, and condition code 3 is set.

If bit 0 of general register 0 is zero, the sum of the contents of general registers 4 and 5 is used as the intermediate value for normal operand address generation. The generated address is the address of a node in storage.

The contents of general register 0 are logically compared with the contents of the first word of the currently addressed node. If the register operand


LzLabs GmbH. Ex. 1008-445

is low, the contents of general-register pair 0-1 are interchanged with those of the node, and a unit of operation is completed. If the register operand is high, no additional action is taken, and the unit of operation is completed. If the compare values are equal, general-register pair 2-3 is loaded from the currently addressed node, the instruction is completed, and condition code 0 is set.

In those cases when the value in the first word of the node is less than or equal to the value in the register, the contents of the node remain unchanged. However, in some models, these contents may be fetched and subsequently stored back.

Access exceptions are recognized only for one doubleword node at a time. Access exceptions, change-bit action, and PER storage alteration do not occur for subsequent nodes until the previous node has been successfully compared and updated, and they also do not occur if a specification-exception condition exists.

***Resulting Condition Code:***

0 Equal compare values at currently addressed node
1 No equal compare values found on path, or no comparison made
2 --
3 General register 5 nonzero and general register 0 negative

***Program Exceptions:***

- Access (fetch and store, nodes of tree)
- Specification

**Programming Notes:**

1. An example of the use of UPDATE TREE is given in "Sorting Instructions" in Appendix A, "Number Representation and Instruction-Use Examples."
2. For use in sorting, when equal compare values have been found, the contents of general registers 1 and 3 can be appropriate (depending on the contents of the tree) for the subsequent execution of COMPARE AND FORM CODEWORD. The contents of general register 2, shifted right 16 bit positions, can be similarly appropriate, and they can provide for minimal recomparison of partially equal keys. Refer to "Sorting Instructions" on page A-51 for a discussion of trees and their use in sorting.
3. The program should avoid placing a nonzero value in bit positions 0-6 of general register 5 when in the 24-bit addressing mode. If any bit in bit positions 0-6 is a one, the nodes of the tree will not be examined successively.
4. When general register 0 is negative, and provided that the tree has been updated properly previously, the node represented by the general-register pair 0-1 either is the node or is equal to the node (equal keys) that would be selected if the unit of operation continued. In this case, ending the unit of operation and setting condition code 3 is a faster means of selecting an appropriate node because it does not require further examination and updating of the tree.
5. Setting condition code 3 provides improved performance when the replacement record is equal to the old winner and, more importantly (since the first case can be detected by means of the condition code of CFC), when the update path contains a negative codeword, indicating equality with the old winner.
6. In those cases when the value in the first word of the node is less than or equal to the value in the register, depending on the model, the contents of the node may be fetched and subsequently stored back. As a result, any of the following may occur for the storage location containing the node: a PER storage-alteration event may be recognized; a protection exception for storing may be recognized; and, provided no access exceptions exist, the change bit may be set to one. Because the contents of storage remain unchanged, the change bit may or may not be one when a PER storage-alteration event is recognized.
7. Special precautions should be taken when UPDATE TREE is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.
8. Further programming notes concerning interruptible instructions are included in "Interruptible Instructions" on page 5-17.



LzLabs GmbH. Ex. 1008-446

9. The storage-operand references for UPDATE TREE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)

10. Figure 7-73 is a summary of the operation of UPDATE TREE.

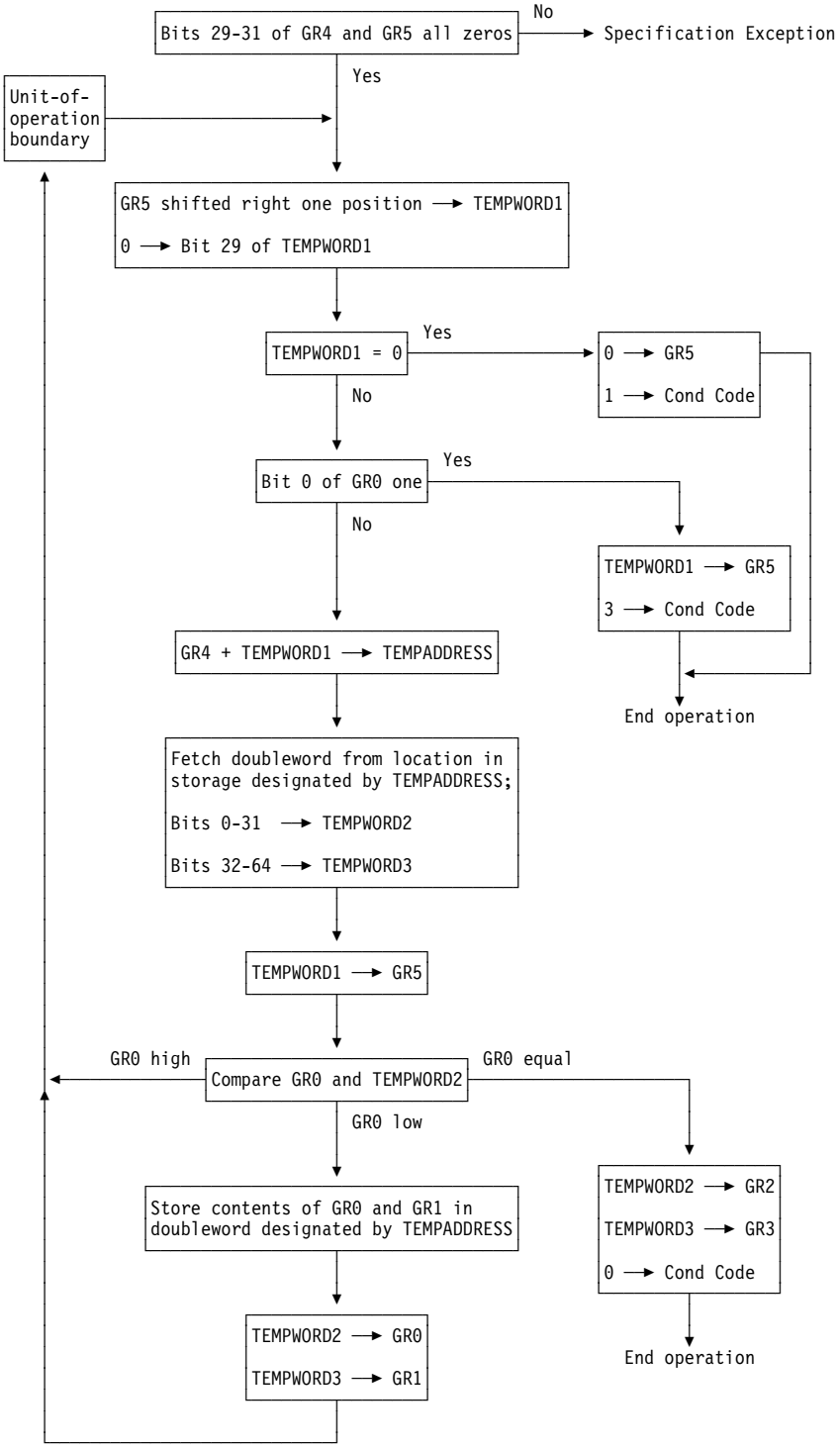



*Figure 7-73. Execution of UPDATE TREE*



LzLabs GmbH. Ex. 1008-447

LzLabs GmbH. Ex. 1008-448

# Chapter 8. Decimal Instructions

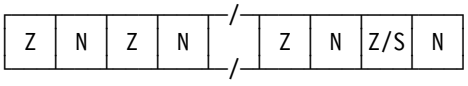

Decimal-Number Formats . . . . . 8-1
Zoned Format . . . . . 8-1
Packed Format . . . . . 8-1
Decimal Codes . . . . . 8-2
Decimal Operations . . . . . 8-2
Decimal-Arithmetic Instructions . . . . . 8-2
Editing Instructions . . . . . 8-3
Execution of Decimal Instructions . . . . . 8-3
Other Instructions for Decimal Operands . 8-3
Decimal-Operand Data Exception . . . . . 8-4
Instructions . . . . . 8-4
ADD DECIMAL . . . . . 8-6
COMPARE DECIMAL . . . . . 8-6
DIVIDE DECIMAL . . . . . 8-7
EDIT . . . . . 8-7
EDIT AND MARK . . . . . 8-12
MULTIPLY DECIMAL . . . . . 8-12
SHIFT AND ROUND DECIMAL . . . . . 8-13
SUBTRACT DECIMAL . . . . . 8-14
TEST DECIMAL . . . . . 8-14
ZERO AND ADD . . . . . 8-14


The decimal instructions of this chapter perform arithmetic and editing operations on decimal data. Additional operations on decimal data are provided by several of the instructions in Chapter 7, "General Instructions." Decimal operands always reside in storage, and most decimal instructions use the SS instruction format. Decimal operands occupy storage fields that can start on any byte boundary.

## Decimal-Number Formats

Decimal numbers may be represented in either the zoned or packed format. Both decimal-number formats are of variable length; the instructions used to operate on decimal data each specify the length of their operands and results. Each byte of either format consists of a pair of four-bit codes; the four-bit codes include decimal-digit codes, sign codes, and a zone code.

## Zoned Format

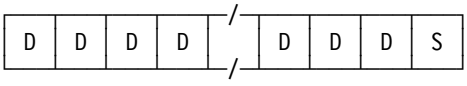



In the zoned format, the rightmost four bits of a byte are called the numeric bits (N) and normally consist of a code representing a decimal digit. The leftmost four bits of a byte are called the zone bits (Z), except for the rightmost byte of a decimal operand, where these bits may be treated either as a zone or as a sign (S).

Decimal digits in the zoned format may be part of a larger character set, which includes also alphabetic and special characters. The zoned format is, therefore, suitable for input, editing, and output of numeric data in human-readable form. There are no decimal-arithmetic instructions which operate directly on decimal numbers in the zoned format; such numbers must first be converted to the packed format.

The editing instructions produce a result of up to 256 bytes; each byte may be a decimal digit in the zoned format, a message byte, or a fill byte.

## Packed Format




In the packed format, each byte contains two decimal digits (D), except for the rightmost byte, which contains a sign to the right of a decimal digit. Decimal arithmetic is performed with operands in the packed format and generates results in the packed format.

The packed-format operands and results of decimal-arithmetic instructions may be up to 16 bytes (31 digits and sign), except that the maximum length of a multiplier or divisor is eight bytes (15 digits and sign). In division, the sum of the lengths of the quotient and remainder may be from two to 16 bytes. The editing instructions can fetch as many as 256 decimal digits from one or more decimal numbers of variable length, each in the packed format.

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-449

## Decimal Codes

The decimal digits 0-9 have the binary encoding 0000-1001.

The preferred sign codes are 1100 for plus and 1101 for minus. These are the sign codes generated for the results of the decimal-arithmetic instructions and the CONVERT TO DECIMAL instruction.

Alternate sign codes are also recognized as valid in the sign position: 1010, 1110, and 1111 are alternate codes for plus, and 1011 is an alternate code for minus. Alternate sign codes are accepted for any decimal source operand, but are not generated in the completed result of a decimal-arithmetic instruction or CONVERT TO DECIMAL. This is true even when an operand remains otherwise unchanged, such as when adding zero to a number. An alternate sign code is, however, left unchanged by MOVE NUMERICS, MOVE WITH OFFSET, MOVE ZONES, PACK, and UNPACK.

When an invalid sign or digit code is detected, a data exception is recognized. For the decimal-arithmetic instructions and CONVERT TO BINARY, the action taken for a data exception depends on whether a sign code is invalid. When a sign code is invalid, the operation is suppressed regardless of whether any other condition causing a data exception exists. When an invalid digit code is detected but no sign code is invalid, the operation is terminated on some models and suppressed on others.

For the editing instructions EDIT and EDIT AND MARK, an invalid sign code is not recognized. The operation is terminated for a data exception due to an invalid digit code. No validity checking is performed by MOVE NUMERICS, MOVE WITH OFFSET, MOVE ZONES, PACK, and UNPACK.

The zone code 1111 is generated in the left four bit positions of each byte representing a zone and a decimal digit in zoned-format results. Zoned-format results are produced by EDIT, EDIT AND MARK, and UNPACK. For EDIT and EDIT AND MARK, each result byte representing a zoned-format decimal digit contains the zone code 1111 in the left four bit positions and the decimal-digit code in the right four bit positions. For UNPACK, zone bits with a coding of 1111 are supplied for all bytes except the rightmost byte, the zone of which receives the sign.

The meaning of the decimal codes is summarized in Figure 8-1.

| Code (Binary) | Recognized As | |
|---|---|---|
| | Digit | Sign |
| 0000 | 0 | Invalid |
| 0001 | 1 | Invalid |
| 0010 | 2 | Invalid |
| 0011 | 3 | Invalid |
| 0100 | 4 | Invalid |
| 0101 | 5 | Invalid |
| 0110 | 6 | Invalid |
| 0111 | 7 | Invalid |
| 1000 | 8 | Invalid |
| 1001 | 9 | Invalid |
| 1010 | Invalid | Plus |
| 1011 | Invalid | Minus |
| 1100 | Invalid | Plus (preferred) |
| 1101 | Invalid | Minus (preferred) |
| 1110 | Invalid | Plus |
| 1111 | Invalid | Plus (zone) |

*Figure 8-1. Summary of Digit and Sign Codes*

**Programming Note:** Since 1111 is both the zone code and an alternate code for plus, unsigned (positive) decimal numbers may be represented in the zoned format with 1111 zone codes in all byte positions. The result of the PACK instruction converting such a number to the packed format may be used directly as an operand for decimal instructions.

## Decimal Operations

The decimal instructions in this chapter consist of two classes, the decimal-arithmetic instructions and the editing instructions.

## Decimal-Arithmetic Instructions

The decimal-arithmetic instructions perform addition, subtraction, multiplication, division, comparison, and shifting.

Operands of the decimal-arithmetic instructions are in the packed format and are treated as signed decimal integers. A decimal integer is represented in true form as an absolute value with a separate plus or minus sign. It contains an odd



LzLabs GmbH. Ex. 1008-450

number of decimal digits, from one to 31, and the sign; this corresponds to an operand length of one to 16 bytes.

A decimal zero normally has a plus sign, but multiplication, division, and overflow may produce a zero value with a minus sign. Such a negative zero is a valid operand and is treated as equal to a positive zero by COMPARE DECIMAL.

The lengths of the two operands specified in the instruction need not be the same. If necessary, the shorter operand is considered to be extended with zeros on the left. Results, however, cannot exceed the first-operand length as specified in the instruction.

When a carry or leftmost nonzero digits of the result are lost because the first-operand field is too short, the result is obtained by ignoring the overflow digits, condition code 3 is set, and, if the decimal-overflow mask bit is one, a program interruption for decimal overflow occurs. The operand lengths alone are not an indication of overflow; nonzero digits must have been lost during the operation.

The operands of decimal-arithmetic instructions should not overlap at all or should have coincident rightmost bytes. In ZERO AND ADD, the operands may also overlap in such a manner that the rightmost byte of the first operand (which becomes the result) is to the right of the rightmost byte of the second operand. For these cases of proper overlap, the result is obtained as if operands were processed right to left. Because the codes for digits and signs are verified during the performance of the arithmetic, improperly overlapping operands are recognized as data exceptions. However, in ZERO AND ADD when the rightmost byte of the first operand is to the left of the rightmost byte of the second operand, the entire second operand may be fetched, depending on the model, before any storing occurs, which will cause a data exception not to be recognized. See "Interlocks within a Single Instruction" on page 5-80 for how overlap is detected in the access-register mode.

**Programming Note:** A packed decimal number in storage may be designated as both the first and second operand of ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, SUBTRACT DECIMAL, or ZERO AND ADD. Thus, a decimal number may be added to itself, compared with itself, and so forth; SUBTRACT DECIMAL may be used to set a decimal field in storage to zero; and, for MULTIPLY DECIMAL, a decimal number may be squared in place. In these cases, the lengths of the two operands are not necessarily equal and may, depending on the instruction, be prohibited from being equal.

## Editing Instructions

The editing instructions are EDIT and EDIT AND MARK. For these instructions, only the first operand (the pattern) has an explicitly specified length. The second operand (the source) is considered to have as many digits as necessary for the completion of the operation.

Overlapping operands for the editing instructions yield unpredictable results.

## Execution of Decimal Instructions

During the execution of a decimal instruction, all bytes of the operands are not necessarily accessed concurrently, and the fetch and store accesses to a single location do not necessarily occur one immediately after the other. Furthermore, for decimal instructions, data in source fields may be accessed more than once, and intermediate values may be placed in the result field that may differ from the original operand and final result values. (See "Storage-Operand Consistency" on page 5-87.) Thus, in a multiprocessing configuration, an instruction such as ADD DECIMAL cannot be safely used to update a shared storage location when the possibility exists that another CPU may also be updating that location.

## Other Instructions for Decimal Operands

In addition to the decimal instructions in this chapter, MOVE NUMERICS and MOVE ZONES are provided for operating on data of lengths up to 256 bytes in the zoned format. Two instructions are provided for converting data between the zoned and packed formats: PACK transforms zoned data of lengths up to 16 bytes into packed data, and UNPACK performs the reverse transformation. MOVE WITH OFFSET can operate on


LzLabs GmbH. Ex. 1008-451

packed data of lengths up to 16 bytes. Two instructions are provided for conversion between the packed-decimal and signed-binary-integer formats. CONVERT TO BINARY converts packed decimal to binary, and CONVERT TO DECIMAL converts binary to packed decimal; the length of the packed decimal operand of these instructions is eight bytes (15 digits and sign). These seven instructions are not considered to be decimal instructions and are described in Chapter 7, "General Instructions." The editing instructions in this chapter may also be used to change data from the packed to the zoned format.

## Decimal-Operand Data Exception

A decimal-operand data exception is recognized when any of the following is true:

1. The sign or digit codes of operands in the decimal instructions or in CONVERT TO BINARY (described in Chapter 7, "General Instructions") are invalid.
2. The operand fields in ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, and SUBTRACT DECIMAL overlap in a way other than with coincident rightmost bytes; or operand fields in ZERO AND ADD overlap, and the rightmost byte of the second operand is to the right of the rightmost byte of the first operand. On some models, the improper overlap of operands for ZERO AND ADD is not recognized as a decimal-operand data exception; instead, the operation is performed as if the entire second operand were fetched before any byte of the result is stored.
3. The multiplicand in MULTIPLY DECIMAL has an insufficient number of leftmost zeros.

The action taken for a decimal-operand data exception depends on whether a sign code is invalid. The operation is suppressed when a sign code is invalid, regardless of whether any other condition causing the exception exists; when no sign code is invalid, the operation is terminated on some models and suppressed on others.

For all instructions other than EDIT and EDIT AND MARK, when the operation is terminated, the contents of the sign position in the rightmost byte of the result field either remain unchanged or are set to the preferred sign code; the contents of the remainder of the result field are unpredictable.

In the case of EDIT and EDIT AND MARK, an invalid sign code cannot occur; the operation is terminated on a decimal-operand data exception for an invalid digit code.

**Programming Notes:**

1. The definition for decimal-operand data exception permits termination when digit codes are invalid but no sign code is invalid. On some models, valid digit codes may be placed in the result field even if the original contents were invalid. Thus it is possible, after a decimal-operand data exception occurs, for all fields to contain valid codes.
2. An invalid sign code for the rightmost byte of the result field is not generated when the operation is terminated. However, an invalid second-operand sign code is not necessarily preserved when it is located in the numeric portion of the result field.
3. When, after a program interruption for decimal-operand data exception, a sign code is found to be invalid, the operation has been suppressed if both of the following conditions are met:
   a. The invalid sign of the source field is not located in the numeric portion of the result field.
   b. The invalid sign code is in a position specified by the instruction to be checked for a valid sign. (This condition excludes the first operand of ZERO AND ADD and both operands of EDIT and EDIT AND MARK.)

## Instructions

The decimal instructions and their mnemonics, formats, and operation codes are listed in Figure 8-2 on page 8-5. The figure also indicates when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

**Note:** In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For ADD DECIMAL, for example, AP is the mnemonic and $D_1(L_1,B_1),D_2(L_2,B_2)$ the operand designation.



LzLabs GmbH. Ex. 1008-452

**Programming Note:** The decimal instruction TEST DECIMAL is available when the extended-translation facility 2 is installed.

| Name | Mne-monic | Characteristics | | | | | Op Code |
|---|---|---|---|---|---|---|---|
| ADD DECIMAL | AP | SS C | A | Dd DF | ST | $B_1$ $B_2$ | FA |
| COMPARE DECIMAL | CP | SS C | A | Dd | | $B_1$ $B_2$ | F9 |
| DIVIDE DECIMAL | DP | SS | A SP | Dd DK | ST | $B_1$ $B_2$ | FD |
| EDIT | ED | SS C | A | Dd | ST | $B_1$ $B_2$ | DE |
| EDIT AND MARK | EDMK | SS C | A | Dd G1 | R ST | $B_1$ $B_2$ | DF |
| MULTIPLY DECIMAL | MP | SS | A SP | Dd | ST | $B_1$ $B_2$ | FC |
| SHIFT AND ROUND DECIMAL | SRP | SS C | A | Dd DF | ST | $B_1$ | F0 |
| SUBTRACT DECIMAL | SP | SS C | A | Dd DF | ST | $B_1$ $B_2$ | FB |
| TEST DECIMAL | TP | RSL C E2 | A | | | $B_1$ | EBC0 |
| ZERO AND ADD | ZAP | SS C | A | Dd DF | ST | $B_1$ $B_2$ | F8 |

**Explanation:**

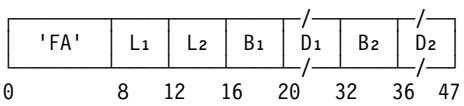
A Access exceptions for logical addresses.
$B_1$ $B_1$ field designates an access register in the access-register mode.
$B_2$ $B_2$ field designates an access register in the access-register mode.
C Condition code is set.
Dd Decimal-operand data exception.
DF Decimal-overflow exception.
DK Decimal-divide exception.
E2 Extended-translation facility 2.
G1 Instruction execution includes the implied use of general register 1.
R PER general-register-alteration event.
RSL RSL instruction format.
SP Specification exception.
SS SS instruction format.
ST PER storage-alteration event.

*Figure 8-2. Summary of Decimal Instructions*


LzLabs GmbH. Ex. 1008-453

## ADD DECIMAL

```
AP     D1(L1,B1),D2(L2,B2)     [SS]
```




The second operand is added to the first operand, and the resulting sum is placed at the first-operand location. The operands and result are in the packed format.

Addition is algebraic, taking into account the signs and all digits of both operands. All sign and digit codes are checked for validity.

If the first operand is too short to contain all left-most nonzero digits of the sum, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs.

The sign of the sum is determined by the rules of algebra. In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, a zero result is given either a positive or negative sign, as determined by what the sign of the correct sum would have been.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

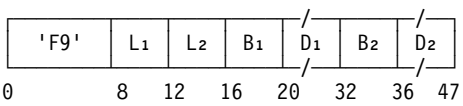
***Program Exceptions:***

- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Decimal overflow

**Programming Note:** An example of the use of the ADD DECIMAL instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

## COMPARE DECIMAL

```
CP     D1(L1,B1),D2(L2,B2)     [SS]
```




The first operand is compared with the second operand, and the result is indicated in the condition code. The operands are in the packed format.

Comparison is algebraic and follows the procedure for decimal subtraction, except that both operands remain unchanged. When the difference is zero, the operands are equal. When a nonzero difference is positive or negative, the first operand is high or low, respectively.

Overflow cannot occur because the difference is discarded.

All sign and digit codes are checked for validity.

***Resulting Condition Code:***

0 Operands equal
1 First operand low
2 First operand high
3 --

***Program Exceptions:***

- Access (fetch, operands 1 and 2)
- Data

**Programming Notes:**

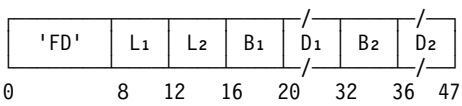
1. An example of the use of the COMPARE DECIMAL instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The preferred and alternate sign codes for a particular sign are treated as equivalent for comparison purposes.
3. A negative zero and a positive zero compare equal.



LzLabs GmbH. Ex. 1008-454

## DIVIDE DECIMAL

DP D1(L1,B1),D2(L2,B2) [SS]




The first operand (the dividend) is divided by the second operand (the divisor). The resulting quotient and remainder are placed at the first-operand location. The operands and results are in the packed format.

The quotient is placed leftmost in the first-operand location. The number of bytes in the quotient field is equal to the difference between the dividend and divisor lengths ($L_1$ - $L_2$). The remainder is placed rightmost in the first-operand location and has a length equal to the divisor length. Together, the quotient and remainder fields occupy the entire first operand; therefore, the address of the quotient is the address of the first operand.

The divisor length cannot exceed 15 digits and sign ($L_2$ not greater than seven) and must be less than the dividend length ($L_2$ less than $L_1$); otherwise, a specification exception is recognized.

The dividend, divisor, quotient, and remainder are each signed decimal integers in the packed format and are right-aligned in their fields. All sign and digit codes of the dividend and divisor are checked for validity.

The sign of the quotient is determined by the rules of algebra from the dividend and divisor signs. The sign of the remainder has the same value as the dividend sign. These rules hold even when the quotient or remainder is zero.

Overflow cannot occur. If the divisor is zero or the quotient is too large to be represented by the number of digits specified, a decimal-divide exception is recognized. This includes the case of division of zero by zero. The decimal-divide exception is indicated only if the sign codes of both the dividend and divisor are valid, and only if the digit or digits used in establishing the exception are valid.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

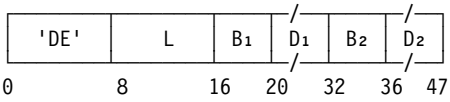
- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Decimal divide
- Specification

**Programming Notes:**

1. An example of the use of the DIVIDE DECIMAL instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. The dividend cannot exceed 31 digits and sign. Since the remainder cannot be shorter than one digit and sign, the quotient cannot exceed 29 digits and sign.
3. The condition for a decimal-divide exception can be determined by a trial comparison. The leftmost digit of the divisor is aligned one digit to the right of the leftmost dividend digit, with rightmost zeros appended up to the length of the dividend. When the divisor, so aligned, is less than or equal to the dividend, ignoring signs, a divide exception is indicated.
4. If a data exception does not exist, a decimal-divide exception occurs when the leftmost dividend digit is not zero.

## EDIT

ED D1(L,B1),D2(B2) [SS]




The second operand (the source), which normally contains one or more decimal numbers in the packed format, is changed to the zoned format and modified under the control of the first operand (the pattern). The edited result replaces the first operand.

The length field specifies the length of the first operand, which may contain bytes of any value.

The length of the source is determined by the operation according to the contents of the pattern. The source normally consists of one or more decimal numbers, each in the packed format. The leftmost four bits of each source byte must specify


LzLabs GmbH. Ex. 1008-455

a decimal-digit code (0000-1001); a sign code (1010-1111) is recognized as a data exception. The rightmost four bits may specify either a sign code or a decimal-digit code. Access and data exceptions are recognized only for those bytes in the second operand which are actually required.

The result is obtained as if both operands were processed left to right one byte at a time. Overlapping pattern and source fields give unpredictable results.

During the editing process, each byte of the pattern is affected in one of three ways:

1. It is left unchanged.
2. It is replaced by a source digit expanded to the zoned format.
3. It is replaced by the first byte in the pattern, called the fill byte.

Which of the three actions takes place is determined by one or more of the following: the type of the pattern byte, the state of the significance indicator, and whether the source digit examined is zero.

***Pattern Bytes:*** There are four types of pattern bytes: digit selector, significance starter, field separator, and message byte. Their coding is as follows:

| Name | Code (Binary) |
|---|---|
| Digit selector | 0010 0000 |
| Significance starter | 0010 0001 |
| Field separator | 0010 0010 |
| Message byte | Any other |

The detection of either a digit selector or a significance starter in the pattern causes an examination to be made of the significance indicator and of a source digit. As a result, either the expanded source digit or the fill byte, as appropriate, is selected to replace the pattern byte. Additionally, encountering a digit selector or a significance starter may cause the significance indicator to be changed.

The field separator identifies individual fields in a multiple-field editing operation. It is always replaced in the result by the fill byte, and the significance indicator is always off after the field separator is encountered.

Message bytes in the pattern are either replaced by the fill byte or remain unchanged in the result, depending on the state of the significance indicator. They may thus be used for padding, punctuation, or text in the significant portion of a field or for the insertion of sign-dependent symbols.

***Fill Byte:*** The first byte of the pattern is used as the fill byte. The fill byte can have any code and may concurrently specify a control function. If this byte is a digit selector or significance starter, the indicated editing action is taken after the code has been assigned to the fill byte.

***Source Digits:*** Each time a digit selector or significance starter is encountered in the pattern, a new source digit is examined for placement in the pattern field. Either the source digit is disregarded, or it is expanded to the zoned format, by appending the zone code 1111 on the left, and stored in place of the pattern byte.

Execution is as if the source digits were selected one byte at a time and as if a source byte were fetched for inspection only once during an editing operation. Each source digit is examined only once for a zero value. The leftmost four bits of each byte are examined first, and the rightmost four bits, when they represent a decimal-digit code, remain available for the next pattern byte that calls for a digit examination. When the leftmost four bits contain an invalid digit code, a data exception is recognized, and the operation is terminated.

At the time the left digit of a source byte is examined, the rightmost four bits are checked for the existence of a sign code. When a sign code is encountered in the rightmost four bit positions, these bits are not treated as a decimal-digit code, and a new source byte is fetched from storage when the next pattern byte calls for a source-digit examination.

When the pattern contains no digit selector or significance starter, no source bytes are fetched and examined.

***Significance Indicator:*** The significance indicator is turned on or off to indicate the significance or nonsignificance, respectively, of subsequent source digits or message bytes. Significant source digits replace their corresponding digit



LzLabs GmbH. Ex. 1008-456

selectors or significance starters in the result. Significant message bytes remain unchanged in the result.

The significance indicator, by its on or off state, indicates also the negative or positive value, respectively, of a completed source field and is used as one factor in the setting of the condition code.

The significance indicator is set to off at the start of the editing operation, after a field separator is encountered, or after a source byte is examined that has a plus code in the rightmost four bit positions.

The significance indicator is set to on when a significance starter is encountered whose source digit is a valid decimal digit, or when a digit selector is encountered whose source digit is a nonzero decimal digit, provided that in both instances the source byte does not have a plus code in the rightmost four bit positions.

In all other situations, the significance indicator is not changed. A minus sign code has no effect on the significance indicator.

***Result Bytes:*** The result of an editing operation replaces and is equal in length to the pattern. It is composed of pattern bytes, fill bytes, and zoned source digits.

If the pattern byte is a message byte and the significance indicator is on, the message byte remains unchanged in the result. If the pattern byte is a field separator or if the significance indicator is off when a message byte is encountered in the pattern, the fill byte replaces the pattern byte in the result.

If the digit selector or significance starter is encountered in the pattern with the significance indicator off and the source digit zero, the source digit is considered nonsignificant, and the fill byte replaces the pattern byte. If the digit selector or significance starter is encountered with either the significance indicator on or with a nonzero decimal source digit, the source digit is considered significant, is changed to the zoned format, and replaces the pattern byte in the result.

***Condition Code:*** The sign and magnitude of the last field edited are used to set the condition code. The term “last field” refers to those source digits, if any, in the second operand selected by digit selectors or significance starters after the last field separator; if the pattern contains no field separator, there is only one field, which is considered to be the last field. If no such source digits are selected, the last field is considered to be of zero length.

Condition code 0 is set when the last field edited is zero or of zero length.

Condition code 1 is set when the last field edited is nonzero and the significance indicator is on. (This indicates a result less than zero if the last source byte examined contained a sign code in the rightmost four bits.)

Condition code 2 is set when the last field edited is nonzero and the significance indicator is off. (This indicates a result greater than zero if the last source byte examined contained a sign code in the rightmost four bits.)

Figure 8-3 on page 8-11 summarizes the functions of the EDIT and EDIT AND MARK operations. The leftmost four columns list all the significant combinations of the four conditions that can be encountered in the execution of an editing operation. The rightmost two columns list the action taken for each case — the type of byte placed in the result field and the new setting of the significance indicator.

***Resulting Condition Code:***

0 Last field zero or zero length
1 Last field less than zero
2 Last field greater than zero
3 --

***Program Exceptions:***

- Access (fetch, operand 2; fetch and store, operand 1)
- Data


LzLabs GmbH. Ex. 1008-457

**Programming Notes:**

1. Examples of the use of the EDIT instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. Editing includes sign and punctuation control, and the suppression and protection of leading zeros by replacing them with blanks or asterisks. It also facilitates programmed blanking of all-zero fields. Several fields may be edited in one operation, and numeric information may be combined with text.
3. In most cases, the source is shorter than the pattern because each four-bit source digit produces an eight-bit byte in the result.
4. The total number of digit selectors and significance starters in the pattern always equals the number of source digits edited.
5. If the fill byte is a blank, if no significance starter exists in the pattern, and if the source digit examined for each digit selector is zero, the editing operation blanks the result field.
6. The resulting condition code indicates whether or not the last field is all zeros and, if nonzero, reflects the state of the significance indicator. The significance indicator reflects the sign of the source field only if the last source byte examined contains a sign code in the rightmost four bits. For multiple-field editing operations, the condition code reflects the sign and value only of the field following the last field separator.
7. Significant performance degradation is possible when, with DAT on, the second-operand address of an EDIT instruction designates a location that is closer to the left of a 4K-byte boundary than the length of the first operand of that instruction. This is because the machine may perform a trial execution of the instruction to determine if the second operand actually crosses the boundary. The second operand of EDIT, while normally shorter than the first operand, can in the extreme case have the same length as the first.



LzLabs GmbH. Ex. 1008-458

| Conditions | | | | Results | |
|---|---|---|---|---|---|
| Pattern Byte | Previous State of Significance Indicator | Source Digit | Right Four Source Bits Are Plus Code | Result Byte | State of Significance Indicator at End of Digit Examination |
| Digit selector | Off | 0 | * | Fill byte | Off |
| | | 1-9 | No | Source digit# | On |
| | | 1-9 | Yes | Source digit# | Off |
| | On | 0-9 | No | Source digit | On |
| | | 0-9 | Yes | Source digit | Off |
| Significance starter | Off | 0 | No | Fill byte | On |
| | | 0 | Yes | Fill byte | Off |
| | | 1-9 | No | Source digit# | On |
| | | 1-9 | Yes | Source digit# | Off |
| | On | 0-9 | No | Source digit | On |
| | | 0-9 | Yes | Source digit | Off |
| Field separator | * | ** | ** | Fill byte | Off |
| Message byte | Off | ** | ** | Fill byte | Off |
| | On | ** | ** | Message byte | On |

**Explanation:**

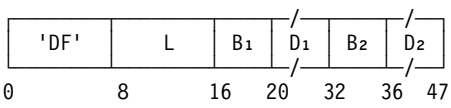
* No effect on result byte or on new state of significance indicator.
** Not applicable because source is not examined.
\# For EDIT AND MARK only, the address of the rightmost such result byte is placed in general register 1.

*Figure 8-3. Summary of Editing Functions*


LzLabs GmbH. Ex. 1008-459

## EDIT AND MARK

```
EDMK   D1(L,B1),D2(B2)          [SS]
```




The second operand (the source), which normally contains one or more decimal numbers in the packed format, is changed to the zoned format and modified under the control of the first operand (the pattern). The address of the first significant result byte is inserted in general register 1. The edited result replaces the pattern.

EDIT AND MARK is identical to EDIT, except for the additional function of inserting the address of the result byte in general register 1 if the result byte is a zoned source digit and the significance indicator was off before the examination. If no result byte meets the criteria, general register 1 remains unchanged; if more than one result byte meets the criteria, the address of the rightmost such result byte is inserted.

In the 24-bit addressing mode, the address replaces bits 8-31 of general register 1, and bits 0-7 of the register are not changed. In the 31-bit addressing mode, the address replaces bits 1-31 of general register 1, and bit 0 of the register is set to zero.

The contents of access register 1 remain unchanged.

See Figure 8-3 on page 8-11 for a summary of the EDIT and EDIT AND MARK operations.

***Resulting Condition Code:***

0 Last field zero or zero length
1 Last field less than zero
2 Last field greater than zero
3 --

***Program Exceptions:***

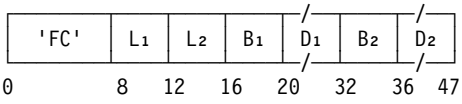
- Access (fetch, operand 2; fetch and store, operand 1)
- Data

**Programming Notes:**

1. Examples of the use of the EDIT AND MARK instruction are given Appendix A, “Number Representation and Instruction-Use Examples.”
2. EDIT AND MARK facilitates the programming of floating currency-symbol insertion. Using appropriate source and pattern data, the address inserted in general register 1 is one greater than the address where a floating currency-sign would be inserted. BRANCH ON COUNT (BCTR), with zero in the $R_2$ field, may be used to reduce the inserted address by one.
3. No address is inserted in general register 1 when the significance indicator is turned on as a result of encountering a significance starter with the corresponding source digit zero. To ensure that general register 1 contains a proper address when this occurs, the address of the pattern byte that immediately follows the appropriate significance starter could be placed in the register beforehand.
4. When multiple fields are edited with one execution of the EDIT AND MARK instruction, the address, if any, inserted in general register 1 applies to the rightmost field edited for which the criteria were met.
5. See also the programming note under EDIT regarding performance degradation due to a possible trial execution.

## MULTIPLY DECIMAL

```
MP     D1(L1,B1),D2(L2,B2)      [SS]
```




The product of the first operand (the multiplicand) and the second operand (the multiplier) is placed at the first-operand location. The operands and result are in the packed format.

The multiplier length cannot exceed 15 digits and sign ($L_2$ not greater than seven) and must be less than the multiplicand length ($L_2$ less than $L_1$); otherwise, a specification exception is recognized.



LzLabs GmbH. Ex. 1008-460

The multiplicand must have at least as many bytes of leftmost zeros as the number of bytes in the multiplier; otherwise, a data exception is recognized. This restriction ensures that no product overflow occurs.

The multiplicand, multiplier, and product are each signed decimal integers in the packed format and are right-aligned in their fields. All sign and digit codes of the multiplicand and multiplier are checked for validity. The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs, even if one or both operands are zeros.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

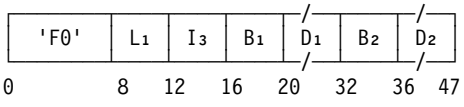
- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Specification

**Programming Notes:**

1. An example of the use of the MULTIPLY DECIMAL instruction is given Appendix A, “Number Representation and Instruction-Use Examples.”
2. The product cannot exceed 31 digits and sign. The leftmost digit of the product is always zero.

## SHIFT AND ROUND DECIMAL

SRP $D_1(L_1,B_1),D_2(B_2),I_3$ [SS]




The first operand is shifted in the direction and for the number of decimal-digit positions specified by the second-operand address, and, when shifting to the right is specified, the absolute value of the first operand is rounded by the rounding digit, $I_3$. The first operand and the result are in the packed format.

The first operand is considered to be in the packed-decimal format. Only its digit portion is shifted; the sign position does not participate in the shifting. Zeros are supplied for the vacated digit positions. The result replaces the first operand. Nothing is stored outside of the specified first-operand location.

The second-operand address, specified by the $B_2$ and $D_2$ fields, is not used to address data; bits 26-31 of that address are the shift value, and the leftmost bits of the address are ignored.

The shift value is a six-bit signed binary integer, indicating the direction and the number of decimal-digit positions to be shifted. Positive shift values specify shifting to the left. Negative shift values, which are represented in two's complement notation, specify shifting to the right. The following are examples of the interpretation of shift values:

| Shift Value (Binary) | Amount and Direction |
|---|---|
| 011111 | 31 digits to the left |
| 000001 | One digit to the left |
| 000000 | No shift |
| 111111 | One digit to the right |
| 100000 | 32 digits to the right |

For a right shift, the $I_3$ field, bits 12-15 of the instruction, is used as a decimal rounding digit. The first operand, which is treated as positive by ignoring the sign, is rounded by decimally adding the rounding digit to the leftmost of the digits to be shifted out and by propagating the carry, if any, to the left. The result of this addition is then shifted right. Except for validity checking and the participation in rounding, the digits shifted out of the rightmost decimal-digit position are ignored and are lost.

If one or more nonzero digits are shifted out during a left shift, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs. Overflow cannot occur for a right shift, with or without rounding, or when no shifting is specified.

In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, the sign of the result is the same as the original sign but with the preferred sign code.

A data exception is recognized when the first operand does not have valid sign and digit codes or when the rounding digit is not a valid digit code. The validity of the first-operand codes is checked


LzLabs GmbH. Ex. 1008-461

even when no shift is specified, and the validity of the rounding digit is checked even when no addition for rounding takes place.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

***Program Exceptions:***

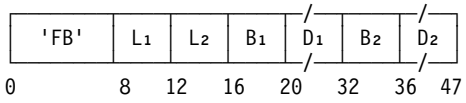
- Access (fetch and store, operand 1)
- Data
- Decimal overflow

**Programming Notes:**

1. Examples of the use of the SHIFT AND ROUND DECIMAL instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. SHIFT AND ROUND DECIMAL can be used for shifting up to 31 digit positions left and up to 32 digit positions right. This is sufficient to clear all digits of any decimal number even with rounding.
3. For right shifts, the rounding digit 5 provides conventional rounding of the result. The rounding digit 0 specifies truncation without rounding.
4. When the $B_2$ field is zero, the six-bit shift value is obtained directly from bits 42-47 of the instruction.

## SUBTRACT DECIMAL

SP D1(L1,B1),D2(L2,B2) [SS]




The second operand is subtracted from the first operand, and the resulting difference is placed at the first-operand location. The operands and result are in the packed format.

SUBTRACT DECIMAL is executed the same as ADD DECIMAL, except that the second operand is considered to have a sign opposite to the sign in storage. The second operand in storage remains unchanged.

***Resulting Condition Code:***

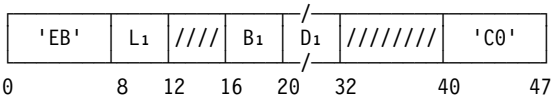
0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

***Program Exceptions:***

- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Decimal overflow

## TEST DECIMAL

TP D1(L1,B1) [RSL]




The first operand is tested for valid decimal digits and a valid sign code, and the result is indicated in the condition code. The operand is in the packed format.

***Resulting Condition Code:***

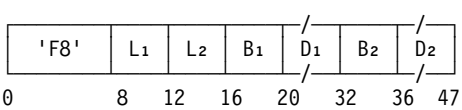
0 All digit codes and the sign valid
1 Sign invalid
2 At least one digit code invalid
3 Sign invalid and at least one digit code invalid

***Program Exceptions:***

- Access (fetch, operand 1)
- Operation (if the extended-translation facility 2 is not installed)

## ZERO AND ADD

ZAP D1(L1,B1),D2(L2,B2) [SS]




The second operand is placed at the first-operand location. The operation is equivalent to an addition to zero. The operand and result are in the packed format.



LzLabs GmbH. Ex. 1008-462

Only the second operand is checked for valid sign and digit codes. Extra zeros are supplied on the left for the shorter operand if needed.

If the first operand is too short to contain all leftmost nonzero digits of the second operand, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs.

In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, a zero result is given the sign of the second operand but with the preferred sign code.

The two operands may overlap, provided the rightmost byte of the first operand is coincident with or to the right of the rightmost byte of the second operand. In this case, the result is obtained as if the operands were processed right to left. When the operands overlap and the rightmost byte of the first operand is to the left of the rightmost byte of the second operand, then, depending on the model, either a data exception is recognized or the result is obtained as if the entire second operand were fetched before any byte of the result is stored.

***Resulting Condition Code:***

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Data
- Decimal overflow

**Programming Note:** An example of the use of the ZERO AND ADD instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”


LzLabs GmbH. Ex. 1008-463

LzLabs GmbH. Ex. 1008-464

# Chapter 9. Floating-Point Overview and Support Instructions


Registers And Controls . . . . . 9-2
Floating-Point Registers . . . . . 9-2
Additional Floating-Point (AFP) Registers . . . . . 9-2
Valid Floating-Point-Register Designations . . . . . 9-2
Floating-Point-Control (FPC) Register . . . 9-2
AFP-Register-Control Bit . . . . . 9-3
Explicit Rounding Methods . . . . . 9-3
Summary of Rounding Action . . . . . 9-3
Comparison of BFP and HFP Number Representations . . . . . 9-4
BFP and HFP Number Ranges . . . . . 9-4
Equivalent BFP and HFP Number Representations . . . . . 9-4
Instructions . . . . . 9-6
CONVERT BFP TO HFP . . . . . 9-8
CONVERT HFP TO BFP . . . . . 9-9
LOAD . . . . . 9-10
LOAD ZERO . . . . . 9-11
STORE . . . . . 9-11
Summary of All Floating-Point Instructions . . 9-11


Floating-point instructions are used to perform calculations on operands having a wide range of magnitude and to obtain results scaled to preserve precision.

Floating-point operands have formats based on either the radix 16 or, when the binary-floating-point facility is installed, the radix 2. The radix values 16 and 2 lead to the terminology “hexadecimal” and “binary” floating point (HFP and BFP). The formats are also based on three operand lengths: short (32 bits), long (64 bits), and extended (128 bits). Short operands require less storage than long or extended operands. On the other hand, long and extended operands permit greater precision in computation.

A floating-point operand may be numeric or, for BFP only, nonnumeric (a not-a-number, or NaN). A numeric operand, called a floating-point number, has three components: a sign bit, a signed binary exponent, and a significand. The significand consists of an implicit unit digit to the left of an implied radix point and an explicit fraction field to the right. The significand digits are based on the radix, 2 or 16. The magnitude (an unsigned value) of the number is the product of the significand and the radix raised to the power of the exponent. The number is positive or negative depending on whether the sign bit is zero or one, respectively. A nonnumeric BFP operand also has a sign bit, signed exponent, and fraction field.

Hexadecimal-floating-point (HFP) operands have formats which provide for exponents that specify powers of the radix 16 and significands that are hexadecimal numbers. The exponent range is the same for the short, long, and extended formats. The results of most operations on HFP data are truncated to fit into the target format, but there are instructions available to round the result when converting to a narrower format. For HFP operands, the implicit unit digit of the significand is always zero. Since the value of the significand and fraction are the same, HFP operations are described in terms of the fraction, and the term significand is not used.

Binary-floating-point (BFP) operands have formats which provide for exponents that specify powers of the radix 2 and significands that are binary numbers. The exponent range differs for different formats, the range being greater for the longer formats. In the long and extended formats, the exponent range is significantly greater for BFP data than for HFP data. The results of operations performed on BFP data are rounded automatically to fit into the target format; the manner of rounding is determined by a program-settable rounding mode.

Either normalized or unnormalized numbers may be used as operands for any HFP operation, where a normalized number is one having a nonzero leftmost fraction digit. Most HFP instructions generate normalized results for greatest precision. HFP add and subtract instructions that generate unnormalized results are also available.

There are no unnormalized operands for BFP operations. For normalized BFP numbers, the

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-465

implicit unit digit of the significand is one. For values too small in magnitude to be represented in normalized form, the implicit unit digit is zero. These numbers are called "denormalized" numbers. Unlike the HFP format, where the same value can have multiple representations in a given format because of the possibility of unnormalized numbers, the BFP format does not allow such redundancy.

Both BFP and HFP data formats appear in storage in the same left-to-right sequence as all other data formats. Bits of a data format that are numbered 0-7 constitute the byte in the leftmost (lowest-numbered) byte location in storage, bits 8-15 form the byte in the next sequential location, and so on. (See also the section "Storage Addressing" on page 3-2.)

Most of the floating-point instructions are defined in detail in this publication in Chapter 18, "Hexadecimal-Floating-Point Instructions," and Chapter 19, "Binary-Floating-Point Instructions." This chapter, Chapter 9, defines in detail instructions called floating-point-support (FPS) instructions. The FPS instructions either have operands that may be in either the BFP or the HFP format or have the function of converting between the two formats. This chapter also provides summary information about all of the floating-point instructions.

# Registers And Controls

## Floating-Point Registers

All floating-point instructions (FPS, BFP, and HFP) use the same floating-point registers. When the basic-floating-point-extensions facility is installed, the CPU has 16 floating-point registers. The floating-point registers are identified by the numbers 0-15 and are designated by a four-bit R field in floating-point instructions. Each floating-point register is 64 bits long and can contain either a short (32-bit) or a long (64-bit) floating-point operand.

A short floating-point number requires only the leftmost 32 bit positions of a floating-point register. The rightmost 32 bit positions of the register are ignored when the register is the source of an operand in the short format, and they remain unchanged when a short result is placed in the register.

A number in the extended (128-bit) format occupies a register pair. Register pairs are formed by coupling the 16 registers as follows: 0 and 2, 4 and 6, 8 and 10, 12 and 14, 1 and 3, 5 and 7, 9 and 11, and 13 and 15.

Each of the eight pairs is referred to by the number of the lower-numbered register of the pair.

### Additional Floating-Point (AFP) Registers

Floating-point registers 0, 2, 4, and 6 are available on all ESA/390 models. The remaining 12 floating-point registers (1, 3, 5, and 7-15) are referred to as the additional floating-point (AFP) registers. The AFP registers are installed in the CPU when the basic-floating-point-extensions facility is installed and can be used only if bit 13 of control register 0, the AFP-register-control bit, is one. Attempting to use an AFP register when the basic-floating-point-extensions facility is not installed results in a specification exception. Attempting to use an AFP register when the basic-floating-point-extensions facility is installed and the AFP-register-control bit is zero results in an AFP-register data exception (DXC 1).

### Valid Floating-Point-Register Designations

Any installed register may be designated by an instruction to specify the register location of a short or long floating-point operand.

An instruction specifying a floating-point operand in the extended format must designate register 0 or 4, if only registers 0, 2, 4, and 6 are installed, or register 0, 1, 4, 5, 8, 9, 12, or 13 if all registers are installed; otherwise, a specification exception is recognized.

## Floating-Point-Control (FPC) Register

The floating-point-control (FPC) register is a 32-bit register that contains mask bits, flag bits, a data-exception code, and rounding-mode bits. The FPC register is installed when the binary-floating-point facility is installed and is described in the section "Floating-Point-Control (FPC) Register" on page 19-2.



LzLabs GmbH. Ex. 1008-466

## AFP-Register-Control Bit

Bit 13 of control register 0 is the AFP-register-control bit. The AFP registers and the BFP instructions can be used successfully only when the AFP-register-control bit is one. Attempting to use one of the 12 additional floating-point registers when the AFP-register-control bit is zero results in an AFP-register data exception (DXC 1). Attempting to execute any BFP instruction when the AFP-register-control bit is zero results in a BFP-instruction data exception (DXC 2). If the conditions for both DXC 1 and DXC 2 exist, DXC 1 is reported. If the conditions for both a data exception and a specification exception exist, it is unpredictable which exception is reported.

The initial value of the AFP-register-control bit is zero.

## Explicit Rounding Methods

The floating-point-support instruction CONVERT HFP TO BFP includes an $M_3$ modifier field which can specify any of five rounding methods. One HFP instruction (CONVERT TO FIXED) and three BFP instructions (CONVERT TO FIXED, DIVIDE TO INTEGER, and LOAD FP INTEGER) also include either an $M_3$ modifier field or a similar $M_4$ modifier field. The five rounding methods are as follows:

**$M_3$ or $M_4$ Rounding Method**

1 Biased round to nearest:

Round the intermediate result up or down to the nearest representable value; that is, add, ignoring the sign, a one to the bit just beyond the last result bit to be retained, propagate the carry, and discard the bits beyond the last one to be retained.

4 Round to nearest:

Round the intermediate result up or down to the nearest representable value; that is, add, ignoring the sign, a one to the bit just beyond the last result bit to be retained, propagate the carry, and discard the bits beyond the last one to be retained. If the difference was exactly one-half ulp (a one in the bit position just beyond the last place, with all zeros beyond that), the nearest even number is chosen; that is, after the rounding addition, the last result bit retained is set to zero.

5 Round toward 0:

Discard all bits to the right of the last intermediate-result bit to be retained.

6 Round toward $+\infty$:

If the intermediate result is positive and there are any ones to the right of the last result bit to be retained, add one to that bit. Then, for either sign, discard the bits beyond the last one to be retained.

7 Round toward $-\infty$:

If the intermediate result is negative and there are any ones to the right of the last result bit to be retained, subtract one from that bit (that is, add one to the magnitude). Then, for either sign, discard the bits beyond the last one to be retained.

The handling of an $M_3$ or $M_4$ value of zero depends on the type of instruction. For BFP instructions, an $M_3$ or $M_4$ value of zero causes rounding to be performed according to the current rounding mode specified in the FPC register. The floating-point-support and HFP instructions treat an $M_3$ or $M_4$ of zero the same as 5, that is, round toward zero.

### Summary of Rounding Action

Figure 9-1 on page 9-4 summarizes the rounding action for floating-point-support (FPS), BFP, and HFP instructions.



LzLabs GmbH. Ex. 1008-467

| Instruction | Rounding Action For | | |
|---|---|---|---|
| | FPS Inst. | HFP Inst. | BFP Inst. |
| ADD | — | — | CRM |
| ADD NORMALIZED | — | GD | — |
| ADD UNNORMALIZED | — | GD | — |
| CONVERT BFP TO HFP | E | — | — |
| CONVERT FROM FIXED | — | RTZ | CRM |
| CONVERT HFP TO BFP | M | — | — |
| CONVERT TO FIXED | — | M | M |
| DIVIDE | — | RTZ | CRM |
| DIVIDE TO INTEGER | — | — | M |
| HALVE | — | RTZ | — |
| LOAD FP INTEGER | — | RTZ | M |
| LOAD ROUNDED | — | BR | CRM |
| MULTIPLY | — | RTZ | CRM |
| MULTIPLY AND ADD | — | — | CRM |
| MULTIPLY AND SUBTRACT | — | — | CRM |
| SQUARE ROOT | — | BR | CRM |
| SUBTRACT | — | — | CRM |
| SUBTRACT NORMALIZED | — | GD | — |
| SUBTRACT UNNORMALIZED | — | GD | — |

**Explanation:**

BR Biased round to nearest.
CRM Rounded according to current rounding mode.
E Result is exact, no rounding is required.
GD Round using a guard digit; see the instruction definition. This is almost, but not quite, round toward 0.
M Rounding is specified by a modifier field in the instruction.
RTZ Round toward 0.

*Figure 9-1. Comparison of Rounding Action*

# Comparison of BFP and HFP Number Representations

## BFP and HFP Number Ranges

Figure 9-2 shows the range of numbers, in decimal form, that can be represented in different floating-point formats.

| | Type | Short | Long | Extended |
|---|---|---|---|---|
| Nmax | BFP | $\pm 3.4\times10^{+38}$ | $\pm 1.8\times10^{+308}$ | $\pm 1.2\times10^{+4932}$ |
| | HFP | $\pm 7.2\times10^{+75}$ | $\pm 7.2\times10^{+75}$ | $\pm 7.2\times10^{+75}$ |
| Nmin | BFP | $\pm 1.2\times10^{-38}$ | $\pm 2.2\times10^{-308}$ | $\pm 3.4\times10^{-4932}$ |
| | HFP | $\pm 5.5\times10^{-79}$ | $\pm 5.5\times10^{-79}$ | $\pm 5.5\times10^{-79}$ |
| Dmin | BFP | $\pm 1.4\times10^{-45}$ | $\pm 4.9\times10^{-324}$ | $\pm 6.5\times10^{-4966}$ |
| | HFP | $\pm 5.2\times10^{-85}$ | $\pm 1.2\times10^{-94}$ | $\pm 1.7\times10^{-111}$ |

**Explanation:**

Dmin Smallest (in magnitude) representable denormalized (BFP) or nonzero unnormalized (HFP) number.
Nmax Largest (in magnitude) representable number.
Nmin Smallest (in magnitude) representable normalized number.

Values are decimal approximations.

*Figure 9-2. Number Ranges for BFP and HFP Formats*

## Equivalent BFP and HFP Number Representations

The exponent of an HFP number is represented in the number as an unsigned seven-bit binary integer called the characteristic. The characteristic is obtained by adding 64 to the exponent value (excess-64 notation). The range of the characteristic is 0 to 127, which corresponds to an exponent range of –64 to +63.

The exponent of a BFP number is represented in the number as an unsigned binary integer called the biased exponent. The biased exponent is obtained by adding a bias to the exponent value. The number of bit positions containing the biased exponent, the value of the bias, and the exponent range depend on the number format (short, long, or extended) and are shown for the three formats in Figure 19-7 on page 19-5. Biased exponents



LzLabs GmbH. Ex. 1008-468

are similar to the characteristics of the HFP format, except that special meanings are attached to biased exponents of all zeros and all ones, which are discussed in the section "Classes of BFP Data" on page 19-5.

In each of the three BFP or HFP formats, the binary or hexadecimal point of a number, respectively, is considered to be to the left of the leftmost fraction digit. To the left of the point there is an implied unit digit, which is considered to be zero for HFP numbers or, for BFP numbers, one for normalized numbers and zero for zeros and denormalized numbers.

Figure 9-3 and Figure 9-4 on page 9-6 give examples of the closest representation of the same numbers in the BFP and HFP formats, with BFP values being rounded to nearest and HFP values being truncated.

The figures do not necessarily show the results of BFP/HFP conversions exactly. Rounding errors may make a small difference. Also, Figure 9-3 shows corresponding rounded short-format numbers, not the long HFP results of conversion from short BFP operands.

| Value | | S BE or C Fraction |
|---|---|---|
| 1.0 | B | 0 01111111 00000000000000000000000 |
| | H | 0 1000001 00010000000000000000000 |
| 0.5 | B | 0 01111110 00000000000000000000000 |
| | H | 0 1000000 10000000000000000000000 |
| 1/64 | B | 0 01111001 00000000000000000000000 |
| | H | 0 0111111 01000000000000000000000 |
| +0 | B | 0 00000000 00000000000000000000000 |
| | H | 0 0000000 00000000000000000000000 |
| −0 | B | 1 00000000 00000000000000000000000 |
| | H | 1 0000000 00000000000000000000000 |
| −15.0 | B | 1 10000010 11100000000000000000000 |
| | H | 1 1000001 11110000000000000000000 |
| 20/7 | B | 0 10000000 01101101101101101101110 |
| | H | 0 1000001 00101101101101101101011 |
| $2^{-126}$ | B | 0 00000001 00000000000000000000000 |
| | H | 0 0100001 01000000000000000000000 |
| $2^{-149}$ | B | 0 00000000 00000000000000000000001 |
| | H | 0 0011011 10000000000000000000000 |
| $2^{128}\times F$ $F=1-2^{-24}$ | B | 0 11111110 11111111111111111111111 |
| | H | 0 1100000 11111111111111111111111 |
| $2^{-260}$ | B | Zero (number too small) |
| | H | 0 0000000 00010000000000000000000 |
| $2^{248}\times F$ $F=1-2^{-24}$ | B | Not representable |
| | H | 0 1111110 11111111111111111111111 |

**Explanation:**

| | |
|---|---|
| B | BFP. |
| BE or C | Biased exponent of BFP number or characteristic of HFP number. |
| H | HFP. |
| S | Sign. |

*Figure 9-3. Examples of FP and HFP Numbers in Short Format*



LzLabs GmbH. Ex. 1008-469

| Value | | S BE or C Fraction |
|---|---|---|
| 1.0 | B | 0 01111111111 00000000000000000000 00000000000000000000000000000000 |
| | H | 0 1000001 00010000000000000000000 00000000000000000000000000000000 |
| 0.5 | B | 0 01111111110 00000000000000000000 00000000000000000000000000000000 |
| | H | 0 1000000 10000000000000000000000 00000000000000000000000000000000 |
| 1/64 | B | 0 01111111001 00000000000000000000 00000000000000000000000000000000 |
| | H | 0 0111111 01000000000000000000000 00000000000000000000000000000000 |
| +0 | B | 0 00000000000 00000000000000000000 00000000000000000000000000000000 |
| | H | 0 0000000 00000000000000000000000 00000000000000000000000000000000 |
| −0 | B | 1 00000000000 00000000000000000000 00000000000000000000000000000000 |
| | H | 1 0000000 00000000000000000000000 00000000000000000000000000000000 |
| −15.0 | B | 1 10000000010 11100000000000000000 00000000000000000000000000000000 |
| | H | 1 1000001 11110000000000000000000 00000000000000000000000000000000 |
| 20/7 | B | 0 10000000000 01101101101101101101 10110110110110110110110110110111 |
| | H | 0 1000001 00101101101101101101101 01101101101101101101101101101101 |
| $2^{-1022}$ | B | 0 00000000001 00000000000000000000 00000000000000000000000000000000 |
| | H | Zero (number too small) |
| $2^{-1074}$ | B | 0 00000000000 00000000000000000000 00000000000000000000000000000001 |
| | H | Zero (number too small) |
| $2^{1024}\times F$ $F=1-2^{-53}$ | B | 0 11111111110 11111111111111111111 11111111111111111111111111111111 |
| | H | Not representable |
| $2^{-260}$ | B | 0 01011111011 00000000000000000000 00000000000000000000000000000000 |
| | H | 0 0000000 00010000000000000000000 00000000000000000000000000000000 |
| $2^{248}\times F$ $F=1-2^{-56}$ | B | 0 10011110111 00000000000000000000 00000000000000000000000000000000 |
| | H | 0 1111110 11111111111111111111111 11111111111111111111111111111111 |

**Explanation:**

| | |
|---|---|
| B | BFP. |
| BE or C | Biased exponent of BFP number or characteristic of HFP number. |
| H | HFP. |
| S | Sign. |

*Figure 9-4. Examples of BFP and HFP Numbers in Long Format*

## Instructions

The floating-point-support instructions and their mnemonics and operation codes are listed in Figure 9-5 on page 9-7. The figure indicates, in the column labeled “Characteristics,” the instruction format, when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

All floating-point-support instructions are subject to the AFP-register-control bit, bit 13 of control register 0. The AFP-register-control bit must be one when an AFP register is specified as an operand location; otherwise, an AFP-register data exception, DXC 1, is recognized. An operation exception is recognized when the CPU attempts to execute an instruction which is part of the floating-point-support extensions facility when the facility is not installed.

Mnemonics for the floating-point instructions have an R as the last letter when the instruction is in the RR, RRE, or RRF format. Certain letters are used for floating-point instructions to represent operand-format length, as follows:

D Long
E Short
X Extended



LzLabs GmbH. Ex. 1008-470

**Note:** In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For a register-to-register operation using LOAD (short), for example, LER is the mnemonic and $R_1$,$R_2$ the operand designation.

**Programming Note:** The following additional floating-point-support instructions are available when the floating-point-support-extensions facility is installed:

- CONVERT BFP TO HFP
- CONVERT HFP TO BFP
- LOAD (LXR)
- LOAD ZERO

| Name | Mnemonic | Characteristics | | | | | Op Code |
|---|---|---|---|---|---|---|---|
| CONVERT BFP TO HFP (long) | THDR | RRE C FX | | Da | | | B359 |
| CONVERT BFP TO HFP (short to long) | THDER | RRE C FX | | Da | | | B358 |
| CONVERT HFP TO BFP (long) | TBDR | RRF C FX | SP | Da | | | B351 |
| CONVERT HFP TO BFP (long to short) | TBEDR | RRF C FX | SP | Da | | | B350 |
| LOAD (extended) | LXR | RRE FX | SP | Da | | | B365 |
| LOAD (long) | LDR | RR | SP | Da | | | 28 |
| LOAD (long) | LD | RX | A SP | Da | | $B_2$ | 68 |
| LOAD (short) | LER | RR | SP | Da | | | 38 |
| LOAD (short) | LE | RX | A SP | Da | | $B_2$ | 78 |
| LOAD ZERO (extended) | LZXR | RRE FX | SP | Da | | | B376 |
| LOAD ZERO (long) | LZDR | RRE FX | | Da | | | B375 |
| LOAD ZERO (short) | LZER | RRE FX | | Da | | | B374 |
| STORE (long) | STD | RX | A SP | Da | ST | $B_2$ | 60 |
| STORE (short) | STE | RX | A SP | Da | ST | $B_2$ | 70 |

**Explanation:**

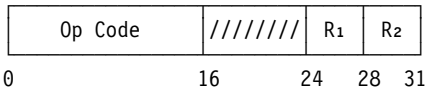
A Access exceptions for logical addresses.
$B_2$ $B_2$ field designates an access register in the access-register mode.
C Condition code is set.
Da AFP-register data exception.
FX Floating-point-support extensions facility.
RR RR instruction format.
RRE RRE instruction format.
RRF RRF instruction format.
RX RX instruction format.
SP Specification exception.
ST PER storage-alteration event.

*Figure 9-5. Summary of Floating-Point-Support Instructions*


LzLabs GmbH. Ex. 1008-471

## CONVERT BFP TO HFP

```
Mnemonic    R1,R2              [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| THDER | 'B358' | Short BFP operand, long HFP result |
| THDR | 'B359' | Long BFP operand, long HFP result |

The second operand (the source operand) is converted from the binary-floating-point (BFP) format to the hexadecimal-floating-point (HFP) format, and the normalized result is placed at the first-operand location. The sign and magnitude of the source operand are tested to determine the setting of the condition code.

For numeric operands, the sign of the result is the sign of the source operand. If the source operand has a sign bit of one and all other operand bits are zeros, the result also is a one followed by all zeros.

When, for THDR, the characteristic of the result would be negative, the result is made all zeros but with the same sign as that of the source operand, and condition code 1 or 2 is set to indicate the sign of the source operand.

When, for THDR, the characteristic of the hexadecimal intermediate result is too large to fit into the target format, the result is set to all ones (that is, the largest-in-magnitude representable number) but with the same sign as that of the source operand, and condition code 3 is set.

See Figure 9-6 for a detailed description of the results of this instruction.

***Resulting Condition Code:***

0 Source was zero
1 Source was less than zero
2 Source was greater than zero
3 Special case

***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (if the floating-point-support-extensions facility is not installed)

**Programming Notes:**

1. The BFP-to-HFP conversion instructions are summarized in Figure 9-7 on page 9-9.
2. CONVERT BFP TO HFP (THDER) converts BFP operands in the short format to HFP operands in the long format, rather than converting short to short, to retain full precision. Using this long HFP result subsequently as a short operand requires no extra conversion steps.

| Source Operand (a) | Results |
|---|---|
| $-\infty \le a < -Hmax$ | T(–Hmax), cc3 |
| $-Hmax \le a \le -Hmin$ | T(r), cc1 |
| $-Hmin < a < 0$ | T(–0)[1], cc1 |
| –0 | T(–0), cc0 |
| +0 | T(+0), cc0 |
| $0 < a < +Hmin$ | T(+0)[2], cc2 |
| $+Hmin \le a \le +Hmax$ | T(r), cc2 |
| $+Hmax < a \le +\infty$ | T(+Hmax), cc3 |
| NaN | T(+Hmax), cc3 |

**Explanation:**

[1] Condition code 1 is set to indicate the source was less than zero.
[2] Condition code 2 is set to indicate the source was greater than zero.
ccn Condition code is set to n.
r The value derived when the BFP source value a is converted to the HFP format. This result is always exact.
Hmax Largest (in magnitude) representable number in the target HFP format.
Hmin Smallest (in magnitude) representable normalized number in the target HFP format.
T(x) The value x is placed at the target operand location.

*Figure 9-6. Results: CONVERT BFP TO HFP*



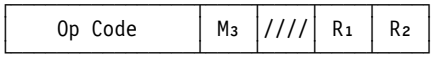
LzLabs GmbH. Ex. 1008-472

| Instruction | Mnemonic | Source | | Target | | | Overflow, Underflow Possible |
|---|---|---|---|---|---|---|---|
| | | Format | Significant Bits | Format | Significant Bits | Result | |
| CONVERT BFP TO HFP | THDER | BFP short | 24 | HFP long | 53-56 | Exact | No |
| | THDR | BFP long | 53 | HFP long | 53-56 | Exact | Yes |
| CONVERT HFP TO BFP | TBEDR | HFP long | 53-56 | BFP short | 24 | Rounded | Yes |
| | TBDR | HFP long | 53-56 | BFP long | 53 | Rounded | No |

*Figure 9-7. Summary of BFP-to/from-HFP Conversion Instructions*

## CONVERT HFP TO BFP

```
Mnemonic     R1,M3,R2          [RRF]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| TBEDR | 'B350' | Long HFP operand, short BFP result |
| TBDR | 'B351' | Long HFP operand, long BFP result |

The second operand (the source operand) is converted from the hexadecimal-floating-point (HFP) format to the binary-floating-point (BFP) format, and the result rounded according to the rounding method specified by the $M_3$ field is placed at the first-operand location. The sign and magnitude of the source operand are tested to determine the setting of the condition code.

The $M_3$ field contains a modifier specifying a rounding method, as follows:

| $M_3$ | Rounding Method |
|---|---|
| 0 | Round toward 0 |
| 1 | Biased round to nearest |
| 4 | Round to nearest |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |
| 7 | Round toward $-\infty$ |

A modifier other than 0, 1, or 4-7 is invalid.

The sign of the result is the sign of the second operand. If the second operand has a sign bit of one and all other operand bits are zeros, the result also is a one followed by all zeros.

See Figure 9-8 on page 9-10 for a detailed description of the results of this instruction.

The $M_3$ field must designate a valid modifier; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Source was zero
1 Source was less than zero
2 Source was greater than zero
3 Special case

***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (if the floating-point-support-extensions facility is not installed)
- Specification

**Programming Notes:**

1. The HFP-to-BFP conversion instructions are summarized in Figure 9-7.
2. Conversion to short BFP numbers requires HFP operands in the long format; a short HFP operand should be extended to long by ensuring that the right half of the register is cleared. Thus, the entire register should be cleared before loading a short HFP operand into it for conversion to BFP. This avoids unrepeatable rounding errors in the BFP result due to data left over from previous use.


LzLabs GmbH. Ex. 1008-473

| Source Operand (a) | Results |
|---|---|
| a < –Nmax | See Part 2 of this figure. |
| –Nmax ≤ a ≤ –Nmin | T(r), cc1 |
| –Nmin < a ≤ –Dmin | T(d), cc1 |
| –Dmin < a < 0 | T(d)[1], cc1 |
| –0 | T(–0), cc0 |
| +0 | T(+0), cc0 |
| 0 < a < +Dmin | T(d)[2], cc2 |
| +Dmin ≤ a < +Nmin | T(d), cc2 |
| +Nmin ≤ a ≤ +Nmax | T(r), cc2 |
| +Nmax < a | See Part 2 of this figure. |

*Figure 9-8 (Part 1 of 2). Results: CONVERT HFP to BFP*

| Source Operand (a) | Results for Rounding Method Specified in $M_3$ | | | | |
|---|---|---|---|---|---|
| | Biased Round to Nearest | Round to Nearest | Round toward 0 | Round toward +∞ | Round toward –∞ |
| a < –Nmax | T(–∞), cc3 | T(–∞), cc3 | T(–Nmax), cc3 | T(–Nmax), cc3 | T(–∞), cc3 |
| +Nmax < a | T(+∞), cc3 | T(+∞), cc3 | T(+Nmax), cc3 | T(+∞), cc3 | T(+Nmax), cc3 |

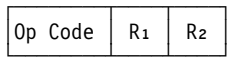
**Explanation:**

| | |
|---|---|
| [1] | Condition code 1 is set for this case, even when the rounded result is zero. |
| [2] | Condition code 2 is set for this case, even when the rounded result is zero. |
| ccn | Condition code is set to n. |
| d | The denormalized value derived when the HFP source value a is rounded to the format of the target using the rounding method specified in the $M_3$ field. |
| r | The value derived when the HFP source value a is rounded to the format of the target using the rounding method specified in the $M_3$ field. |
| Dmin | Smallest (in magnitude) representable denormalized number in the target BFP format. |
| Nmax | Largest (in magnitude) representable finite number in the target BFP format. |
| Nmin | Smallest (in magnitude) representable normalized number in the target BFP format. |
| T(x) | The value x is placed at the target operand location. |

*Figure 9-8 (Part 2 of 2). Results: CONVERT HFP to BFP*

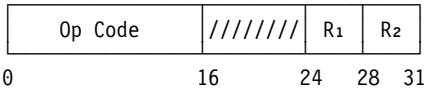
# LOAD

Mnemonic1 $R_1$,$R_2$ [RR]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| LER | '38' | Short |
| LDR | '28' | Long |

Mnemonic2 $R_1$,$R_2$ [RRE]

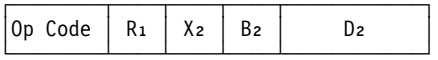



| Mnemonic2 | Op Code | Operands |
|---|---|---|
| LXR | 'B365' | Extended |



LzLabs GmbH. Ex. 1008-474

```
Mnemonic3    R1,D2(X2,B2)          [RX]
```




| Mnemonic3 | Op Code | Operands |
|---|---|---|
| LE | '78' | Short |
| LD | '68' | Long |

The second operand is placed unchanged at the first-operand location.

The operation is performed without inspecting the contents of the second operand; no arithmetic exceptions are recognized.

For LXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional-floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

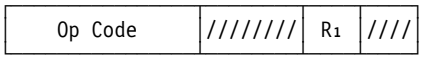
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2 of LE and LD only)
- Data with DXC 1, AFP register
- Operation (LXR if the floating-point-support-extensions facility is not installed)
- Specification

## LOAD ZERO

```
Mnemonic     R1                    [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| LZER | 'B374' | Short |
| LZDR | 'B375' | Long |
| LZXR | 'B376' | Extended |

All bits of the first operand are set to zeros.

For LZXR, The $R_1$ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

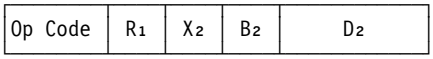
***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (if the floating-point-support-extensions facility is not installed)
- Specification (LZXR only)

**Programming Note:** LOAD ZERO sets all bits of a register to zeros, which produces a positive zero value in both the HFP and BFP formats.

## STORE

```
Mnemonic     R1,D2(X2,B2)          [RX]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| STE | '70' | Short |
| STD | '60' | Long |

The first operand is placed unchanged in storage at the second-operand location.

The $R_1$ field may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Data with DXC 1, AFP register
- Specification

## Summary of All Floating-Point Instructions

Figures 9-9 through 9-13 on following pages show all floating-point instructions arranged in various categories of operand format and type of operation (principally, register-and-register and register-and-storage operations). Figure 9-9 on page 9-12 shows the floating-point-support instructions. Figure 9-10 on page 9-13 shows the BFP and HFP instructions with all operands of the same length. Figure 9-11 on page 9-14 shows the BFP and HFP instructions in which the result is longer than the source operand. Figure 9-12 on page 9-14 shows the BFP and HFP instructions in which the result is shorter than the source operand. Figure 9-13 on page 9-14


LzLabs GmbH. Ex. 1008-475

shows the other BFP instructions, including those instructions which operate on the FPC register. Figure 9-14 on page 9-15 shows the numbers of all floating-point instructions in the categories of "basic" (those provided in ESA/390, before the introduction of the new floating-point facilities); "FX" (those provided as part of the floating-point-support-extensions facility); "HX" (those provided as part of the HFP-extensions facility); "BFP" (those provided as part of the BFP facility); and the totals. The instructions CONVERT FROM FIXED and CONVERT TO FIXED convert between fixed-point and floating-point formats. In the figures, entries for these 32-bit fixed-point operands are combined in the same column with entries for 32-bit short operands.

| | Short (32) | | Long (64) | | Ext. (128) | 32 to 64 | 64 to 32 |
|---|---|---|---|---|---|---|---|
| Instruction Name | R-R | R-S | R-R | R-S | R-R | R-R | R-R |
| CONVERT BFP TO HFP | | | THDR[1] | | | THDER[1] | |
| CONVERT HFP TO BFP | | | TBDR[1] | | | | TBEDR[1] |
| LOAD | LER | LE | LDR | LD | LXR[1] | | |
| LOAD ZERO | LZER[1] | | LZDR[1] | | LZXR[1] | | |
| STORE | | STE | | STD | | | |

**Explanation:**

[1] Operation code is part of floating-point-support-extensions facility.
R-R Register-and-register operation.
R-S Register-and-storage operation.

*Figure 9-9. Floating-Point-Support Instructions*



LzLabs GmbH. Ex. 1008-476

| | HFP Instructions | | | | | BFP Instructions | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Short (32) | | Long (64) | | Ext. (128) | Short (32) | | Long (64) | | Ext. (128) |
| Instruction Name | R-R | R-S | R-R | R-S | R-R | R-R | R-S | R-R | R-S | R-R |
| ADD | | | | | | AEBR[3] | AEB[3] | ADBR[3] | ADB[3] | AXBR[3] |
| ADD NORMALIZED | AER | AE | ADR | AD | AXR | | | | | |
| ADD UNNORMALIZED | AUR | AU | AWR | AW | | | | | | |
| COMPARE | CER | CE | CDR | CD | CXR[2] | CEBR[3] | CEB[3] | CDBR[3] | CDB[3] | CXBR[3] |
| COMPARE AND SIGNAL | | | | | | KEBR[3] | KEB[3] | KDBR[3] | KDB[3] | KXBR[3] |
| CONVERT FROM FIXED[1] | CEFR[2] | | | | | CEFBR[3] | | | | |
| CONVERT TO FIXED[1] | CFER[2] | | | | | CFEBR[3] | | | | |
| DIVIDE | DER | DE | DDR | DD | DXR | DEBR[3] | DEB[3] | DDBR[3] | DDB[3] | DXBR[3] |
| DIVIDE TO INTEGER | | | | | | DIEBR[3] | | DIDBR[3] | | |
| HALVE | HER | | HDR | | | | | | | |
| LOAD AND TEST | LTER | | LTDR | | LTXR[2] | LTEBR[3] | | LTDBR[3] | | LTXBR[3] |
| LOAD COMPLEMENT | LCER | | LCDR | | LCXR[2] | LCEBR[3] | | LCDBR[3] | | LCXBR[3] |
| LOAD FP INTEGER | FIER[2] | | FIDR[2] | | FIXR[2] | FIEBR[3] | | FIDBR[3] | | FIXBR[3] |
| LOAD NEGATIVE | LNER | | LNDR | | LNXR[2] | LNEBR[3] | | LNDBR[3] | | LNXBR[3] |
| LOAD POSITIVE | LPER | | LPDR | | LPXR[2] | LPEBR[3] | | LPDBR[3] | | LPXBR[3] |
| MULTIPLY[1] | MEER[2] | MEE[2] | MDR | MD | MXR | MEEBR[3] | MEEB[3] | MDBR[3] | MDB[3] | MXBR[3] |
| MULTIPLY AND ADD | | | | | | MAEBR[3] | MAEB[3] | MADBR[3] | MADB[3] | |
| MULTIPLY AND SUBTRACT | | | | | | MSEBR[3] | MSEB[3] | MSDBR[3] | MSDB[3] | |
| SQUARE ROOT | SQER | SQE[2] | SQDR | SQD[2] | SQXR[2] | SQEBR[3] | SQEB[3] | SQDBR[3] | SQDB[3] | SQXBR[3] |
| SUBTRACT | | | | | | SEBR[3] | SEB[3] | SDBR[3] | SDB[3] | SXBR[3] |
| SUBTRACT NORMALIZED | SER | SE | SDR | SD | SXR | | | | | |
| SUBTRACT UNNORMALIZED | SUR | SU | SWR | SW | | | | | | |
| TEST DATA CLASS | | | | | | TCEB[3] | | TCDB[3] | | TCXB[3] |

**Explanation:**

[1] This instruction also has mixed-length operands.
[2] Operation code is part of HFP-extensions facility.
[3] Operation code is part of the BFP facility.
R-R Register-and-register operation.
R-S Register-and-storage operation.

*Figure 9-10. BFP and HFP Instructions with All Operands of Same Length*


LzLabs GmbH. Ex. 1008-477

| | HFP Instructions | | | | | | BFP Instructions | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 32 to 64 | | 64 to 128 | | 32 to 128 | | 32 to 64 | | 64 to 128 | | 32 to 128 | |
| Instruction Name | R-R | R-S | R-R | R-S | R-R | R-S | R-R | R-S | R-R | R-S | R-R | R-S |
| CONVERT FROM FIXED | CDFR[2] | | | | CXFR[2] | | CDFBR[3] | | | | CXFBR[3] | |
| LOAD LENGTHENED | LDER[2] | LDE[2] | LXDR[2] | LXD[2] | LXER[2] | LXE[2] | LDEBR[3] | LDEB[3] | LXDBR[3] | LXDB[3] | LXEBR[3] | LXEB[3] |
| MULTIPLY | MDER | MDE | MXDR | MXD | | | MDEBR[3] | MDEB[3] | MXDBR[3] | MXDB[3] | | |

**Explanation:**

[2] Operation code is part of the HFP-extensions facility.
[3] Operation code is part of the BFP facility.
R-R Register-and-register operation.
R-S Register-and-storage operation.

*Figure 9-11. BFP and HFP Instructions with Result Longer than Source*

| | HFP Instructions | | | BFP Instructions | | |
|---|---|---|---|---|---|---|
| | 64 to 32 | 128 to 64 | 128 to 32 | 64 to 32 | 128 to 64 | 128 to 32 |
| Instruction Name | R-R | R-R | R-R | R-R | R-R | R-R |
| CONVERT TO FIXED | CFDR[2] | | CFXR[2] | CFDBR[3] | | CFXBR[3] |
| LOAD ROUNDED | LEDR | LDXR | LEXR[2] | LEDBR[3] | LDXBR[3] | LEXBR[3] |

**Explanation:**

[2] Operation code is part of the HFP-extensions facility.
[3] Operation code is part of the BFP facility.
R-R Register-and-register operation.

*Figure 9-12. BFP and HFP Instructions with Result Shorter than Source*

| Instruction Name | Mnemonic |
|---|---|
| EXTRACT FPC | EFPC[3] |
| LOAD FPC | LFPC[3] |
| SET FPC | SFPC[3] |
| SET ROUNDING MODE | SRNM[3] |
| STORE FPC | STFPC[3] |

**Explanation:**

[3] Operation code is part of the BFP facility.

*Figure 9-13. Other BFP Instructions*



LzLabs GmbH. Ex. 1008-478

| Length of Operands (Bits) | Number of Operation Codes | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Basic | | | FX | HX | | | BFP | | | | Total | | | |
| | R-R | R-S | Total | R-R | R-R | R-S | Total | R-R | R-S | Other | Total | R-R | R-S | Other | Total |
| Short (32) | 13 | 8 | 21 | 1 | 4 | 2 | 6 | 18 | 9 | | 27 | 36 | 19 | | 55 |
| Long (64) | 14 | 9 | 23 | 3 | 1 | 1 | 2 | 16 | 9 | | 25 | 34 | 19 | | 53 |
| Extended (128) | 4 | | 4 | 2 | 7 | | 7 | 13 | | | 13 | 26 | | | 26 |
| Short to Long (32 to 64) | 1 | 1 | 2 | 1 | 2 | 1 | 3 | 3 | 2 | | 5 | 7 | 4 | | 11 |
| Long to Extended (64 to 128) | 1 | 1 | 2 | | 1 | 1 | 2 | 2 | 2 | | 4 | 4 | 4 | | 8 |
| Short to Extended (32 to 128) | | | | | 2 | 1 | 3 | 2 | 1 | | 3 | 4 | 2 | | 6 |
| Long to Short (64 to 32) | 1 | | 1 | 1 | 1 | | 1 | 2 | | | 2 | 5 | | | 5 |
| Extended to Long (128 to 64) | 1 | | 1 | | | | | 1 | | | 1 | 2 | | | 2 |
| Extended to Short (128 to 32) | | | | | 2 | | 2 | 2 | | | 2 | 4 | | | 4 |
| Other | | | | | | | | | | 5 | 5 | | | 5 | 5 |
| Totals | 35 | 19 | 54 | 8 | 20 | 6 | 26 | 59 | 23 | 5 | 87 | 122 | 48 | 5 | 175 |

**Explanation:**

FX Floating-point-support-extensions facility.
HF HFP-extensions facility.
R-R Register-and-register operation.
R-S Register-and-storage operation.

*Figure 9-14. Summary of All Floating-Point Instructions*


LzLabs GmbH. Ex. 1008-479

LzLabs GmbH. Ex. 1008-480

# Chapter 10. Control Instructions


BRANCH AND SET AUTHORITY . . . . . 10-7
BRANCH AND STACK . . . . . 10-10
BRANCH IN SUBSPACE GROUP . . . . 10-13
COMPARE AND SWAP AND PURGE . 10-17
DIAGNOSE . . . . . 10-19
EXTRACT PRIMARY ASN . . . . . 10-19
EXTRACT SECONDARY ASN . . . . . 10-20
EXTRACT STACKED REGISTERS . . . 10-20
EXTRACT STACKED STATE . . . . . 10-22
INSERT ADDRESS SPACE CONTROL . 10-23
INSERT PSW KEY . . . . . 10-24
INSERT STORAGE KEY EXTENDED . 10-25
INSERT VIRTUAL STORAGE KEY . . . 10-25
INVALIDATE PAGE TABLE ENTRY . . . 10-26
LOAD ADDRESS SPACE PARAMETERS . . . . . 10-28
LOAD CONTROL . . . . . 10-37
LOAD PSW . . . . . 10-37
LOAD REAL ADDRESS . . . . . 10-38
LOAD USING REAL ADDRESS . . . . . 10-40
MODIFY STACKED STATE . . . . . 10-40
MOVE PAGE (Facility 2) . . . . . 10-42
MOVE TO PRIMARY . . . . . 10-45
MOVE TO SECONDARY . . . . . 10-45
MOVE WITH DESTINATION KEY . . . . 10-47
MOVE WITH KEY . . . . . 10-47
MOVE WITH SOURCE KEY . . . . . 10-48
PAGE IN . . . . . 10-50
PAGE OUT . . . . . 10-51
PROGRAM CALL . . . . . 10-52
PROGRAM CALL FAST . . . . . 10-63
PROGRAM RETURN . . . . . 10-67
PROGRAM TRANSFER . . . . . 10-70
PURGE ALB . . . . . 10-76
PURGE TLB . . . . . 10-76
RESET REFERENCE BIT EXTENDED . 10-76
RESUME PROGRAM . . . . . 10-77
SET ADDRESS SPACE CONTROL . . . 10-79
SET ADDRESS SPACE CONTROL FAST . . . . . 10-79
SET CLOCK . . . . . 10-81
SET CLOCK COMPARATOR . . . . . 10-82
SET CLOCK PROGRAMMABLE FIELD . 10-82
SET CPU TIMER . . . . . 10-82
SET PREFIX . . . . . 10-83
SET PSW KEY FROM ADDRESS . . . . 10-83
SET SECONDARY ASN . . . . . 10-84
SET STORAGE KEY EXTENDED . . . . 10-87
SET SYSTEM MASK . . . . . 10-87
SIGNAL PROCESSOR . . . . . 10-88
STORE CLOCK COMPARATOR . . . . 10-89
STORE CONTROL . . . . . 10-89
STORE CPU ADDRESS . . . . . 10-90
STORE CPU ID . . . . . 10-90
STORE CPU TIMER . . . . . 10-91
STORE FACILITY LIST . . . . . 10-91
STORE PREFIX . . . . . 10-92
STORE SYSTEM INFORMATION . . . . 10-92
STORE THEN AND SYSTEM MASK . 10-103
STORE THEN OR SYSTEM MASK . . 10-103
STORE USING REAL ADDRESS . . . 10-104
TEST ACCESS . . . . . 10-104
TEST BLOCK . . . . . 10-107
TEST PROTECTION . . . . . 10-109
TRACE . . . . . 10-111
TRAP . . . . . 10-112


This chapter includes all privileged and semiprivileged instructions described in this publication, except the input/output instructions, which are described in Chapter 14, “I/O Instructions.”

Privileged instructions may be executed only when the CPU is in the supervisor state. An attempt to execute a privileged instruction in the problem state generates a privileged-operation exception.

The semiprivileged instructions are those instructions that can be executed in the problem state when certain authority requirements are met. An attempt to execute a semiprivileged instruction in the problem state when the authority requirements are not met generates a privileged-operation exception or some other program-interruption condition depending on the particular requirement which is violated. Those requirements which cause a privileged-operation exception to be generated in the problem state are not enforced when execution is attempted in the supervisor state.

The control instructions and their mnemonics, formats, and operation codes are listed in Figure 10-1 on page 10-3. The figure also indicates when the condition code is set, the instruction fields that designate access registers, and the

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-481

exceptional conditions in operand designations, data, or results that cause a program interruption.

For those control instructions which have special rules regarding the handling of exceptional situations, a section called “Special Conditions” is included. This section indicates the type of ending (suppression, nullification, or completion) only for those exceptions for which the ending may vary.

**Note:** In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For LOAD PSW, for example, LPSW is the mnemonic and $D_2(B_2)$ the operand designation.

**Programming Notes:**

1. The following additional control instructions are available in ESA/370 and ESA/390 as compared to 370-XA:
   - BRANCH AND STACK
   - COMPARE AND SWAP AND PURGE
   - EXTRACT STACKED REGISTERS
   - EXTRACT STACKED STATE
   - LOAD USING REAL ADDRESS
   - MODIFY STACKED STATE
   - MOVE PAGE
   - MOVE WITH DESTINATION KEY
   - MOVE WITH SOURCE KEY
   - PROGRAM RETURN
   - PURGE ALB
   - STORE USING REAL ADDRESS
   - TEST ACCESS

   The function of the MOVE PAGE instruction is expanded when the move-page facility 2 is installed.
2. The control instruction BRANCH IN SUBSPACE GROUP is available when the subspace-group facility is installed.
3. The control instruction BRANCH AND SET AUTHORITY is available when the branch-and-set-authority facility is installed.
4. The control instructions PROGRAM CALL FAST, RESUME PROGRAM, and TRAP are available when the program-call-fast, resume-program, and trap facilities, respectively, are installed.
5. The control instructions SET CLOCK PROGRAMMABLE MODE, and STORE SYSTEM INFORMATION are available when the extended-TOD-clock and store-system-information facilities, respectively, are installed.
6. The control instruction STORE FACILITY LIST is available in the ESA/390 architectural mode when the z/Architecture architectural mode is installed.



LzLabs GmbH. Ex. 1008-482

| Name | Mne-monic | Characteristics | | | | | Op Code |
|---|---|---|---|---|---|---|---|
| BRANCH AND SET AUTHORITY | BSA | RRE BS | Q A[1] | SO T | B R | | B25A |
| BRANCH AND STACK | BAKR | RRE | A[1] | SF T | B ST | | B240 |
| BRANCH IN SUBSPACE GROUP | BSG | RRE SG | A[1] | SO T | B R | R[2] | B258 |
| COMPARE AND SWAP AND PURGE | CSP | RRE C | P A[1] SP | $ | R ST | R[2] | B250 |
| DIAGNOSE | | DM | P DM | | | MD | 83 |
| EXTRACT PRIMARY ASN | EPAR | RRE | Q | SO | R | | B226 |
| EXTRACT SECONDARY ASN | ESAR | RRE | Q | SO | R | | B227 |
| EXTRACT STACKED REGISTERS | EREG | RRE | A[1] | SE | R | U[1] U[2] | B249 |
| EXTRACT STACKED STATE | ESTA | RRE C | A[1] SP | SE | R | | B24A |
| INSERT ADDRESS SPACE CONTROL | IAC | RRE C | Q | SO | R | | B224 |
| INSERT PSW KEY | IPK | S | Q | G2 | R | | B20B |
| INSERT STORAGE KEY EXTENDED | ISKE | RRE | P A[1] | | | | B229 |
| INSERT VIRTUAL STORAGE KEY | IVSK | RRE | Q A[1] | SO | R | R[2] | B223 |
| INVALIDATE PAGE TABLE ENTRY | IPTE | RRE | P A[1] | $ | | | B221 |
| LOAD ADDRESS SPACE PARAMETERS | LASP | SSE C | P A[1] SP | AS | | B[1] | E500 |
| LOAD CONTROL | LCTL | RS | P A SP | | | B[2] | B7 |
| LOAD PSW | LPSW | S L | P A SP | ¢ | | B[2] | 82 |
| LOAD REAL ADDRESS | LRA | RX C | P A[1] | AT | R | BP | B1 |
| LOAD USING REAL ADDRESS | LURA | RRE | P A[1] SP | | R | | B24B |
| MODIFY STACKED STATE | MSTA | RRE | A[1] SP | SE | ST | | B247 |
| MOVE PAGE (facility 2) | MVPG | RRE C M2 | Q A[1] SP | G0 | ST | R[1] R[2] | B254 |
| MOVE TO PRIMARY | MVCP | SS C | Q A | SO ¢ | ST | | DA |
| MOVE TO SECONDARY | MVCS | SS C | Q A | SO ¢ | ST | | DB |
| MOVE WITH DESTINATION KEY | MVCDK | SSE | Q A | GM | ST | B[1] B[2] | E50F |
| MOVE WITH KEY | MVCK | SS C | Q A | | ST | B[1] B[2] | D9 |
| MOVE WITH SOURCE KEY | MVCSK | SSE | Q A | GM | ST | B[1] B[2] | E50E |
| PAGE IN | PGIN | RRE C ES | P A[1] | ¢ | | | B22E |
| PAGE OUT | PGOUT | RRE C ES | P A[1] | ¢ | | | B22F |
| PROGRAM CALL | PC | S | Q A[1] | Z[1] T ¢ GM | B R ST | | B218 |
| PROGRAM CALL FAST | PCF | S PC | A[1] | Z[5] ¢ G4 | B R ST | | B218 |
| PROGRAM RETURN | PR | E U | A[1] SP | Z[4] T ¢[2] | B R ST | | 0101 |
| PROGRAM TRANSFER | PT | RRE | Q A[1] SP | Z[2] T ¢ | B | | B228 |
| PURGE ALB | PALB | RRE | P | $ | | | B248 |
| PURGE TLB | PTLB | S | P | $ | | | B20D |
| RESET REFERENCE BIT EXTENDED | RRBE | RRE C | P A[1] | | | | B22A |

*Figure 10-1 (Part 1 of 4). Summary of Control Instructions*


LzLabs GmbH. Ex. 1008-483

| Name | Mne-monic | Characteristics | | | | | Op Code |
|---|---|---|---|---|---|---|---|
| RESUME PROGRAM | RP | S L RP | Q A SP | SW T | B R | $B_2$ | B277 |
| SET ADDRESS SPACE CONTROL | SAC | S | Q SP | SW ¢ | | | B219 |
| SET ADDRESS SPACE CONTROL FAST | SACF | S SA | Q SP | SW | | | B279 |
| SET CLOCK | SCK | S C | P A SP | | | $B_2$ | B204 |
| SET CLOCK COMPARATOR | SCKC | S | P A SP | | | $B_2$ | B206 |
| SET CLOCK PROGRAMMABLE FIELD | SCKPF | E EK | P SP | G0 | | | 0107 |
| SET CPU TIMER | SPT | S | P A SP | | | $B_2$ | B208 |
| SET PREFIX | SPX | S | P A SP | $ | | $B_2$ | B210 |
| SET PSW KEY FROM ADDRESS | SPKA | S | Q | | | | B20A |
| SET SECONDARY ASN | SSAR | RRE | A[1] | $Z^3$ T ¢ | | | B225 |
| SET STORAGE KEY EXTENDED | SSKE | RRE | P A[1] | ¢ | | | B22B |
| SET SYSTEM MASK | SSM | S | P A SP | SO | | $B_2$ | 80 |
| SIGNAL PROCESSOR | SIGP | RS C | P | $ | R | | AE |
| STORE CLOCK COMPARATOR | STCKC | S | P A SP | | ST | $B_2$ | B207 |
| STORE CONTROL | STCTL | RS | P A SP | | ST | $B_2$ | B6 |
| STORE CPU ADDRESS | STAP | S | P A SP | | ST | $B_2$ | B212 |
| STORE CPU ID | STIDP | S | P A SP | | ST | $B_2$ | B202 |
| STORE CPU TIMER | STPT | S | P A SP | | ST | $B_2$ | B209 |
| STORE FACILITY LIST | STFL | S N3 | P | | | | B2B1 |
| STORE PREFIX | STPX | S | P A SP | | ST | $B_2$ | B211 |
| STORE SYSTEM INFORMATION | STSI | S C SN | P A SP | GM | R ST | $B_2$ | B27D |
| STORE THEN AND SYSTEM MASK | STNSM | SI | P A | | ST | $B_1$ | AC |
| STORE THEN OR SYSTEM MASK | STOSM | SI | P A SP | | ST | $B_1$ | AD |
| STORE USING REAL ADDRESS | STURA | RRE | P A[1] SP | | SU | | B246 |
| TEST ACCESS | TAR | RRE C | A[1] | AS | | $U_1$ | B24C |
| TEST BLOCK | TB | RRE C | P A[1] | II $ G0 | R | | B22C |
| TEST PROTECTION | TPROT | SSE C | P A[1] | | | $B_1$ | E501 |
| TRACE | TRACE | RS | P A SP | T ¢ | | $B_2$ | 99 |
| TRAP | TRAP2 | E TR | A | SO T | B R ST | | 01FF |
| TRAP | TRAP4 | S TR | A | SO T | B R ST | | B2FF |

*Figure 10-1 (Part 2 of 4). Summary of Control Instructions*



LzLabs GmbH. Ex. 1008-484

**Explanation:**

| | |
|---|---|
| ¢ | Causes serialization and checkpoint synchronization. |
| ¢² | Causes serialization and checkpoint synchronization when the state entry to be unstacked is a program-call state entry. |
| $ | Causes serialization. |
| A | Access exceptions for logical addresses. |
| A¹ | Access exceptions; not all access exceptions may occur; see instruction description for details. |
| AS | ASN-translation-specification and special-operation exceptions. |
| AT | ASN-translation-specification exception. |
| B | PER branch event. |
| B₁ | B₁ field designates an access register in the access-register mode. |
| B₂ | B₂ field designates an access register in the access-register mode. |
| BP | B₂ field designates an access register when PSW bits 16 and 17 have the value 01. |
| BS | Branch-and-set-authority facility |
| C | Condition code is set. |
| DM | Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code. |
| E | E instruction format. |
| ES | Expanded-storage facility. |
| G0 | Instruction execution includes the implied use of general register 0. |
| G2 | Instruction execution includes the implied use of general register 2. |
| G4 | Instruction execution includes the implied use of general register 4. |
| GM | Instruction execution includes the implied use of multiple general registers:<br>General registers 0 and 1 for MOVE WITH DESTINATION KEY, MOVE WITH SOURCE KEY, and STORE SYSTEM INFORMATION.<br>General registers 3, 4, and 14 for PROGRAM CALL. |
| II | Interruptible instruction. |
| L | New condition code is loaded. |
| MD | Designation of access registers in the access-register mode is model-dependent. |
| M2 | Move-page facility 2. |
| N3 | Instruction is added to ESA/390 from z/Architecture. |
| P | Privileged-operation exception. |
| PC | Program-call-fast facility. |
| Q | Privileged-operation exception for semiprivileged instructions. |
| R | PER general-register-alteration event. |
| R₁ | R₁ field designates an access register in the access-register mode. |
| R₂ | R₂ field designates an access register in the access-register mode. |
| RP | Resume-program facility. |
| RRE | RRE instruction format. |

*Figure 10-1 (Part 3 of 4). Summary of Control Instructions*


LzLabs GmbH. Ex. 1008-485

**Explanation** (Continued):

RS RS instruction format.
RX RX instruction format.
S S instruction format.
SA Set-address-space-control-fast facility.
SE Special-operation, stack-empty, stack-specification, and stack-type exceptions.
SF Special-operation, stack-full, and stack-specification exceptions.
SG Subspace-group facility.
SI SI instruction format.
SN Store-system-information facility.
SO Special-operation exception.
SP Specification exception.
SS SS instruction format.
SSE SSE instruction format.
ST PER storage-alteration event.
SU PER store-using-real-address event.
SW Special-operation exception and space-switch event.
T Trace exceptions (which include trace table, addressing, and low-address protection).
TR Trap facility
U Condition code is unpredictable.
$U_1$ $R_1$ field designates an access register unconditionally.
$U_2$ $R_2$ field designates an access register unconditionally.
$Z^1$ Additional exceptions and events for PROGRAM CALL (which include AFX-translation, ASN-translation-specification, ASX-translation, EX-translation, LX-translation, PC-translation-specification, special-operation, stack-full, stack-specification, and subspace-replacement exceptions and space-switch event).
$Z^2$ Additional exceptions and events for PROGRAM TRANSFER (which include AFX-translation, ASN-translation-specification, ASX-translation, primary-authority, special-operation, and subspace-replacement exceptions and space-switch event).
$Z^3$ Additional exceptions for SET SECONDARY ASN (which include AFX translation, ASN-translation specification, ASX translation, secondary authority, special operation, and subspace replacement).
$Z^4$ Additional exceptions and events for PROGRAM RETURN (which include AFX-translation, ASN-translation-specification, ASX-translation, secondary-authority, special-operation, stack-empty, stack-operation, stack-specification, stack-type, and subspace-replacement exceptions and space-switch event).
$Z^5$ Additional exceptions and events for PROGRAM CALL FAST (which include EX-translation, special-operation, stack-full, stack-specification, and subspace-replacement exceptions and space-switch event).

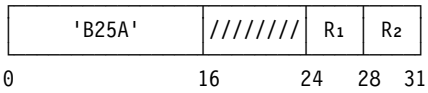
*Figure 10-1 (Part 4 of 4). Summary of Control Instructions*



LzLabs GmbH. Ex. 1008-486

## BRANCH AND SET AUTHORITY

BSA R1,R2 [RRE]

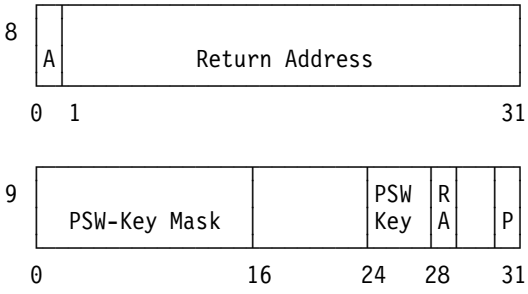



If the dispatchable unit is in the base-authority state: bits 32-63 of the current PSW, including the updated instruction address, are saved in the dispatchable-unit control table (DUCT); the PSW-key mask (PKM), PSW key, and problem-state bit also are saved in the DUCT; the PKM and PSW key are replaced using the contents of general register $R_1$; the problem-state bit is set to one; bits 32-63 of the PSW are replaced using the contents of general register $R_2$; and the dispatchable unit is placed in the reduced-authority state.

If the dispatchable unit is in the reduced-authority state: bits 32-63 of the current PSW, including the updated instruction address, are saved in general register $R_1$ if $R_1$ is not zero; bits 32-63 of the PSW and the PKM, PSW key, and problem-state bit are replaced by values saved in the DUCT; and the dispatchable unit is placed in the base-authority state.

Bits 16-23 of the instruction are ignored.

Words 8 and 9 of the DUCT are used by this instruction. The contents of those words are as follows:




The fields in words 8 and 9 of the DUCT are allocated as follows:

***Addressing Mode (A):*** Bit position 0 of word 8 contains the addressing-mode bit, bit 32 of the PSW, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The addressing-mode bit is restored to the PSW from the DUCT by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

***Return Address:*** Bit positions 1-31 of word 8 contain the updated instruction address, bits 33-63 of the PSW, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The return address is restored to the PSW (it is treated as the branch address) by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

***PSW-Key Mask:*** Bit positions 0-15 of word 9 contain the PSW-key mask (PKM), bits 0-15 of control register 3, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The PKM is restored to control register 3 by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

***PSW Key:*** Bit positions 24-27 of word 9 contain the PSW key, bits 8-11 of the PSW, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The PSW key is restored to the PSW by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

***Reduced Authority (RA):*** Bit 28 of word 9 indicates, when zero, that the dispatchable unit associated with the DUCT is in the base-authority state or, when one, that the dispatchable unit is in the reduced-authority state. Bit 28 is set to one by BRANCH AND SET AUTHORITY executed in the base-authority state, and it is set to zero by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

***Problem State (P):*** Bit position 31 of word 9 contains the problem-state bit, bit 15 of the PSW, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The problem-state bit is restored to the PSW by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

Bits 16-23, 29, and 30 of word 9 are set to zeros when saving occurs in words 8 and 9 in the base-authority state. Word 8 and bits 0-27 and 29-31 of word 9 remain unchanged when bit 28 of word 9 is set to zero in the reduced-authority state.

The fetch, store, and update references to the DUCT are single-access references and appear to

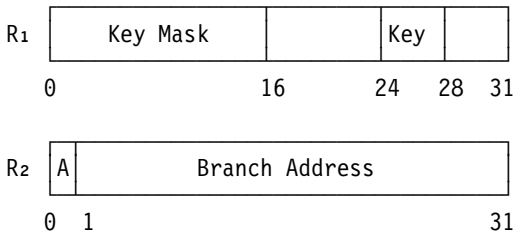

LzLabs GmbH. Ex. 1008-487

be word concurrent as observed by other CPUs. The words of the DUCT are accessed in no particular order.

**Base-Authority Operation**

When BRANCH AND SET AUTHORITY is executed in the base-authority state, as indicated by the reduced-authority bit (RA) in the DUCT being zero, $R_2$ must be nonzero; otherwise, a special-operation exception is recognized. $R_1$ may be zero or nonzero.

The contents of general registers $R_1$ and $R_2$ when the execution of the instruction begins in the base-authority state are as follows:




PSW bits 32-63, the PKM, the PSW key, and the problem-state bit are saved in the DUCT, the RA bit in the DUCT is set to one, and bits 16-23, 29, and 30 of word 9 of the DUCT are set to zeros.

Bits 24-27 of general register $R_1$ are placed in bit positions 8-11 of the PSW as the new PSW key. In the problem state, the new PSW key must be authorized by the PKM; otherwise, if the new PSW key is not authorized, a privileged-operation exception is recognized.

After the new PSW key has been placed in the PSW, bits 0-15 of general register $R_1$ are ANDed with the PKM in control register 3, and the result replaces the PKM in control register 3.

The problem-state bit in the PSW is set to one.

Bit 0 of general register $R_2$ is placed in bit position 32 of the PSW as the new addressing-mode bit. A branch address is generated from bits 1-31 of general register $R_2$ under the control of the new addressing mode, and the result is placed in bit positions 33-63 of the PSW as the new instruction address.

Bits 16-23 and 28-31 of general register $R_1$ may be used for future extensions and should be zeros; otherwise, the program may not operate compatibly in the future.

**Reduced-Authority Operation**

When BRANCH AND SET AUTHORITY is executed in the reduced-authority state, as indicated by the reduced-authority bit (RA) in the DUCT being one, $R_2$ must be zero; otherwise, a special-operation exception is recognized. $R_1$ may be zero or nonzero. The initial contents of general registers $R_1$ and $R_2$ are ignored.

If $R_1$ is nonzero, bits 32-63 of the current PSW, including the addressing-mode bit and the updated instruction address, are placed in general register $R_1$. If $R_1$ is zero, general register 0 remains unchanged.

PSW bits 32-63, the PKM, the PSW key, and the problem-state bit are restored from the DUCT, and the RA bit in the DUCT is set to zero. There is no test for whether the restored PSW key is authorized by the restored PKM.

**Special Conditions**

The instruction can be executed successfully only when the address-space-function control, bit 15 of control register 0, is one. In addition, $R_2$ must be nonzero in the base-authority state and zero in the reduced-authority state. If any of these rules is violated, a special-operation exception is recognized, and the operation is suppressed.

In the problem state, the execution of the instruction in the base-authority state is subject to control by the PSW-key mask in control register 3. When the bit in the PSW-key mask corresponding to the PSW-key value to be set is one, the instruction is executed successfully. When the selected bit in the PSW-key mask is zero, a privileged-operation exception is recognized. In the supervisor state, any value for the PSW key is valid.

Key-controlled protection does not apply to any access made during the operation. Low-address protection does apply.

The contents of word 8 of the DUCT are not checked for validity before they are loaded into the PSW. However, after loading, a specification exception is recognized, and a program interruption occurs, when the newly loaded PSW con-



LzLabs GmbH. Ex. 1008-488

tains a zero in bit position 32 and the contents of bit positions 33-39 are not all zeros. In this case, the operation is completed, and the resulting instruction-length code is 0. The specification exception, which in this case is listed as a program exception in this instruction, is described in “Early Exception Recognition” on page 6-9. It may be considered as occurring early in the process of preparing to execute the following instruction.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-2 on page 10-10.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (dispatchable-unit control table)
- Operation (if the branch-and-set-authority facility is not installed)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state, base-authority operation only)
- Protection (low-address; dispatchable-unit control table)
- Special operation
- Specification
- Trace

**Programming Notes:**

1. BRANCH AND SET AUTHORITY can improve performance by replacing to-current-primary forms of PROGRAM TRANSFER (PT-cp) and basic (nonstacking) PROGRAM CALL (PC-cp) instructions. PT-cp and PC-cp are often used (within a single address space) to reduce the authority of the PSW-key mask (PKM) or change from supervisor state to problem state during a calling linkage made by PT-cp and then to restore the PKM authority or supervisor state during a return linkage made by PC-cp. Also, the PSW-key-setting operations of BRANCH AND SET AUTHORITY can be substituted for SET PSW KEY FROM ADDRESS instructions, and, since BRANCH AND SET AUTHORITY combines branching with PSW-key setting, it can be used to change the PSW key when branching from or to a fetch-protected program.
2. Only one base-authority state and one reduced-authority state are available to a dispatchable unit. Nested use of BRANCH AND SET AUTHORITY, that is, use within different subroutine levels, is not possible. The requirement that $R_2$ must be nonzero in the base-authority state and zero in the reduced-authority state provides detection of an attempt to use BRANCH AND SET AUTHORITY in the base-authority state when the dispatchable unit is already in the reduced-authority state because of a previous use of the instruction in the base-authority state.
3. The instruction may be referred to as BSA-ba or BSA-ra depending on whether it is executed in the base-authority state or the reduced-authority state, respectively.

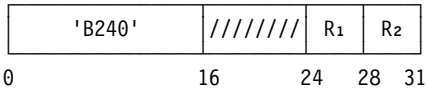

LzLabs GmbH. Ex. 1008-489

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B.1 | Operation exception if the branch-and-set-authority facility is not installed. |
| 7.B.2 | Special-operation exception due to the address-space-function control, bit 15 of control register 0, being zero. |
| 8.A | Trace exceptions. |
| 8.B | Protection exception (low-address protection) for access to dispatchable-unit control table. |
| 8.C.1 | Addressing exception for access to dispatchable-unit control table. |
| 8.C.2 | Special-operation exception due to $R_2$ being zero in the base-authority state or $R_2$ being nonzero in the reduced-authority state. |
| 8.C.3 | Privileged-operation exception due to selected PSW-key-mask bit being zero (base-authority operation only). |
| 9. | Specification exception due to bit 32 of the newly loaded PSW zero when bits 33-39 are not all zeros (reduced-authority operation only). |

*Figure 10-2. Priority of Execution: BRANCH AND SET AUTHORITY*

## BRANCH AND STACK

BAKR $R_1$,$R_2$ [RRE]




A linkage-stack branch state entry is formed, and the current PSW, except with an unpredictable PER mask and an addressing-mode bit and instruction address from the first operand substituted for bits 32-63, is placed in the state entry. Subsequently, the updated instruction address in the current PSW is replaced from the second operand. The new value of PSW bits 32-63 and the PSW-key mask, PASN, SASN, EAX, and contents of general registers 0-15 and access registers 0-15 also are placed in the state entry. The action associated with an operand is not performed if the R field designating the operand is zero.

Bits 16-23 of the instruction are ignored.

When the $R_1$ field is nonzero, the contents of general register $R_1$ specify an address referred to as the return address. The return address is generated from the contents of the register under the control of the addressing mode specified by bit 0 of the register: 24-bit mode if bit 0 is zero, or 31-bit mode if bit 0 is one. Bit 0 of the register and the return address are substituted for the addressing-mode bit and the updated instruction address, respectively, in the current PSW when the contents of that PSW are placed in the state entry. The contents of the current PSW are not changed.

When the $R_1$ field is zero, there is no substitution for the addressing-mode bit and instruction address in the current PSW when that PSW is placed in the state entry.

Subsequently, when the $R_2$ field is nonzero, the instruction address in the current PSW is replaced by the branch address. The branch address is generated from the contents of general register $R_2$ under the control of the current addressing mode.



LzLabs GmbH. Ex. 1008-490

When the $R_2$ field is zero, the operation is performed without branching.

The branch state entry is formed and information is placed in it as described in “Stacking Process” on page 5-73. The entry-type code in the state entry is 0000100 binary.

Key-controlled protection does not apply to accesses to the linkage stack, but low-address and page protection do apply.

**Special Conditions**

The CPU must be in the primary-space mode or access-register mode, and the address-space-function control, bit 15 of control register 0 must be one; otherwise, a special-operation exception is recognized.

A stack-full or stack-specification exception may be recognized during the stacking process.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-3 on page 10-12.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch or store, except for key-controlled protection, linkage-stack entry)
- Special operation
- Stack full
- Stack specification
- Trace


LzLabs GmbH. Ex. 1008-491

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to DAT being off, the CPU being in secondary-space mode or home-space mode, or the address-space-function control, bit 15 of control register 0, being zero. |
| 8.A | Trace exceptions (only if $R_2$ is nonzero). |
| 8.B.1 | Access exceptions (fetch) for entry descriptor of the current linkage-stack entry.<br><br>Note: Exceptions 8.B.2-8.B.7 can occur only if there is not enough remaining free space in the current linkage-stack section. |
| 8.B.2 | Stack-specification exception due to remaining-free-space value in current linkage-stack entry not being a multiple of 8. |
| 8.B.3 | Access exceptions (fetch) for second word of the trailer entry of the current section. The entry is presumed to be a trailer entry; its entry-type field is not examined. |
| 8.B.4 | Stack-full exception due to forward-section validity bit in the trailer entry being zero. |
| 8.B.5 | Access exceptions (fetch) for entry descriptor of the header entry of the next section. This entry is presumed to be a header entry; its entry-type field is not examined. |
| 8.B.6 | Stack-specification exception due to not enough remaining free space in the next section. |
| 8.B.7 | Access exceptions (store) for second word of the header entry of the next section. If there is no exception, the header is now called the current entry. |
| 8.B.8 | Access exceptions (store) for entry descriptor of the current entry and for the new state entry. |

*Figure 10-3. Priority of Execution: BRANCH AND STACK*

**Programming Notes:**

1. Examples of the use of the BRANCH AND STACK instruction are given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. In no case does BRANCH AND STACK change the current addressing mode.
3. The effect when the $R_1$ field is zero is that the return address, which would otherwise be specified by the $R_1$ general register, is the address of the next sequential instruction. In this case, BRANCH AND STACK provides a program-linkage function that is comparable to the function of BRANCH AND SAVE.
4. BRANCH AND STACK with a nonzero $R_1$ field is intended for use at or near the entry point of a called program. The program may be called by means of BRANCH AND LINK (BALR), BRANCH AND SAVE (BAS or BASR), or BRANCH AND SAVE AND SET MODE, or by means of a BRANCH AND SET MODE instruction located in a “glue module.” In all of these cases when the nonzero $R_1$ field of the calling instruction is the same as the $R_1$ field of BRANCH AND STACK, and

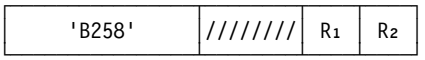


LzLabs GmbH. Ex. 1008-492

even when the addressing mode was changed during the calling linkage, BRANCH AND STACK correctly saves the addressing mode and 24-bit or 31-bit return address of the calling program so that the subsequent execution of PROGRAM RETURN will return correctly to the calling program.

## BRANCH IN SUBSPACE GROUP

```
BSG   R1,R2              [RRE]
```




Provided that the current primary address space is in the subspace group, if any, associated with the current dispatchable unit, the access-list-entry token (ALET) in access register $R_2$ is translated by means of a special form of access-register translation (ART) to locate a destination ASN-second-table entry (DASTE). If the DASTE specifies the base space of the subspace group, the primary segment-table designation (PSTD) in control register 1 is replaced by the STD in the DASTE. If the DASTE specifies a subspace of the group, bits 1-23 and 25-31 of the PSTD are replaced by the same bits of the STD in the DASTE. In either case, the following actions also occur. Bits 32-63 of the current PSW, including the updated instruction address, are saved in general register $R_1$. Subsequently, the addressing-mode bit and instruction address in the current PSW are replaced from general register $R_2$; the secondary STD (SSTD) in control register 7 is set equal to the new PSTD in control register 1; and the secondary ASN (SASN), bits 16-31 of control register 3, is set equal to the primary ASN (PASN), bits 16-31 of control register 4. General register 0 remains unchanged if the $R_1$ field is zero.

Bits 16-23 of the instruction are ignored.

The current primary address space is in the subspace group for the dispatchable unit if the current primary-ASTE origin (PASTEO), bits 1-25 of control register 5, designates the ASTE for the base space of the group. The PASTEO designates the base-space ASTE if the PASTEO is equal to the base-ASTE origin (BASTEO), bits 1-25 of word 0 of the dispatchable-unit control table (DUCT). For determining whether the PASTEO equals the BASTEO, either the PASTEO may be compared to the BASTEO or the entire contents of control register 5 may be compared to the entire contents of word 0 of the DUCT.

Ordinary ART is described in “Access-Register-Translation Process” on page 5-48. The special ART performed by this instruction is contrasted to ordinary ART as follows:

1. The special ART is performed regardless of whether the CPU is in the access-register mode.
2. If the ALET being translated is 00000000 hex, called ALET 0, the DASTE is the ASTE for the base space. Bit 0 of the DASTE is ignored.
3. If the ALET is 00000001 hex, called ALET 1, the DASTE is the ASTE for the last subspace entered by the dispatchable unit by means of BRANCH IN SUBSPACE GROUP. That ASTE is designated by the subspace-ASTE origin (SSASTEO), bits 1-25 of word 1 of the DUCT. A special-operation exception is recognized if a subspace has not previously been entered, as indicated by that the SSASTEO is all zeros. An ASTE-validity exception is recognized if bit 0 of the DASTE is one. An ASTE-sequence exception is recognized if the ASTE sequence number (ASTESN) in the DASTE does not equal the subspace ASTESN (SSASTESN) in word 3 of the DUCT. The DASTE located because of ALET 1 is considered to specify a subspace even if, due to an error, the DASTE is the ASTE for the base space. That is, there is no comparison of the SSASTEO to the BASTEO.
4. If the ALET is other than ALET 0 and ALET 1, an ASTE is located by obtaining its origin from an access-list entry (ALE) in a way similar to ordinary ART, and the DASTE is that located ASTE. In this case, as in ordinary ART:
   - An ALET-specification exception is recognized if bits 0-6 of the ALET are not zeros.
   - An ALEN-translation exception is recognized if the ALE is outside the effective access list or bit 0 of the ALE is one.
   - An ASTE-validity exception is recognized if bit 0 of the DASTE is one.
   - An ASTE-sequence exception is recognized if the ASTE sequence number


LzLabs GmbH. Ex. 1008-493

(ASTESN) in the DASTE does not equal the ASTESN in the ALE.

The operation differs from ordinary ART in that the ALE sequence number (ALESN) in the ALE is not compared to the ALESN in the ALET, and the private bit in the ALE is treated as zero. Thus, ALE-sequence and extended-authority exceptions cannot occur.

The fetch-only bit in the ALE is ignored.

When the ALET is other than ALET 0 and ALET 1, the special ART may be performed by using the ART-lookaside buffer (ALB).

The DASTE located due to an ALET other than ALET 0 and ALET 1 may be the ASTE for the base space of the subspace group associated with the dispatchable unit. The DASTE is the base-space ASTE if the DASTE origin (DASTEO) obtained from an ALE by ART equals the BASTEO in the DUCT. For determining whether the DASTEO equals the BASTEO, either the DASTEO may be compared to the BASTEO, or the DASTEO with one leftmost and six rightmost zeros appended may be compared to the entire contents of word 0 of the DUCT. If the DASTE is not the base-space ASTE, the DASTE is treated as the ASTE for a subspace of the dispatchable unit's subspace group provided that (1) the subspace-group bit, bit 22, in the STD in the DASTE is one, and (2) the DASTE does not specify the base space of another subspace group. The DASTE specifies the base space of another subspace group if the base-space bit, bit 31 of word 0 of the DASTE, is one. A special-operation exception is recognized if either of those two provisions is not met.

If the DASTE specifies the base space of the subspace group, the PSTD in control register 1 is replaced by the STD in the DASTE. If the DASTE specifies a subspace, bits 1-23 and 25-31 of the PSTD are replaced by the same bits of the STD in the DASTE, and bit 0 of the PSTD, the space-switch-event-control bit, and bit 24 of the PSTD, the storage-alteration-event bit, remain unchanged.

If $R_1$ is nonzero, bits 32-63 of the current PSW, including the updated instruction address, are placed in general register $R_1$. If $R_1$ is zero, general register 0 remains unchanged.

Whether $R_2$ is nonzero or zero, the contents of general register $R_2$ specify the new addressing mode and designate the branch address. Bit 0 of the register specifies the new addressing mode and replaces bit 32 of the current PSW, and the branch address is generated from the contents of the register under the control of the new addressing mode. The new value for the PSW is computed before general register $R_1$ is changed.

The secondary STD (SSTD) in control register 7 is set equal to the new PSTD in control register 1. The secondary ASN (SASN), bits 16-31 of control register 3, is set equal to the primary ASN (PASN), bits 16-31 of control register 4.

If the DASTE specifies the base space, the subspace-active bit, bit 0 of word 1 of the DUCT, is set to zero, and bits 1-31 of word 1 remain unchanged. If the DASTE specifies a subspace by means of ALET 1, then (1) the subspace-active bit is set to one, (2) the SSASTEO in bit positions 1-25 of word 1 remains unchanged, and (3) bits 26-31 of word 1 either are set to zeros or remain unchanged. If the DASTE specifies a subspace by means of an ALET other than ALET 1, then (1) the subspace-active bit is set to one, (2) the DASTEO is stored in bit positions 1-25 of word 1 as the SSASTEO, (3) zeros are stored in bit positions 26-31 of word 1, and (4) the ASTESN in the DASTE is stored in word 3 of the DUCT as the SSASTESN.

The fetch, store, and update references to the DUCT are single-access references and appear to be word concurrent as observed by other CPUs. The words of the DUCT are accessed in no particular order.

The operation, since it changes a translation parameter in control register 1, causes all copies of prefetched instructions to be discarded, except when in the home-space mode.

**Special Conditions**

The address-space-function control, bit 15 of control register 0, must be one, and DAT must be on; otherwise, a special-operation exception is recognized. A special-operation exception is also recognized if the current primary address space is not in a subspace group associated with the current dispatchable unit, if the ALET in access register $R_2$ is ALET 1 but a subspace has not pre-



LzLabs GmbH. Ex. 1008-494

viously been entered by the dispatchable unit by means of BRANCH IN SUBSPACE GROUP, or if the ALET used is other than ALET 0 and ALET 1 and the destination ASTE does not specify the base space or a subspace of the subspace group.

The primary space-switch-event-control bit, bit 0 of control register 1 either before or after the operation, does not cause a space-switch-event program interruption to occur.

Key-controlled protection does not apply to any access made during the operation. Low-address protection does apply.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: BRANCH IN SUBSPACE GROUP."

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (dispatchable-unit control table, effective access-list designation, access-list entry, destination ASN-second-table entry)
- ALET specification
- ALEN translation
- ASTE sequence
- ASTE validity
- Operation (if the subspace-group facility is not installed)
- Protection (low-address; dispatchable-unit control table)
- Special operation
- Trace

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B.1 | Operation exception due to subspace-group facility not being installed. |
| 7.B.2 | Special-operation exception due to DAT being off or the address-space-function control, bit 15 of control register 0, being zero. |
| 8.A | Trace exceptions. |
| 8.B | Protection exception (low-address protection) for access to dispatchable-unit control table. |
| 8.C.1 | Addressing exception for access to dispatchable-unit control table. |
| 8.C.2 | Special-operation exception due to current primary address space not being in a subspace group associated with the current dispatchable unit (primary-ASTE origin in control register 5 not equal to base-ASTE origin in dispatchable-unit control table).<br><br>**Note:** Exception 8.C.3.A can occur only if the access-list-entry token (ALET) in access register $R_2$ is ALET 0. |
| 8.C.3.A | Addressing exception for access to base ASTE (ASTE designated by base-ASTE origin in dispatchable-unit control table). |

*Figure 10-4 (Part 1 of 2). Priority of Execution: BRANCH IN SUBSPACE GROUP*


LzLabs GmbH. Ex. 1008-495

| | |
|---|---|
| | **Note:** Exceptions 8.C.3.B.1-8.C.3.B.4 can occur only if the access-list-entry token (ALET) in access register $R_2$ is ALET 1. |
| 8.C.3.B.1 | Special-operation exception due to subspace-ASTE origin in dispatchable-unit control table being zero. |
| 8.C.3.B.2 | Addressing exception for access to subspace ASTE. |
| 8.C.3.B.3 | ASTE-validity exception due to bit 0 in subspace ASTE being one. |
| 8.C.3.B.4 | ASTE-sequence exception due to ASTE sequence number in subspace ASTE not being equal to subspace-ASTE sequence number in dispatchable-unit control table. |
| | **Note:** Exceptions 8.C.3.C.1-8.C.3.C.9 can occur only if the access-list-entry token (ALET) in access register $R_2$ is other than ALET 0 and ALET 1. |
| 8.C.3.C.1 | ALET-specification exception due to bits 0-6 of ALET not being all zeros. |
| 8.C.3.C.2 | Addressing exception for access to effective access-list designation. |
| 8.C.3.C.3 | ALEN-translation exception due to access-list entry being outside the list. |
| 8.C.3.C.4 | Addressing exception for access to access-list entry. |
| 8.C.3.C.5 | ALEN-translation exception due to I bit in access-list entry being one. |
| 8.C.3.C.6 | Addressing exception for access to destination ASTE. |
| 8.C.3.C.7 | ASTE-validity exception due to bit 0 in destination ASTE being one. |
| 8.C.3.C.8 | ASTE-sequence exception due to ASTE sequence number (ASTESN) in access-list entry not being equal to ASTESN in destination ASTE. |
| 8.C.3.C.9 | Special-operation exception due to destination-ASTE origin not equal to base-ASTE origin in dispatchable-unit control table and (1) subspace-group bit, bit 22 in segment-table designation, in destination ASTE being zero or (2) base-space bit, bit 31, in destination ASTE being one. |

*Figure 10-4 (Part 2 of 2). Priority of Execution: BRANCH IN SUBSPACE GROUP*

**Programming Notes:**

1. See the discussion of BRANCH IN SUBSPACE GROUP in "Subroutine Linkage without the Linkage Stack" on page 5-10. It is intended that there be a separate ASN-second-table entry (ASTE) for each of the base space and each subspace of a subspace group. The ASTEs for the subspaces can be "pseudo" ASTEs as described in the programming note in "Address-Space Number" on page 3-17. A subspace can contain a subset of the storage in the base space by having the segment table for the subspace point to a subset of the page tables that are pointed to from the segment table for the base space. A dispatchable unit has access to a subspace if an access-list entry designating the ASTE for the subspace is in



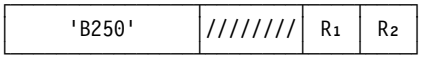
LzLabs GmbH. Ex. 1008-496

the primary-space or dispatchable-unit access list of the dispatchable unit.

2. BRANCH IN SUBSPACE GROUP can be used to give control from the base space to a subspace, from a subspace to another subspace, and from a subspace to the base space. The instruction can also be used to give control from the base space to the base space or from a subspace to the same subspace.
3. Since BRANCH IN SUBSPACE GROUP sets the secondary segment-table designation in control register 7 equal to the new primary segment-table designation in control register 1 (along with setting the secondary ASN in control register 3 equal to the primary ASN in control register 4), the program in an address space given control by BRANCH IN SUBSPACE GROUP does not have access to the calling program's address space by means of that address space being the secondary address space.
4. When a dispatchable unit has used BRANCH IN SUBSPACE GROUP to enter a subspace and has not subsequently used BRANCH IN SUBSPACE GROUP to return to the base space, the dispatchable unit is said to be "subspace active." When PROGRAM CALL, PROGRAM TRANSFER, PROGRAM RETURN, SET SECONDARY ASN, or LOAD ADDRESS SPACE PARAMETERS places a segment-table designation (STD) in control register 1 as the primary STD or in control register 7 as the secondary STD, and if (1) the STD has the subspace-group bit, bit 22, on in it, (2) the dispatchable unit is subspace active, and (3) the STD was obtained from the ASN-second-table entry (ASTE) for the base space of the current dispatchable unit, then the instruction (any of the five named instructions) replaces bits 1-23 and 25-31 of the STD in the control register with the corresponding bits of the STD in the ASTE for the subspace in which the dispatchable unit last had control. Further details about the effects of the subspace-group facility on the five named instructions are given in "Subspace-Replacement Operations" on page 5-59 and in the definitions of the instructions.
5. The use of BRANCH IN SUBSPACE GROUP (BSG) along with PROGRAM CALL (PC) and either PROGRAM TRANSFER (PT) or PROGRAM RETURN (PR) can produce results that may be unexpected. Consider the following sequence of operations:
   a. Start in the base space
   b. BSG to a subspace
   c. PC (the first PC) to an address space that is not in the subspace group.
   d. PC (the second PC) to the base space. Since the dispatchable unit is subspace active, control is given to the subspace.
   e. BSG back to the base space.
   f. PT or PR (paired with the second PC) back to the address space that is not in the subspace group.
   g. PT or PR (paired with the first PC) back to the subspace group. Since the dispatchable unit is no longer subspace active, control is given to the base space even though the first PC was issued in the subspace.
6. BRANCH IN SUBSPACE GROUP does not perform the serialization or checkpoint-synchronization functions, but it does cause all copies of prefetched instructions to be discarded except when in the home-space mode.
7. When the $R_2$ field designates access register 0, the access register is treated as containing ALET 0 regardless of the contents of the access register.

## COMPARE AND SWAP AND PURGE

```
CSP   R1,R2       [RRE]
```




The first and second operands are compared. If they are equal, the contents of general register $R_1 + 1$ are stored at the second-operand location, and a purging operation is performed. If they are unequal, the second operand is loaded at the first-operand location. The result of the comparison is indicated in the condition code.

Bits 16-23 of the instruction are ignored.


LzLabs GmbH. Ex. 1008-497

The first operand is the contents of general register $R_1$. The second operand is a word in storage. The location of the leftmost byte of the second operand is designated by the contents of general register $R_2$.

The purging operation applies to ART-lookaside buffers (ALBs) and translation-lookaside buffers (TLBs) in all CPUs in the configuration. Either ALBs or TLBs, or both ALBs and TLBs, may be selected for purging. All entries are cleared from the selected buffers.

The purging operation is specified by means of bits 30 and 31 of general register $R_2$. When bit 30 is one, entries are cleared from ALBs. When bit 31 is one, entries are cleared from TLBs. When bits 30 and 31 both are ones, entries are cleared from ALBs and TLBs. When bits 30 and 31 both are zeros, no entries are cleared.

The handling of the address in general register $R_2$ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 8-29 of general register $R_2$, with two zeros appended on the right, constitute the address, and the contents of bit positions 0-7 are ignored. In the 31-bit addressing mode, the contents of bit positions 1-29 of general register $R_2$, with two zeros appended on the right, constitute the address, and the contents of bit position 0 are ignored.

The contents of the registers just described are shown in Figure 10-5. When an equal comparison occurs, the contents of general register $R_1 + 1$ are stored at the second-operand location. The fetch of the second operand for purposes of comparison and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs.

When the result of the comparison is unequal, the second-operand location remains unchanged. However, on some models, the contents may be fetched and subsequently stored back unchanged at the second-operand location. This update appears to be a block-concurrent interlocked-update reference as observed by other CPUs.

A serialization function is performed before the operand is fetched and again after the operation is completed.

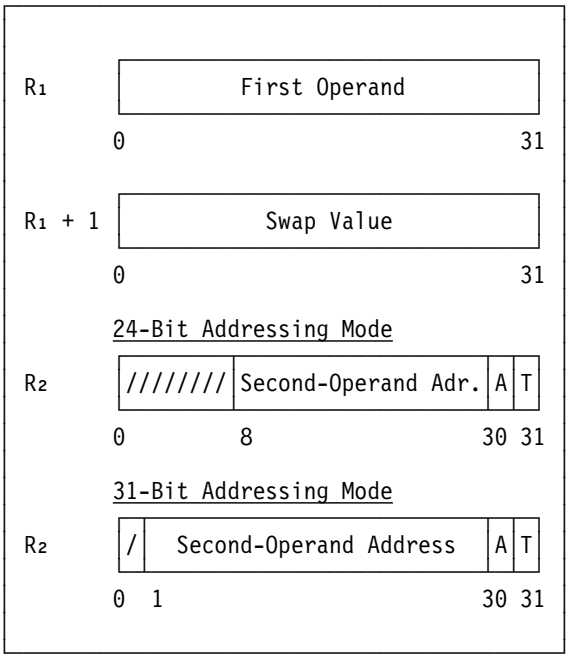



*Figure 10-5. Register Contents for COMPARE AND SWAP AND PURGE*

When an equal comparison occurs, this CPU clears entries from its ALB and TLB, as specified by bits 30 and 31 of general register $R_2$, and signals all CPUs in the configuration to clear the same specified entries from their ALBs and TLBs. The ALB entries that are cleared are all ALB access-list designations, access-list entries, ASN-second-table entries, and authority-table entries. The TLB entries that are cleared are all TLB segment-table entries and page-table entries.

The execution of COMPARE AND SWAP AND PURGE is not completed on the CPU which executes it until (1) all specified entries have been cleared from the ALB and TLB of this CPU and (2) all other CPUs in the configuration have completed any storage accesses, including the updating of the change and reference bits, by using the specified ALB and TLB entries.

**Special Conditions**

The $R_1$ field must designate an even register; otherwise, a specification exception is recognized.

***Resulting Condition Code:***



LzLabs GmbH. Ex. 1008-498

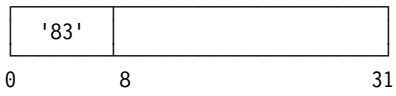
0 First and second operands equal, second operand replaced by contents of general register $R_1 + 1$
1 First and second operands unequal, first operand replaced by second operand
2 --
3 --

***Program Exceptions:***

- Access (fetch and store, operand 2)
- Operation (if the compare-and-swap-and-purge facility is not installed)
- Privileged operation
- Specification

**Programming Note:** COMPARE AND SWAP AND PURGE provides a broadcast form of the PURGE ALB and PURGE TLB instructions, thus making it possible to avoid uses of SIGNAL PROCESSOR.

## DIAGNOSE




The CPU performs built-in diagnostic functions, or other model-dependent functions. The purpose of the diagnostic functions is to verify proper functioning of equipment and to locate faulty components. Other model-dependent functions may include disabling of failing buffers, reconfiguration of CPUs, storage, and channel paths, and modification of control storage.

Bits 8-31 may be used as in the SI or RS formats, or in some other way, to specify the particular diagnostic function. The use depends on the model.

The execution of the instruction may affect the state of the CPU and the contents of a register or storage location, as well as the progress of an I/O operation. Some diagnostic functions may cause the test indicator to be turned on.

***Resulting Condition Code:*** The code is unpredictable.

***Program Exceptions:***

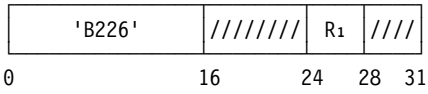
- Privileged operation
- Depending on the model, other exceptions may be recognized.

**Programming Notes:**

1. Since the instruction is not intended for problem-state-program or control-program use, DIAGNOSE has no mnemonic.
2. DIAGNOSE, unlike other instructions, does not follow the rule that programming errors are distinguished from equipment errors. Improper use of DIAGNOSE may result in false machine-check indications or may cause actual machine malfunctions to be ignored. It may also alter other aspects of system operation, including instruction execution and channel-program operation, to an extent that the operation does not comply with that specified in this publication. As a result of the improper use of DIAGNOSE, the system may be left in such a condition that the power-on reset or initial-microprogram-loading (IML) function must be performed. Since the function performed by DIAGNOSE may differ from model to model and between versions of a model, the program should avoid issuing DIAGNOSE unless the program recognizes both the model number and version code stored by STORE CPU ID.

## EXTRACT PRIMARY ASN

```
EPAR   R1                [RRE]
```




The 16-bit PASN, bits 16-31 of control register 4, is placed in bit positions 16-31 of general register $R_1$. Bits 0-15 of the general register are set to zeros.

Bits 16-23 and 28-31 of the instruction are ignored.

**Special Conditions**

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized.


LzLabs GmbH. Ex. 1008-499

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-6.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

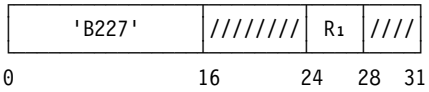
- Privileged operation (extraction-authority control is zero in the problem state)
- Special operation

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to DAT being off. |
| 8. | Privileged-operation exception due to extraction-authority control, bit 4 of control register 0, being zero in problem state. |

*Figure 10-6. Priority of Execution: EXTRACT PRIMARY ASN*

## EXTRACT SECONDARY ASN

ESAR R1 [RRE]




The 16-bit SASN, bits 16-31 of control register 3, is placed in bit positions 16-31 of general register R1. Bits 0-15 of the general register are set to zeros.

Bits 16-23 and 28-31 of the instruction are ignored.

**Special Conditions**

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized.

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-7.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

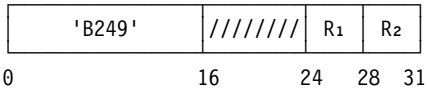
- Privileged operation (extraction-authority control is zero in the problem state)
- Special operation

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to DAT being off. |
| 8. | Privileged-operation exception due to extraction-authority control bit 4 of control register 0, being zero in problem state. |

*Figure 10-7. Priority of Execution: EXTRACT SECONDARY ASN*

## EXTRACT STACKED REGISTERS

EREG R1,R2 [RRE]




The contents of a set of general registers and a set of access registers that were saved in the last state entry in the linkage stack are restored to the registers. Each set of registers begins with register R1 and ends with register R2.

For each of the general registers and the access registers, the registers are loaded in ascending order of their register numbers, starting with register R1 and continuing up to and including register R2, with register 0 following register 15. Each register is loaded from the position in the state entry where the contents of the register were



LzLabs GmbH. Ex. 1008-500

saved when the state entry was created. The contents of the state entry remain unchanged.

The last state entry is located as described in "Unstacking Process" on page 5-75. The state entry remains in the linkage stack, and the linkage-stack-entry address in control register 15 remains unchanged.

Key-controlled protection does not apply to references to the linkage stack.

Bits 16-23 of the instruction are ignored.

**Special Conditions**

The CPU must be in the primary-space mode, access-register mode, or home-space mode, and the address-space-function control, bit 15 of control register 0, must be one; otherwise, a special-operation exception is recognized.

A stack-empty, stack-specification, or stack-type exception may be recognized during the unstacking process.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-8.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, except for protection, linkage-stack entry)
- Special operation
- Stack empty
- Stack specification
- Stack type

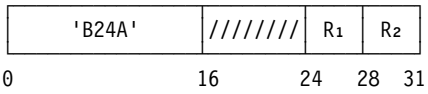
| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to DAT being off, the CPU being in the secondary-space mode or the address-space-function control, bit 15 of control register 0, being zero. |
| 8. | Access exceptions (fetch) for entry descriptor of the current linkage-stack entry. |
| 9. | Stack-type exception due to current entry not being a state entry or header entry.<br><br>**Note:** Exceptions 10-14 can occur only if the current entry is a header entry. |
| 10. | Access exceptions (fetch) for second word of the header entry. |
| 11. | Stack-empty exception due to backward stack-entry validity bit in the header entry being zero. |
| 12. | Access exceptions (fetch) for entry descriptor of preceding entry, which is the entry designated by the backward stack-entry address in the current (header) entry. |
| 13. | Stack-specification exception due to preceding entry being a header entry. |
| 14. | Stack-type exception due to preceding entry not being a state entry. |
| 15. | Access exceptions (fetch) for the selected contents of the state entry. |

*Figure 10-8. Priority of Execution: EXTRACT STACKED REGISTERS*


LzLabs GmbH. Ex. 1008-501

## EXTRACT STACKED STATE

ESTA R1,R2 [RRE]



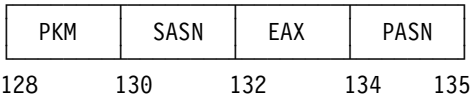

The contents of one of the four eight-byte fields immediately preceding the entry descriptor of the last state entry in the linkage stack are placed in the pair of general registers designated by the $R_1$ field. The condition code is set to indicate whether the state entry is a branch state entry or a program-call state entry.

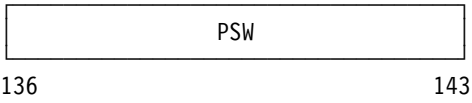
The $R_1$ field designates the even-numbered register of an even-odd pair of general registers.

Bits 24-31 of general register $R_2$ are an unsigned binary integer that is used as a code to select the state-entry byte positions from which information is to be extracted, as follows:

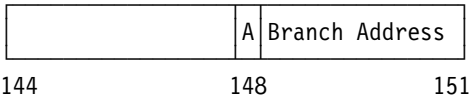
| Code (Bits 24-31 of Gen. Reg. $R_2$) | State-Entry Byte Positions Selected |
|---|---|
| 0 | 128-135 |
| 1 | 136-143 |
| 2 | 144-151 |
| 3 | 152-159 |

The format of byte positions 128-159 of the state entry is as follows:

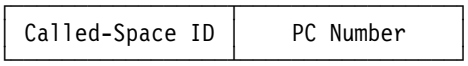


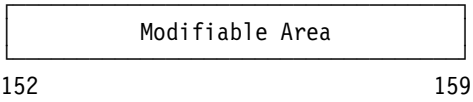




In a Branch State Entry




In a Program-Call State Entry




The contents of the state entry remain unchanged.

The last state entry is located as described in "Unstacking Process" on page 5-75. The state entry remains in the linkage stack, and the linkage-stack-entry address in control register 15 remains unchanged.

When the entry-type code in the entry descriptor of the state entry is 0000100 binary, indicating a branch state entry, the condition code is set to 0. When the entry-type code is 0000101 binary, indicating a program-call state entry, the condition code is set to 1.

Key-controlled protection does not apply to references to the linkage stack.

Bits 16-23 of the instruction and bits 0-23 of general register $R_2$ are ignored.

**Special Conditions**

A specification exception is recognized when $R_1$ is odd or the code in bit positions 24-31 of general register $R_2$ is greater than 3.

The CPU must be in the primary-space mode, access-register mode, or home-space mode, and the address-space-function control, bit 15 of control register 0, must be one; otherwise, a special-operation exception is recognized.

A stack-empty, stack-specification, or stack-type exception may be recognized during the unstacking process.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-9 on page 10-23.



LzLabs GmbH. Ex. 1008-502

***Resulting Condition Code:***

0 Branch state entry
1 Program-call state entry
2 --
3 --

***Program Exceptions:***

- Access (fetch, except for protection, linkage-stack entry)
- Special operation
- Specification
- Stack empty
- Stack specification
- Stack type

```
1.-6.  Exceptions with the same priority as the priority of program-
       interruption conditions for the general case.

7.A    Access exceptions for second instruction halfword.

7.B    Special-operation exception due to DAT being off, the CPU
       being in the secondary-space mode or the address-space-
       function control, bit 15 of control register 0, being zero.

8.A    Specification exception due to R1 being odd or bits 24-31 of
       general register R2 having a value greater than 3.

8.B.1  Access exceptions (fetch) for entry descriptor of the current
       linkage-stack entry.

8.B.2  Stack-type exception due to current entry not being a state
       entry or header entry.

       Note:  Exceptions 8.B.3-8.B.7 can occur only if the current
       entry is a header entry.

8.B.3  Access exceptions (fetch) for second word of the header entry.

8.B.4  Stack-empty exception due to backward stack-entry validity
       bit in the header entry being zero.

8.B.5  Access exceptions (fetch) for entry descriptor of preceding
       entry, which is the entry designated by the backward stack-
       entry address in the current (header) entry.

8.B.6  Stack-specification exception due to preceding entry being a
       header entry.

8.B.7  Stack-type exception due to preceding entry not being a state
       entry.

8.B.8  Access exceptions (fetch) for the selected contents of the
       state entry.
```

*Figure 10-9. Priority of Execution: EXTRACT STACKED STATE*

## INSERT ADDRESS SPACE CONTROL

```
IAC    R1              [RRE]

 ------------------------------------------------
|     'B224'     |////////|  R1  |////|
 ------------------------------------------------
0                16       24     28  31
```

The address-space-control bits, bits 16 and 17 of the current PSW, are placed in reversed order in bit positions 22 and 23 of general register $R_1$; that is, bit 16 is placed in bit position 23, and bit 17 is placed in bit position 22. Bits 16-21 of the register are set to zeros, and bits 0-15 and 24-31 of the register remain unchanged. The address-


LzLabs GmbH. Ex. 1008-503

space-control bits are also used to set the condition code.

Bits 16-23 and 28-31 of the instruction are ignored.

**Special Conditions**

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized.

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-10.

***Resulting Condition Code:***

0 PSW bits 16 and 17 zeros (indicating primary-space mode)
1 PSW bit 16 one and bit 17 zero (indicating secondary-space mode)
2 PSW bit 16 zero and bit 17 one (indicating access-register mode)
3 PSW bits 16 and 17 ones (indicating home-space mode)

***Program Exceptions:***

- Privileged operation (extraction-authority control is zero in the problem state)
- Special operation

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to DAT being off. |
| 8. | Privileged-operation exception due to extraction-authority control, bit 4 of control register 0, being zero in problem state. |

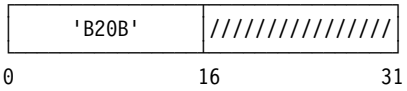
*Figure 10-10. Priority of Execution: INSERT ADDRESS SPACE CONTROL*

**Programming Notes:**

1. Bits 16-21 of general register $R_1$ are reserved for expansion for use with possible future facilities. The program should not depend on these bits being set to zeros.
2. INSERT ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL are defined to operate on the third byte of a general register so that the address-space-control bits can be saved in the same general register as the PSW key, which is placed in the fourth byte of general register 2 by INSERT PSW KEY.

## INSERT PSW KEY

IPK [S]




The four-bit PSW-key, bits 8-11 of the current PSW, is inserted in bit positions 24-27 of general register 2, and bits 28-31 of that register are set to zeros. Bits 0-23 of general register 2 remain unchanged.

Bits 16-31 of the instruction are ignored.

**Special Conditions**

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

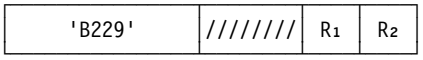
- Privileged operation (extraction-authority control is zero in the problem state)



LzLabs GmbH. Ex. 1008-504

## INSERT STORAGE KEY EXTENDED

ISKE R1,R2 [RRE]




The storage key for the block that is addressed by the contents of general register R2 is inserted in general register R1.

Bits 16-23 of the instruction are ignored.

In the 24-bit addressing mode, bits 8-19 of general register R2 designate a 4K-byte block in real storage, and bits 0-7 and 20-31 of the register are ignored. In the 31-bit addressing mode, bits 1-19 of general register R2 designate a 4K-byte block in real storage, and bits 0 and 20-31 of the register are ignored.

The address designating the storage block, being a real address, is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The seven-bit storage key is inserted in bit positions 24-30 of general register R1, and bit 31 is set to zero. The contents of bit positions 0-23 of the register remain unchanged.

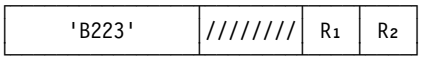
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (address specified by general register R2)
- Privileged operation

## INSERT VIRTUAL STORAGE KEY

IVSK R1,R2 [RRE]




The storage key for the location designated by the virtual address in general register R2 is inserted in general register R1.

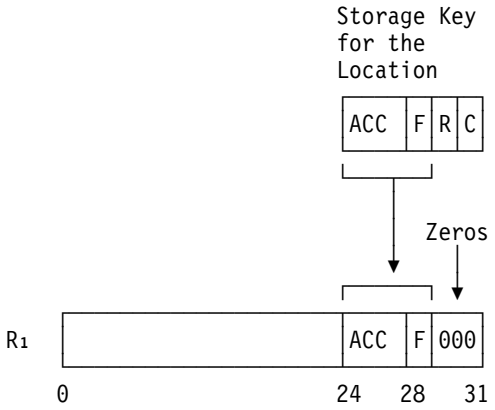
Bits 16-23 of the instruction are ignored.

Selected bits of general register R2 are used as a virtual address. In the 24-bit addressing mode, the address is specified by bits 8-31 of the register, and bits 0-7 are ignored. In the 31-bit addressing mode, the address is specified by bits 1-31, and bit 0 is ignored.

The address is a virtual address and is subject to the address-space-control bits, bits 16 and 17 of the current PSW. The address is treated as a primary virtual address in the primary-space mode, as a secondary virtual address in the secondary-space mode, as an AR-specified virtual address in the access-register mode, or as a home virtual address in the home-space mode. The reference to the storage key is not subject to a protection exception.

Bits 0-4 of the storage key, which are the access-control bits and the fetch-protection bit, are placed in bit positions 24-28 of general register R1, with bits 29-31 set to zeros. The contents of bit positions 0-23 of the register remain unchanged. The change and reference bits in the storage key are not inspected. The change bit is not affected by the operation. The reference bit, depending on the model, may or may not be set to one as a result of the operation.

The following diagram shows the storage key and the register positions just described.




**Special Conditions**

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized.

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one;


LzLabs GmbH. Ex. 1008-505

otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-11.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (except for protection, address specified by general register $R_2$)
- Privileged operation (extraction-authority control is zero in the problem state)
- Special operation

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to DAT being off. |
| 8. | Privileged-operation exception due to extraction-authority control, bit 4 of control register 0, being zero. |
| 9. | Access exceptions (except for protection) for address specified by general register $R_2$. |

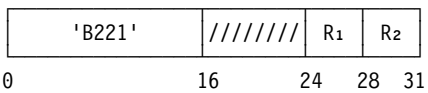
*Figure 10-11. Priority of Execution: INSERT VIRTUAL STORAGE KEY*

**Programming Notes:**

1. Since all bytes in a 4K-byte block are associated with the same page and the same storage key, bits 20-31 of general register $R_2$ essentially are ignored.
2. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

# INVALIDATE PAGE TABLE ENTRY

```
IPTE   R1,R2              [RRE]
```

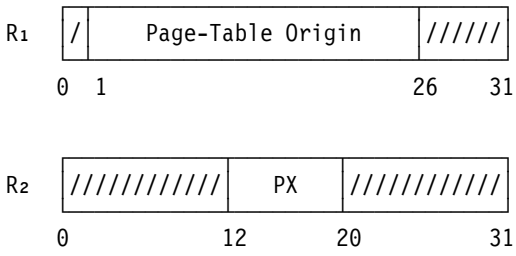



The designated page-table entry is invalidated, and the translation-lookaside buffers (TLBs) in all CPUs in the configuration are cleared of the associated entries.

Bits 16-23 of the instruction are ignored.

The contents of general register $R_1$ have the format of a segment-table entry, with only the page-table origin used. The contents of general register $R_2$ have the format of a virtual address, with only the page index used. The contents of fields that are not part of the page-table origin or page index are ignored.

The contents of the general registers just described are as follows:




The page-table origin and the page index designate a page-table entry in accordance with the dynamic-address-translation rules for page-table lookup. The page-table origin is treated as a 31-bit address, and the addition is performed by using the rules for 31-bit address arithmetic, regardless of the setting of the addressing mode, which is specified by bit 32 of the current PSW. A carry into bit position 0 as a result of the addition of the page index and page-table origin is ignored. The address formed from these two components is a real address. The page-invalid bit of this page-table entry is set to one. During this procedure, no page-table-length check is made, and the page-table entry is not inspected for whether the page-invalid bit is already one or for format errors. Additionally, the page-frame real address contained in the entry is not checked for an addressing exception.

The entire page-table entry appears to be fetched concurrently from storage as observed by other CPUs. Subsequently, the byte containing the page-invalid bit is stored. The fetch access to the page-table entry is subject to key-controlled protection, and the store access is subject to key-controlled protection and low-address protection.



LzLabs GmbH. Ex. 1008-506

A serialization function is performed before the operation begins and again after the operation is completed. As is the case for all serialization operations, this serialization applies only to this CPU; other CPUs are not necessarily serialized.

If no exceptions are recognized, this CPU clears selected entries from its TLB and signals all CPUs in the configuration to clear selected entries from their TLBs. Each TLB is cleared of at least those entries that have been formed using all of the following:

- The page-table origin specified by general register $R_1$
- The page index specified by general register $R_2$
- The page-frame real address contained in the designated page-table entry

The execution of INVALIDATE PAGE TABLE ENTRY is not completed on the CPU which executes it until (1) all entries corresponding to the specified parameters have been cleared from the TLB of this CPU and (2) all other CPUs in the configuration have completed any storage accesses, including the updating of the change and reference bits, by using TLB entries corresponding to the specified parameters.

**Special Conditions**

When bit positions 8-12 of control register 0 contain an invalid code, a translation-specification exception is recognized. The exception is recognized regardless of whether DAT is on or off.

The operation is suppressed on all addressing and protection exceptions.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (page-table entry)
- Privileged operation
- Protection (fetch and store, page-table entry, key-controlled protection, and low-address protection)
- Translation specification (bits 8-12 in control register 0 only)

**Programming Notes:**

1. The selective clearing of entries may be implemented in different ways, depending on the model, and, in general, more entries may be cleared than the minimum number required. Some models may clear all entries which contain the page-frame real address obtained from the page-table entry in storage. Others may clear all entries which contain the designated page index, and some implementations may clear precisely the minimum number of entries required. Therefore, in order for a program to operate on all models, the program should not take advantage of any properties obtained by a less selective clearing on a particular model.
2. The clearing of TLB entries may make use of the page-frame real address in the page-table entry. Therefore, if the page-table entry, when in the attached state, ever contained a page-frame real address that is different from the current value, copies of entries containing the previous values may remain in the TLB.
3. INVALIDATE PAGE TABLE ENTRY cannot be safely used to update a shared location in main storage if the possibility exists that another CPU or a channel program may also be updating the location.
4. The address of the page-table entry for INVALIDATE PAGE TABLE ENTRY is a 31-bit real address, and the address arithmetic is performed by following the normal rules for 31-bit address arithmetic with wraparound at $2^{31} - 1$. Contrast this with implicit translation and the translation for LOAD REAL ADDRESS, both of which, depending on the model, may treat addresses of DAT-table entries as either real or absolute and may result either in wraparound or in an addressing exception when a carry occurs into bit position 0. Accordingly, the DAT tables should not be specified to wrap from maximum storage locations to location 0 and should not be placed at storage locations whose real and absolute addresses are different.


LzLabs GmbH. Ex. 1008-507

# LOAD ADDRESS SPACE PARAMETERS

```
LASP   D1(B1),D2(B2)              [SSE]

        ┌──────────┬────┬────┬────┬────┐
        │  'E500'  │ B1 │ D1 │ B2 │ D2 │
        └──────────┴────┴────┴────┴────┘
0                   16   20   32   36  47
```

The doubleword at the first-operand location contains values to be loaded into control registers 3 and 4, including a secondary ASN and a primary ASN. Execution of the instruction consists in performing four major steps: PASN translation, SASN translation, SASN authorization, and control-register loading. Each of these steps may or may not be performed, depending on the outcome of certain tests and on the setting of bits 29-31 of the second-operand address. The first three of these steps, when performed and successful, obtain additional values, which are loaded into control registers 1, 5, and 7. When the first three steps are not successful when performed, no control registers are changed, and the reason is indicated in the condition code.

When the address-space-function (ASF) control, bit 15 of control register 0, is zero, control register 5 contains the linkage-table designation (LTD), and this instruction may place a new LTD in control register 5. When the ASF control is one, control register 5 contains the primary-ASN-second-table-entry origin (PASTEO), and this instruction may place a new PASTEO in control register 5. For simplicity, this definition sometimes first describes an operation as if the ASF control were zero and then describes the different operation that occurs when the ASF control is one.

The doubleword first operand contains a PSW-key mask (PKM), a secondary ASN (SASN), an authorization index (AX), and a primary ASN (PASN). The primary ASN may be translated by means of the ASN-translation tables to obtain a primary segment-table designation (PSTD), LTD or PASTEO, and, optionally, an AX. The secondary ASN may be translated by means of the ASN-translation tables to obtain an SSTD, and, optionally, an authority check may be made to ensure that the new AX is authorized to establish the new SASN.

The doubleword at the first-operand location has the following format:

```
┌──────────┬──────────┬──────┬──────────┐
│  PKM-d   │  SASN-d  │ AX-d │  PASN-d  │
└──────────┴──────────┴──────┴──────────┘
0          16         32     48         63
```

The “d” stands for designated doubleword and is used to distinguish these fields from other fields with similar names which are referred to in the definition. The current contents of the corresponding fields in the control registers are referred to as PKM-old, SASN-old, etc. The updated contents of the control registers are referred to as PKM-new, SASN-new, etc.

The second-operand address is not used to address data; instead, the rightmost three bits are used to control portions of the operation. The remainder of the second-operand address is ignored. Bits 29-31 of the second-operand address are used as follows:

| | Function Specified in Second-Operand Address | |
|---|---|---|
| Bit | When Bit Is Zero | When Bit Is One |
| 29 | ASN translation performed only when new ASN and old ASN are different. | ASN translation performed.* |
| 30 | AX associated with new PASN used. | AX in first operand used. |
| 31 | SASN authorization performed.* | SASN authorization not performed. |

* SASN translation and SASN authorization are performed only when SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d, the SSTD is set equal to the PSTD, and no authorization is performed.

The operation of LOAD ADDRESS SPACE PARAMETERS is depicted in Figure 10-15 on page 10-36.

### PASN Translation and Related Processing

In the PASN-translation process, the PASN-d is translated by means of the ASN first table and the ASN second table. The STD and LTD fields, and optionally the AX field, obtained from the ASN-second-table entry (ASTE) are subsequently used to update the corresponding control registers. However, when the ASF control is one, the



LzLabs GmbH. Ex. 1008-508

LTD is not obtained, and the PASTEO resulting from PASN translation is used to update control register 5.

When bit 29 of the second-operand address is one, PASN translation is always performed. When bit 29 is zero, PASN translation is performed only when PASN-d is not equal to PASN-old. When bit 29 is zero and PASN-d is equal to PASN-old, the PSTD-old and the LTD-old or PASTEO-old are left unchanged in the control registers and become the PSTD-new and the LTD-new or PASTEO-new, respectively. In this case, if bit 30 is zero, then the AX-old is left unchanged in the control register and becomes the AX-new, or, if bit 30 is one, AX-new is set equal to AX-d.

The PASN translation follows the normal rules for ASN translation, except that the invalid bits, bit 0 in the ASN-first-table entry and bit 0 in the ASTE, when ones, do not result in an ASN-translation exception. When either of the invalid bits is one, condition code 1 is set. When a reason for setting condition code 1 does not exist and either the current primary space-switch-event-control bit in control register 1 is one or the space-switch-event-control bit in the ASTE is one, a space-switch event does not occur; instead, condition code 3 is set. When condition code 1 or 3 is set, the control registers remain unchanged.

The contents of the AX, STD, and LTD fields in the ASTE which is accessed as a result of the PASN translation are referred to as AX-p, STD-p, and LTD-p, respectively. The origin of the ASTE is referred to as PASTEO-p.

The description in this paragraph applies if the subspace-group facility is installed, the ASF control is one, and PASN translation is performed. After STD-p has been obtained, if (1) the subspace-group-control bit, bit 22, in STD-p is one, (2) the dispatchable unit is subspace active, and (3) PASTEO-p designates the ASTE for the base space of the dispatchable unit, then a copy of STD-p, called STD-rp, is made, and bits 1-23 and 25-31 of STD-rp are replaced by bits 1-23 and 25-31 of the STD in the ASTE for the subspace in which the dispatchable unit last had control. Further details are in "Subspace-Replacement Operations" on page 5-59. If bit 0 in the subspace ASTE is one, or if the ASTE sequence number (ASTESN) in the subspace ASTE does not equal the subspace ASTESN in the dispatchable-unit control table, an exception is not recognized; instead, condition code 1 is set, and the control registers remain unchanged.

**SASN Translation and Related Processing**

In the SASN-translation process, the SASN-d is translated by means of the ASN first table and the ASN second table. The STD field obtained from the ASTE subsequently replaces the secondary-segment-table designation (SSTD) in control register 7.

SASN translation is performed only when, but not necessarily when, SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d, SSTD-new is set equal to PSTD-new. When SASN-d is not equal to PASN-d and is equal to SASN-old, bit 29 (force ASN translation) is zero, and bit 31 (skip SASN authorization) is one, SASN translation is not performed, and SSTD-old becomes SSTD-new.

SASN translation is performed in each of the following cases:

- SASN-d is not equal to PASN-d or SASN-old.
- SASN-d is not equal to PASN-d but is equal to SASN-old, and either bit 29 (force ASN translation) of the second-operand address is one or bit 31 (skip secondary authority test) of that address is zero. (The translation must be performed when bit 31 is zero in order to obtain the ATO and ATL from the SASTE.)

The SASN translation follows the normal rules for ASN translation, except that the invalid bits, bit 0 in the ASN-first-table entry and bit 0 in the ASTE, when ones, do not result in an ASN-translation exception. When either of the invalid bits is one, condition code 2 is set, and the control registers remain unchanged.

The contents of the STD, ATO, and ATL fields in the ASTE which is accessed as a result of the SASN translation are referred to as STD-s, ATO-s, and ATL-s, respectively. The origin of the ASTE is referred to as SASTEO-s.

The description in this paragraph applies if the subspace-group facility is installed, the ASF control is one, and SASN translation is performed. After STD-s has been obtained, if (1) the


LzLabs GmbH. Ex. 1008-509

subspace-group-control bit, bit 22, in STD-s is one, (2) the dispatchable unit is subspace active, and (3) SASTEO-s designates the ASTE for the base space of the dispatchable unit, then a copy of STD-s, called STD-rs, is made, and bits 1-23 and 25-31 of STD-rs are replaced by bits 1-23 and 25-31 of the STD in the ASTE for the subspace in which the dispatchable unit last had control. Further details are in "Subspace-Replacement Operations" on page 5-59. If bit 0 in the subspace ASTE is one, or if the ASTE sequence number (ASTESN) in the subspace ASTE does not equal the subspace ASTESN in the dispatchable-unit control table, an exception is not recognized; instead, condition code 2 is set, and the control registers remain unchanged.

**SASN Authorization**

SASN authorization is performed when bit 31 of the second-operand address is zero and SASN-d is not equal to PASN-d; it is performed in this case regardless of whether SASN-d is equal to SASN-old. When SASN-d is equal to PASN-d or when bit 31 of the second-operand address is one, SASN authorization is not performed.

SASN authorization is performed by using ATO-s, ATL-s, and the intended value for AX-new. When bit 30 of the second-operand address is zero and PASN translation was performed, the intended value for AX-new is AX-p. When bit 30 of that address is zero and PASN translation was not performed, the AX is not changed, and AX-new is the same as AX-old. When bit 30 of that address is one, the intended value for AX-new is AX-d. SASN authorization follows the rules for secondary authorization as described in "ASN-Authorization Process" on page 3-24. If the SASN is not authorized (that is, the authority-table length is exceeded, or the selected bit is zero), condition code 2 is set, and none of the control registers is updated.

**Control-Register Loading**

When the PASN-translation and SASN-translation functions and related functions, and the SASN-authorization functions and subspace-replacement operations, if called for in the instruction execution, are performed without encountering any exceptions or exception conditions, the execution is completed by replacing the contents of control registers 1, 3, 4, 5, and 7 with the new values, and condition code 0 is set. The control registers are loaded as follows.

The PSW-key-mask, bits 32-47, and SASN, bits 48-63, in control register 3 are replaced by the contents of the PKM-d and SASN-d fields of the first operand.

The PASN, bits 16-31 of control register 4, is replaced by the PASN-d of the first operand.

The authorization index, bits 0-15 of control register 4, is replaced as follows:

- When bit 30 of the second-operand address is one, by AX-d.
- When bit 30 of the second-operand address is zero and PASN translation is performed, by AX-p.
- When bit 30 of the second-operand address is zero and PASN translation is not performed, the authorization index remains unchanged.

The primary segment-table designation in control register 1 and the linkage-table designation or primary-ASN-second-table-entry origin (PASTEO) in control register 5 are replaced as follows:

- When PASN translation is performed, the primary segment-table designation in control register 1 is replaced by the STD-p obtained as a result of PASN translation, except that it is replaced by STD-rp if a subspace-replacement operation was performed on STD-p. Also, the linkage-table designation in control register 5 is replaced from the LTD-p field if the ASF control is zero, or the primary-ASTE origin (PASTEO) in control register 5 is replaced by PASTEO-p if the ASF control is one. When the ASF control is one, PASTEO-p is placed in bit positions 1-25 of control register 5, and zeros are placed in bit positions 0 and 26-31.
- When PASN translation is not performed, the contents of control registers 1 and 5 remain unchanged.

The secondary segment-table designation in control register 7 is replaced as follows:

- When SASN-d equals PASN-d, by the new contents of control register 1, the primary segment-table designation. The new contents may be PSTD-old, STD-p, or STD-rp.



LzLabs GmbH. Ex. 1008-510

- When SASN translation is performed, by STD-s, or by STD-rs if a subspace-replacement operation was performed on STD-s.

When SASN-d does not equal PASN-d and SASN translation is not performed, the secondary segment-table designation remains unchanged.

**Other Condition-Code Settings**

When PASN translation is called for and cannot be completed because bit 0 is one in either the ASN-first-table entry or the ASTE, or if it can be completed but a subspace-replacement-exception condition exists due to bit 0 or the ASTE sequence number in the subspace ASTE during a subspace-replacement operation on the STD-p, condition code 1 is set, and the control registers are not changed.

When PASN translation is called for and completed and any required subspace-replacement operation on the STD-p is also completed, and then either (1) the current primary space-switch-event-control bit, bit 0 of control register 1, is one or (2) the space-switch-event-control bit in the ASTE designated by PASTEO-p is one, condition code 3 is set, and the control registers are not changed.

When SASN translation is called for and the translation cannot be completed because bit 0 is one in either the ASN-first-table entry or the ASTE, or if it can be completed but (1) SASN authorization is called for and the SASN is not authorized, or (2) a subspace-replacement-exception condition exists due to bit 0 or the ASTE sequence number in the subspace ASTE during a subspace-replacement operation on the STD-s, condition code 2 is set, and the control registers are not changed.

**Special Conditions**

The instruction can be executed only when the ASN-translation control, bit 12 of control register 14, is one. If the ASN-translation-control bit is zero, a special-operation exception is recognized.

The first operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

Figure 10-13 on page 10-33 and Figure 10-12 on page 10-32 summarize the functions of the instruction and the priority of recognition of exceptions and condition codes.

***Resulting Condition Code:***

0 Translation and authorization complete; parameters loaded
1 Primary ASN or subspace not available; parameters not loaded
2 Secondary ASN not available or not authorized, or secondary subspace not available; parameters not loaded
3 Space-switch event specified; parameters not loaded

***Program Exceptions:***

- Access (fetch, operand 1)
- Addressing (ASN-first-table entry, ASN-second-table entry, authority-table entry, dispatchable-unit control table)
- ASN-translation specification
- Privileged operation
- Special operation
- Specification


LzLabs GmbH. Ex. 1008-511

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second and third instruction halfwords. |
| 7.B.1 | Privileged-operation exception. |
| 7.B.2 | Special-operation exception due to the ASN-translation control, bit 12 of control register 14, being zero. |
| 8. | Specification exception. |
| 9. | Access exceptions for the first operand. |
| 10. | PASN translation and related processing (when performed). |
| 10.1 | Addressing exception for access to ASN-first-table entry. |
| 10.2 | Condition code 1 due to I bit (bit 0) in ASN-first-table entry being one. |
| 10.3 | ASN-translation-specification exception due to invalid ones (bits 28-31 or 26-31) in ASN-first-table entry (optional). |
| 10.4 | Addressing exception for access to ASN-second-table entry. |
| 10.5 | Condition code 1 due to I bit (bit 0) in ASN-second-table entry being one. |
| 10.6 | ASN-translation-specification exception due to invalid ones (bits 30, 31, 60-63) in ASN-second-table entry (optional). |
| 10.7 | Addressing exception for access to dispatchable-unit control table. |
| 10.8 | Addressing exception for access to subspace ASN-second-table entry. |
| 10.9 | Condition code 1 due to I bit (bit 0) in subspace ASN-second-table entry being one. |
| 10.10 | Condition code 1 due to subspace ASN-second-table-entry sequence number (SSASTESN) in dispatchable-unit control table not being equal to ASTESN in subspace ASN-second-table entry. |
| 10.11 | Condition code 3 due to either the old or new space-switch-event-control bit being one. |
| 11. | SASN translation and related processing (when performed). |
| 11.1 | Addressing exception for access to ASN-first-table entry. |
| 11.2 | Condition code 2 due to I bit (bit 0) in ASN-first-table entry being one. |
| 11.3 | ASN-translation-specification exception due to invalid ones (bits 28-31 or 26-31) in ASN-first-table entry (optional). |
| 11.4 | Addressing exception for access to ASN-second-table entry. |
| 11.5 | Condition code 2 due to I bit (bit 0) in ASN-second-table entry being one. |

*Figure 10-12 (Part 1 of 2). Priority of Execution: LOAD ADDRESS SPACE PARAMETERS*



LzLabs GmbH. Ex. 1008-512

| | |
|---|---|
| 11.6 | ASN-translation-specification exception due to invalid ones (bits 30, 31, 60-63) in ASN-second-table entry (optional). |
| 12.A | Execution of secondary authorization (when performed). |
| 12.A.1 | Condition code 2 due to authority-table entry being outside table. |
| 12.A.2 | Addressing exception for access to authority-table entry. |
| 12.A.3 | Condition code 2 due to S bit in authority-table entry being zero. |
| 12.B.1 | Addressing exception for access to dispatchable-unit control table. |
| 12.B.2 | Addressing exception for access to subspace ASN-second-table entry. |
| 12.B.3 | Condition code 2 due to I bit (bit 0) in subspace ASN-second-table entry being one. |
| 12.B.4 | Condition code 2 due to subspace ASN-second-table-entry sequence number (SSASTESN) in dispatchable-unit control table not being equal to ASTESN in subspace ASN-second-table entry. |

*Figure 10-12 (Part 2 of 2). Priority of Execution: LOAD ADDRESS SPACE PARAMETERS*

| PASN-d Equals PASN-old | Second-Operand-Address Bits[1] | | PASN Translation Performed | Result Field | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 29 | 30 | | PSTD-new | AX-new | CR5-new[2] | PKM-new | SASN-new | PASN-new |
| Yes | 0 | 0 | No | PSTD-old | AX-old | CR5-old | PKM-d | SASN-d | PASN-d |
| Yes | 0 | 1 | No | PSTD-old | AX-d | CR5-old | PKM-d | SASN-d | PASN-d |
| Yes | 1 | 0 | Yes | STD-p[3] | AX-p | CR5-p | PKM-d | SASN-d | PASN-d |
| Yes | 1 | 1 | Yes | STD-p[3] | AX-d | CR5-p | PKM-d | SASN-d | PASN-d |
| No | - | 0 | Yes | STD-p[3] | AX-p | CR5-p | PKM-d | SASN-d | PASN-d |
| No | - | 1 | Yes | STD-p[3] | AX-d | CR5-p | PKM-d | SASN-d | PASN-d |

*Figure 10-13 (Part 1 of 2). Summary of Actions: LOAD ADDRESS SPACE PARAMETERS*


LzLabs GmbH. Ex. 1008-513

| SASN-d Equals PASN-d | SASN-d Equals SASN-old | Second-Operand-Address Bits[1] | | SASN Translation Performed | SASN Authorization Performed[4] | Result Field SSTD-new |
|---|---|---|---|---|---|---|
| | | 29 | 31 | | | |
| Yes | - | - | - | No | No | PSTD-new |
| No | Yes | 0 | 1 | No | No | SSTD-old |
| No | Yes | 1 | 1 | Yes | No | STD-s[5] |
| No | Yes | - | 0 | Yes | Yes | STD-s[5] |
| No | No | - | 1 | Yes | No | STD-s[5] |
| No | No | - | 0 | Yes | Yes | STD-s[5] |

**Explanation:**

- Action in this case is the same regardless of the outcome of this comparison or of the setting of this bit.

[1] Second-operand-address bits:
29 Force ASN translation.
30 Use AX from first operand.
31 Skip secondary authority test.

[2] "CR5" stands for "LTD" if the ASF control, bit 15 of control register 0, is zero or for "PASTEO" if the ASF control is one.

[3] PSTD-new is STD-rp (a copy of STD-p except with bits 1-23 and 25-31 replaced from the STD in the subspace ASTE) if subspace replacement is performed.

[4] SASN authorization is performed using ATO-s, ATL-s, and AX-new.

[5] SSTD-new is STD-rs (a copy of STD-s except with bits 1-23 and 25-31 replaced from the STD in the subspace ASTE) if subspace replacement is performed.

*Figure 10-13 (Part 2 of 2). Summary of Actions: LOAD ADDRESS SPACE PARAMETERS*

**Programming Notes:**

1. Bits 29 and 31 in the second-operand address are intended primarily to provide improved performance for those cases where the associated action is unnecessary.

   When bit 29 is set to zero, the action of the instruction is based on the assumption that the current values for PSTD-old, LTD-old or PASTEO-old, and AX-old are consistent with PASN-old and that SSTD-old is consistent with SASN-old. When this is not the case, bit 29 should be set to one.

   Bit 31, when one, eliminates the SASN-authorization test. The program may be able to determine in certain cases that the SASN is authorized, either because of prior use or because the AX being loaded is authorized to access all address spaces.

2. The SASN-translation and SASN-authorization steps are not performed when SASN-d is equal to PASN-d. This is consistent with the action in SET SECONDARY ASN to current primary (SSAR-cp), which does not perform the translation or ASN authorization.

3. The storage-operand references for LOAD ADDRESS SPACE PARAMETERS may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)

4. See Figure 10-14 on page 10-35 for a listing of abbreviations used in this instruction description.



LzLabs GmbH. Ex. 1008-514

| Control-Register Number.Bit | Abbreviation for | |
|---|---|---|
| | Previous Contents | Subsequent Contents |
| 1.0-31 | PSTD-old | PSTD-new |
| 3.0-15 | PKM-old | PKM-new |
| 3.16-31 | SASN-old | SASN-new |
| 4.0-15 | AX-old | AX-new |
| 4.16-31 | PASN-old | PASN-new |
| 5.0-31 | LTD-old | LTD-new |
| 5.1-25 | PASTEO-old | PASTEO-new |
| 7.0-31 | SSTD-old | SSTD-new |

| First-Operand Bit Positions | Abbreviation |
|---|---|
| 0-15 | PKM-d |
| 16-31 | SASN-d |
| 32-47 | AX-d |
| 48-63 | PASN-d |

| Field in ASN-Second-Table Entry | Abbreviation Used for the Field When Accessed as Part of | |
|---|---|---|
| | PASN Translation | SASN Translation |
| 1-29 | - | ATO-s |
| 32-47 | AX-p | - |
| 48-59 | - | ATL-s |
| 64-95 | STD-p[1] | STD-s[1] |
| 96-127 | LTD-p[2] | - |

**Explanation:**

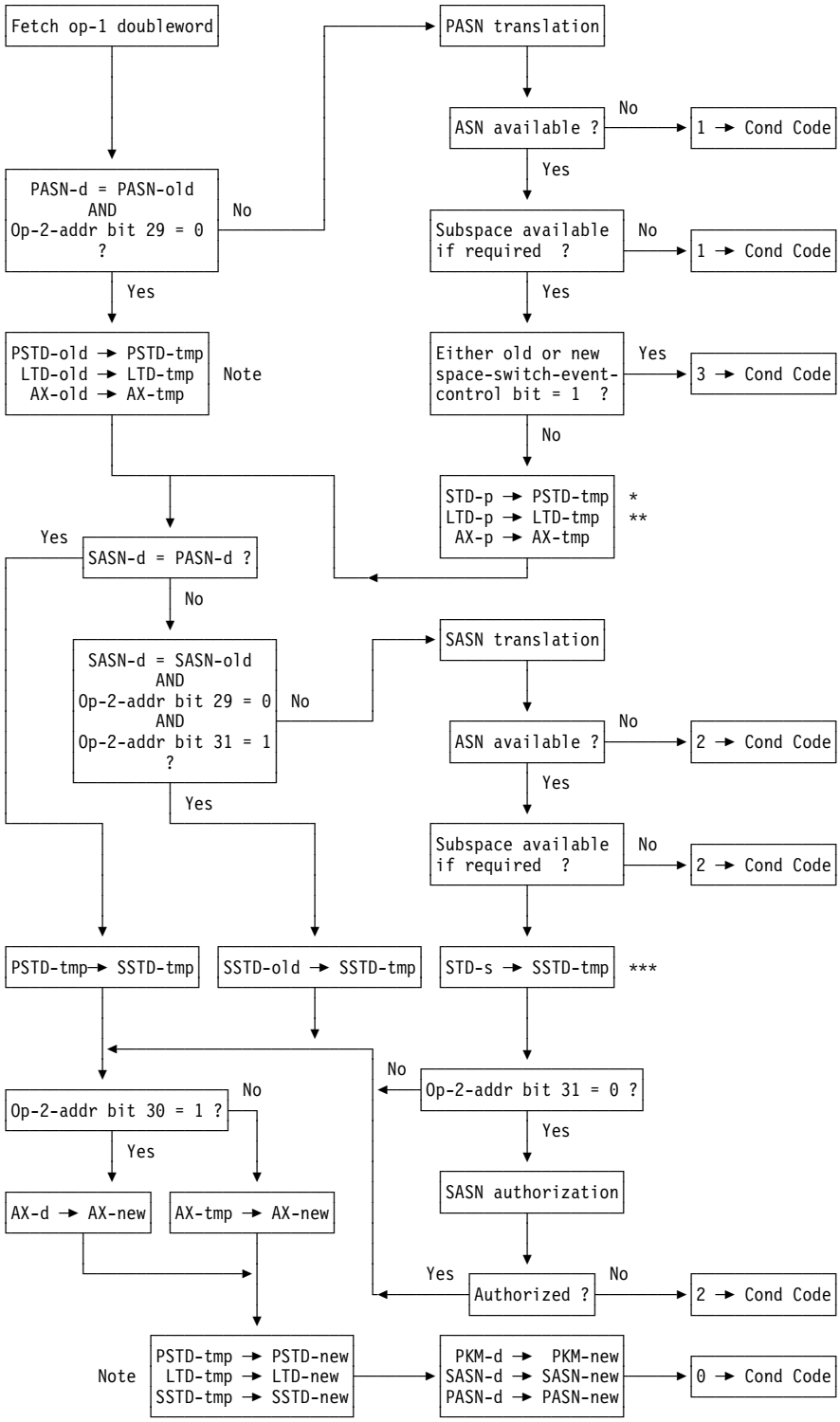
- \- The field is not used in this case.

[1] STD-rp is formed from STD-p, and STD-rs is formed from STD-s, by a subspace-replacement operation.

[2] LTD-p is accessed only when the ASF control is zero. When the ASF control is one, PASTEO-p is used in the operation, and it is bits 1-25 of the address of the ASN-second-table entry.

*Figure 10-14. Summary of Abbreviations for LOAD ADDRESS SPACE PARAMETERS*


LzLabs GmbH. Ex. 1008-515




*: PSTD-tmp is STD-rp if subspace replacement occurred.
**: Replace "LTD" with "PASTEO" when the ASF control is one.
***: SSTD-tmp is STD-rs if subspace replacement occurred.

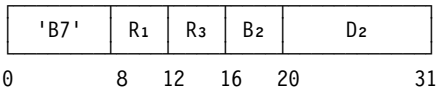
*Figure 10-15. Execution of LOAD ADDRESS SPACE PARAMETERS*



LzLabs GmbH. Ex. 1008-516

## LOAD CONTROL

LCTL R1,R3,D2(B2) [RS]




The set of control registers starting with control register $R_1$ and ending with control register $R_3$ is loaded from the locations designated by the second-operand address.

The storage area from which the contents of the control registers are obtained starts at the location designated by the second-operand address and continues through as many storage words as the number of control registers specified. The control registers are loaded in ascending order of their register numbers, starting with control register $R_1$ and continuing up to and including control register $R_3$, with control register 0 following control register 15. The second operand remains unchanged.

The information loaded into the control registers becomes active when instruction execution has ended.

**Special Conditions**

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2)
- Privileged operation
- Specification

**Programming Notes:**

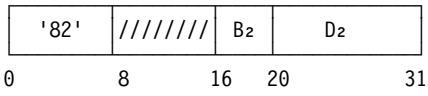
1. To ensure that existing programs operate correctly if and when new facilities using additional control-register positions are defined, only zeros should be loaded in unassigned control-register positions.
2. Loading of control registers on some models may require a significant amount of time. This is particularly true for changes in significant parameters.

   For example, the TLB may be cleared of entries as a result of changing or enabling the program-event-recording parameters in control registers 9-11. Where possible, the program should avoid unnecessary loading of control registers. In loading control registers 9-11, most models attempt to optimize for the case when the bits of control register 9 are zeros.

   As another example, the translation format, bits 8-12 of control register 0, is initialized to all zeros by initial CPU reset. An all-zero value is an invalid translation format, and, on some models, results in purging the TLB even though DAT may be off. Thus, the program should avoid loading invalid values for this field.

## LOAD PSW

LPSW D2(B2) [S]




The current PSW is replaced by the contents of the doubleword at the location designated by the second-operand address.

Bits 8-15 of the instruction are ignored.

A serialization and checkpoint-synchronization function is performed before or after the operand is fetched and again after the operation is completed.

**Special Conditions**

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The value which is to be loaded by the instruction is not checked for validity before it is loaded. However, immediately after loading, a specification exception is recognized and a program interruption occurs when any of the following is true for the newly loaded PSW:

- A one is introduced into an unassigned bit position of the PSW (that is, any of bit positions 0, 2-4, or 24-31).


LzLabs GmbH. Ex. 1008-517

- A zero is introduced into bit position 32 of the PSW, but bits 33-39 are not all zeros.
- A zero is introduced into bit position 12 of the PSW.

In these cases, the operation is completed, and the resulting instruction-length code is zero.

The test for a specification exception after the PSW is loaded is described in "Early Exception Recognition" on page 6-9. It may be considered as occurring early in the process of preparing to execute the subsequent instruction.

The operation is suppressed on all addressing and protection exceptions.

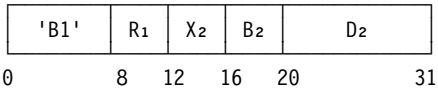
***Resulting Condition Code:*** The code is set as specified in the new PSW loaded.

***Program Exceptions:***

- Access (fetch, operand 2)
- Privileged operation
- Specification

## LOAD REAL ADDRESS

LRA $R_1$,$D_2$($X_2$,$B_2$) [RX]




The real address corresponding to the second-operand virtual address is placed in general register $R_1$.

The virtual address specified by the $X_2$, $B_2$, and $D_2$ fields is translated by means of the dynamic-address-translation facility, regardless of whether DAT is on or off.

DAT is performed by using a segment-table designation that depends on the current value of the address-space-control bits, bits 16 and 17 of the PSW, as shown in the following table:

| PSW Bits 16 and 17 | Segment-Table Designation Used by DAT |
|---|---|
| 00 | Contents of control register 1 |
| 10 | Contents of control register 7 |
| 01 | The segment-table designation obtained by applying the access-register-translation (ART) process to the access register designated by the $B_2$ field |
| 11 | Contents of control register 13 |

ART may be performed with the use of the ART-lookaside buffer (ALB).

On a model without z/Architecture installed, DAT is performed without the use of the translation-lookaside buffer (TLB). If z/Architecture is installed, the TLB may be used.

A zero is appended on the left of the 31-bit real address resulting from DAT to produce a 32-bit result, which is then placed in general register $R_1$. The translated address is not inspected for boundary alignment or for addressing or protection exceptions.

The virtual-address computation is performed according to the current addressing mode, specified by bit 32 of the current PSW.

The addresses of the segment-table entry and page-table entry are treated as 31-bit addresses, regardless of the current addressing mode specified by bit 32 of the current PSW. It is unpredictable whether the addresses of these entries are treated as real or absolute addresses.

Condition code 0 is set when both ART, if applicable, and DAT can be completed, that is, when a segment-table designation can be obtained and the entry in each DAT table lies within the specified table length and has a zero I bit.

When PSW bits 16 and 17 are 01 binary and a segment-table designation cannot be obtained because of a situation that would normally cause one of the exceptions shown in the following table, (1) the interruption code assigned to the exception is placed in bit positions 16-31 of general register $R_1$, and bit 0 of this register is set to one and bits



LzLabs GmbH. Ex. 1008-518

1-15 are set to zeros, and (2) the instruction is completed by setting condition code 3.

| Exception Name | Cause | Code (Hex) |
|---|---|---|
| ALET specification | Access-list-entry-token (ALET) bits 0-6 not all zeros | 0028 |
| ALEN translation | Access-list entry (ALE) outside list or invalid (bit 0 is one) | 0029 |
| ALE sequence | ALE sequence number (ALESN) in ALET not equal to ALESN in ALE | 002A |
| ASTE validity | ASN-second-table entry (ASTE) invalid (bit 0 is one) | 002B |
| ASTE sequence | ASTE sequence number (ASTESN) in ALE not equal to ASTESN in ASTE | 002C |
| Extended authority | ALE private bit not zero, ALE authorization index (ALEAX) not equal to extended authorization index (EAX), and secondary bit selected by EAX either outside authority table or zero | 002D |

When ART is completed normally, the operation is continued through the performance of DAT.

When the I bit in the segment-table entry is one, condition code 1 is set, and the real or absolute address of the segment-table entry is placed in general register $R_1$. When the I bit in the page-table entry is one, condition code 2 is set, and the real or absolute address of the page-table entry is placed in general register $R_1$. When either the segment-table entry or the page-table entry is outside the table, condition code 3 is set, and general register $R_1$ is loaded with the real or absolute address of the entry that would have been fetched if the length violation had not occurred. In all these cases, the address placed in general register $R_1$ is real or absolute in accordance with the type of address that was used during the attempted translation, a zero is appended on the left of the resultant 31-bit address to produce a 32-bit result, and the 32-bit result is placed in the register.

**Special Conditions**

An addressing exception is recognized when the address used by ART to fetch the effective access-list designation or the ALE, ASTE, or authority-table entry designates a location which is not available in the configuration. When it is necessary to access the authority table — when the private bit is not zero and the ALEAX is not equal to the EAX — an ASN-translation-specification exception may be recognized when bits 30, 31, and 60-63 of the ASTE are not all zeros.

An addressing exception is recognized when the address used to fetch the segment-table entry or page-table entry designates a location which is not available in the configuration. A translation-specification exception is recognized when bits 8-12 of control register 0 contain an invalid code or the segment-table entry or page-table entry has a zero I bit and a format error, that is, when any of the reasons listed in “Translation-Specification Exception” on page 6-34 applies.

A carry into bit position 0 as a result of the addition done to compute the address of either the segment-table entry or the page-table entry may be ignored or may result in an addressing exception.

The operation is suppressed on all addressing exceptions.

***Resulting Condition Code:***

0 Translation available
1 Segment-table entry invalid (I bit is one)
2 Page-table entry invalid (I bit is one)
3 Segment-table designation not available or segment- or page-table length exceeded

***Program Exceptions:***

- Addressing (effective access-list designation, access-list entry, ASN-second-table entry, authority-table entry, segment-table entry, or page-table entry)
- ASN-translation specification
- Privileged operation
- Translation specification

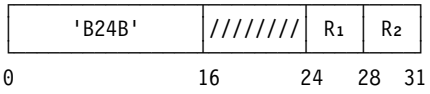
**Programming Note:** Caution must be exercised in the use of LOAD REAL ADDRESS in a multiprocessing configuration. Since INVALIDATE PAGE TABLE ENTRY may set the I bit in storage to one before causing the corresponding entries in


LzLabs GmbH. Ex. 1008-519

TLBs of other CPUs to be cleared, the simultaneous execution of LOAD REAL ADDRESS on this CPU and INVALIDATE PAGE TABLE ENTRY on another CPU may produce inconsistent results. Because LOAD REAL ADDRESS may access the tables in storage, the page-table entry may appear to be invalid (condition code 2) even though the corresponding TLB entry has not yet been cleared, and the TLB entry may remain in the TLB until the completion of INVALIDATE PAGE TABLE ENTRY on the other CPU. There is no guaranteed limit to the number of instructions which may be executed between the completion of LOAD REAL ADDRESS and the TLB being cleared of the entry.

## LOAD USING REAL ADDRESS

```
LURA     R1,R2     [RRE]
```




The word at the real-storage location addressed by the contents of general register $R_2$ is placed in general register $R_1$.

Bits 16-23 of the instruction are ignored.

In the 24-bit addressing mode, bits 8-31 of general register $R_2$ designate a real-storage location on a word boundary, and bits 0-7 of the register are ignored. In the 31-bit addressing mode, bits 1-31 of general register $R_2$ designate a real-storage location on a word boundary, and bit 0 of the register is ignored.

Because it is a real address, the address designating the storage word is not subject to dynamic address translation.

**Special Conditions**

The contents of general register $R_2$ must designate a location on a word boundary; otherwise, a specification exception is recognized.

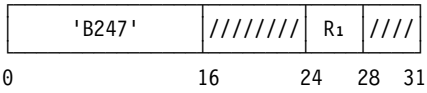
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (address specified by general register $R_2$)
- Privileged operation
- Protection (fetch, operand 2, key-controlled protection)
- Specification

## MODIFY STACKED STATE

```
MSTA     R1     [RRE]
```




The contents of the pair of general registers designated by the $R_1$ field are placed in the modifiable area, byte positions 152-159, of the last state entry in the linkage stack.

The $R_1$ field designates the even-numbered register of an even-odd pair of general registers.

The last state entry is located as described in “Unstacking Process” on page 5-75. The state entry remains in the linkage stack, and the linkage-stack-entry address in control register 15 remains unchanged.

Key-controlled protection does not apply to the references to the linkage stack, but low-address and page protection do apply.

Bits 16-23 and 28-31 of the instruction are ignored.

**Special Conditions**

A specification exception is recognized when $R_1$ is odd.

The CPU must be in the primary-space mode, access-register mode, or home-space mode, and the address-space-function control, bit 15 of control register 0, must be one; otherwise, a special-operation exception is recognized.

A stack-empty, stack-specification, or stack-type exception may be recognized during the unstacking process.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-16 on page 10-41.



LzLabs GmbH. Ex. 1008-520

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch and store, except for key-controlled protection, linkage-stack entry)
- Special operation
- Specification
- Stack empty
- Stack specification
- Stack type

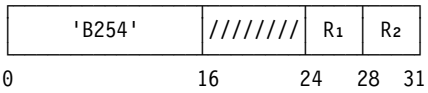
| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to the CPU being in the real mode or secondary-space mode or the address-space-function control, bit 15 of control register 0, being zero. |
| 8.A | Specification exception due to $R_1$ being odd. |
| 8.B.1 | Access exceptions for entry descriptor of the current linkage-stack entry. |
| 8.B.2 | Stack-type exception due to current entry not being a state entry or header entry.<br><br>**Note:** Exceptions 8.B.3-8.B.7 can occur only if the current entry is a header entry. |
| 8.B.3 | Access exceptions for second word of the header entry. |
| 8.B.4 | Stack-empty exception due to backward stack-entry validity bit in the header entry being zero. |
| 8.B.5 | Access exceptions for entry descriptor of preceding entry, which is the entry designated by the backward stack-entry address in the current (header) entry. |
| 8.B.6 | Stack-specification exception due to preceding entry being a header entry. |
| 8.B.7 | Stack-type exception due to preceding entry not being a state entry. |
| 8.B.8 | Access exceptions for the modifiable area of the state entry. |

*Figure 10-16. Priority of Execution: MODIFY STACKED STATE*


LzLabs GmbH. Ex. 1008-521

## MOVE PAGE (Facility 2)

```
MVPG     R1,R2     [RRE]
```




This definition applies if move-page facility 2 is installed. The MOVE PAGE instruction of move-page facility 1 is defined in Chapter 7, “General Instructions.”

The first operand is replaced by the second operand. The first and second operands both are 4K bytes on 4K-byte boundaries. The results are indicated in the condition code. The accesses to the first-operand location or the second-operand location, but not to both locations, may be performed by using the key specified in general register 0; otherwise, the accesses to an operand location are performed by using the PSW key.

Bits 16-23 of the instruction are ignored.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers $R_1$ and $R_2$, respectively.

The handling of the addresses in general registers $R_1$ and $R_2$ depends on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 8-19 of a general register, with 12 rightmost zeros appended, are the address, and bits 0-7 and 20-31 in the register are ignored. In the 31-bit addressing mode, the contents of bit positions 1-19 of a general register, with 12 rightmost zeros appended, are the address, and bits 0 and 20-31 in the register are ignored.

Bits 24-27 of general register 0 are used as the specified access key. Bit 20 of general register 0, when one, specifies that the specified access key is to be used for accessing the first operand, and bit 21 specifies the same for the second operand. A specification exception is recognized if bits 20 and 21 are both ones. Bit 22 of general register 0 is a destination-reference-intention bit, and bit 23 is a condition-code-option bit. Bits 16-19 of general register 0 must be zeros; otherwise, a specification exception is recognized. Bits 0-15 and 28-31 of general register 0 are ignored.

The contents of the registers just described are shown in Figure 10-17 on page 10-43

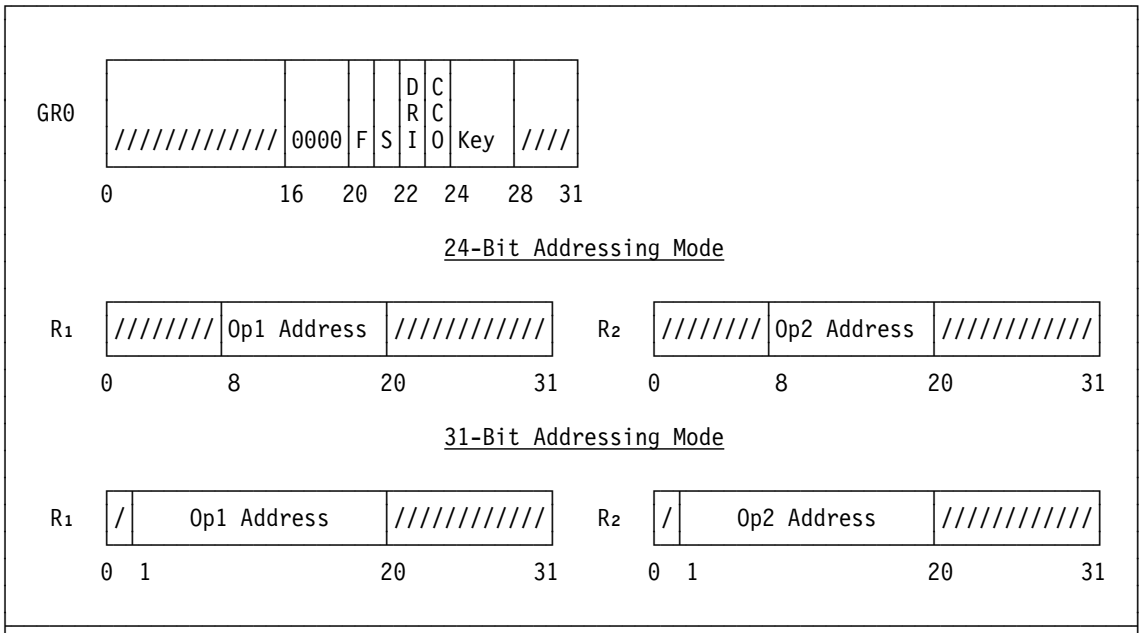
When bit 20 of general register 0 is one, the fetch accesses to the second-operand location are performed by using the PSW key, and the store accesses to the first-operand location are performed by using the key specified in general register 0. When bit 21 of general register 0 is one, the fetch accesses to the second-operand location are performed by using the key specified in general register 0, and the store accesses to the first-operand location are performed by using the PSW key. When bits 20 and 21 are both zeros, the PSW key is used for accessing both operands.

When DAT is on and the page-invalid bit is one in the page-table entry for an operand, additional address translation is performed to determine whether the operand is valid in expanded storage. As a result, the replacement of the first operand by the second operand may be performed by moving data from main storage to main storage, from main storage to expanded storage, or from expanded storage to main storage, depending on whether and where the operands are valid. When 4K bytes have been moved, condition code 0 is set.

Data movement is prevented if a page-translation-exception condition exists. A page-translation-exception condition exists (1) for an operand if either the page-table entry for the operand is outside the page table or the operand is invalid in both main storage and expanded storage; (2) for the first operand if both operands are valid in expanded storage; (3) for the first operand if the first operand is valid in expanded storage, the second operand is valid in main storage, and the destination-reference-intention bit, bit 22 in general register 0, is one; and (4) for an operand if the operand is valid in expanded storage, but the translation path for the expanded-storage operand is locked or the expanded-storage block containing that operand either is not available or causes an expanded-storage data error. provided that both operands are not valid in expanded storage. When both operands are invalid in both main storage and expanded storage, a page-translation-exception condition exists for the second operand. When a page-translation-exception condition exists because of an expanded-storage data error, the contents of the first-operand location are unpredictable, and



LzLabs GmbH. Ex. 1008-522




**Explanation:**

CCO Condition-code-option bit.
DRI Destination-reference-intention bit.
F When one, specified access key applies to first operand. A specification exception is recognized if F and S both are one.
Key Specified access key.
S When one, specified access key applies to second operand. A specification exception is recognized if F and S both are one.

*Figure 10-17. Register Contents for MOVE PAGE of Move-Page Facility 2*

the instruction ending is not true nullification in this case.

When a page-translation-exception condition exists as described in the preceding paragraph, except when the condition is that the page-table entry is outside the page table, the exception is not recognized if the condition-code-option bit, bit 23 in general register 0, is one; instead, condition code 1 or 2 is set. Condition code 1 is set in all cases, except that condition code 2 is set if the second operand is invalid in both main storage and expanded storage, regardless of the validity of the first operand.

When an access exception can be recognized for both operands, it is unpredictable for which operand an exception is recognized. If one of the exceptions is a page-translation exception that would cause condition code 1 or 2 to be set, it is unpredictable whether the access exception for the other operand is recognized or condition code 1 or 2 is set.

When data is moved to or from expanded storage, access-list-controlled, page, and key-controlled protection apply, and it is unpredictable whether low-address protection applies. The protection mechanisms apply to main storage in the normal way.

When the first operand is valid in main storage and the second operand is valid in expanded storage, but the expanded-storage block containing the second operand is unavailable, a storage-alteration PER event may be recognized, and the change bit may be set, for the first operand even though the first-operand location remains unchanged.

**Operation in a Multiple-CPU Configuration**

The references to main storage and to expanded storage are not necessarily single-access references and are not necessarily performed in a left-


LzLabs GmbH. Ex. 1008-523

to-right direction, as observed by other CPUs and by channel programs.

If two or more CPUs move data to or from expanded storage at approximately the same instant, depending on the model, the operations may be performed one at a time, or the operations may be performed concurrently. Concurrent operation may occur even if the instructions address the same expanded-storage block.

When two or more CPUs move data to the same expanded-storage block concurrently, the resulting values in the expanded-storage block for each group of bytes transferred may be from any of the instructions being executed simultaneously. The number of bytes transferred as a group is unpredictable.

Similarly, for concurrent movement to and from the same expanded-storage block, the resulting values for each group of bytes moved from expanded storage may be either the old or the new values from the expanded-storage block.

When data movement is due to occur between main storage and expanded storage, the translation path being used for the expanded-storage operand is set to the locked state. When this data movement is completed successfully, or when a page-translation exception is due to be recognized or condition code 1 is due to be set because the movement cannot be completed successfully, the translation path is set to the unlocked state.

**Special Conditions**

In the problem state, when either bit 20 or bit 21 in general register 0 is one, the operation is performed only if the access key specified in general register 0 is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the specified access key is valid. When bits 20 and 21 are both zeros, the access key in general register 0 is not tested for validity.

In the problem state, when bits 20 and 21 in general register 0 are both ones and the access key in general register 0 is not permitted by the PSW-key mask, it is unpredictable whether a specification exception or a privileged-operation exception is recognized.

***Resulting Condition Code:***

0 Data moved
1 Condition-code-option bit one and (1) first operand invalid and second operand valid; (2) both operands valid in expanded storage; (3) first operand valid in expanded storage, second operand valid in main storage, and destination-reference-intention bit one; (4) translation path locked; (5) expanded-storage block unavailable, or (6) expanded-storage data error
2 Condition-code-option bit one and second operand invalid
3 --

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1, except low-address protection for operand in expanded storage is unpredictable)
- Privileged operation (access key specified, and selected PSW-key-mask bit is zero in the problem state)
- Specification

**Programming Notes:**

1. MOVE PAGE, or a loop of MOVE PAGE instructions that moves multiple pages, may provide, on most models, better performance than a MOVE LONG instruction or a loop of MOVE (MVC) instructions that performs the same function. Whether MOVE PAGE provides better performance depends on control-program specifications and the method by which the control program handles page-translation exceptions.
2. The destination-reference-intention bit should be set to one when there is an intention to reference the first operand by means of an instruction other than MOVE PAGE. The effect when the bit is one is to allow the control program to assign a page frame of real storage to the first operand, without a movement of data having first been performed to the first-operand location in expanded storage.
3. The condition-code-option bit provides compatibility with the MOVE PAGE instruction of move-page facility 1. The bit is for use by the MVS/ESA HSPSERV macro expansion.

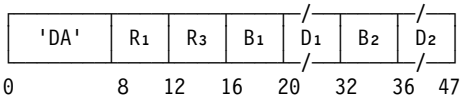


LzLabs GmbH. Ex. 1008-524

4. The condition code set by the instruction normally need not be examined if the condition-code-option bit is zero.
5. Since an expanded-storage location may be accessed by means of more than one translation path or even without translation, the locked state of a translation path does not necessarily prevent concurrent accesses to the location by different processes. To ensure predictable results when data is in either main storage or expanded storage, the program must use a programmed lock to prevent different processes from performing concurrent store accesses or concurrent fetch and store accesses to the same location.
6. Monitoring for PER storage-alteration events is done using logical addresses. Thus, it applies to the operands of MOVE PAGE regardless of whether the operands are in main storage or expanded storage.
7. See the definitions of real locations 144-147 and 162 under “Assigned Storage Locations” in Chapter 3, “Storage,” for a description of information stored during a program interruption due to a page-translation exception recognized by MOVE PAGE.

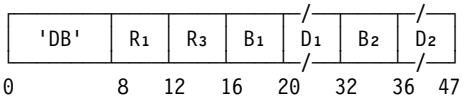
## MOVE TO PRIMARY

MVCP $D_1(R_1,B_1),D_2(B_2),R_3$ [SS]




## MOVE TO SECONDARY

MVCS $D_1(R_1,B_1),D_2(B_2),R_3$ [SS]




The first operand is replaced by the second operand. One operand is in the primary address space, and the other is in the secondary address space. The accesses to the operand in the primary space are performed by using the PSW key; the accesses to the operand in the secondary space are performed by using the key specified by the third operand.

The addresses of the first and second operands are virtual, one operand address being translated by means of the primary segment-table designation and the other by means of the secondary segment-table designation. Operand-address translation is performed in the same way when the address-space-control bits in the current PSW specify either the primary-space mode or the secondary-space mode.

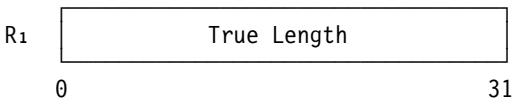
For MOVE TO PRIMARY, movement is to the primary space from the secondary space. The first-operand address is translated by using the primary segment table, and the second-operand address is translated by using the secondary segment table.

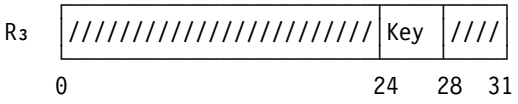
For MOVE TO SECONDARY, movement is to the secondary space from the primary space. The first-operand address is translated by using the secondary segment table, and the second-operand address is translated by using the primary segment table.

Bit positions 24-27 of general register $R_3$ are used as the secondary-space access key. Bit positions 0-23 and 28-31 of the register are ignored.

The contents of general register $R_1$ are a 32-bit unsigned value called the true length.

The contents of the general registers just described are as follows:




The first and second operands are the same length, called the effective length. The effective length is equal to the true length or 256, whichever is less. Access exceptions for the first and second operands are recognized only for that portion of the operand within the effective length. When the effective length is zero, no access exceptions are recognized for the first and second operands, and no movement takes place.


LzLabs GmbH. Ex. 1008-525

Each storage operand is processed left to right. The storage-operand-consistency rules are the same as for MOVE (MVC), except that when the operands overlap in real storage, the use of the common real-storage locations is not necessarily recognized.

As part of the execution of the instruction, the value of the true length is used to set the condition code. If the true length is 256 or less, including zero, the true length and effective length are equal, and condition code 0 is set. If the true length is greater than 256, the effective length is 256, and condition code 3 is set.

For both MOVE TO PRIMARY and MOVE TO SECONDARY, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

**Special Conditions**

Since the secondary space is accessed, the operation is performed only when the secondary-space control, bit 5 of control register 0, is one and DAT is on. When either the secondary-space control is zero or DAT is off, a special-operation exception is recognized. A special-operation exception is also recognized when the address-space-control bits in the current PSW specify the access-register or home-space mode.

In the problem state, the operation is performed only if the secondary-space access key is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the secondary-space access key is valid.

The priority of the recognition of exceptions and condition codes is shown in Figure 10-18.

***Resulting Condition Code:***

0 True length less than or equal to 256
1 --
2 --
3 True length greater than 256

***Program Exceptions:***

- Access (fetch, primary virtual address, operand 2, MVCS; fetch, secondary virtual address, operand 2, MVCP; store, secondary virtual address, operand 1, MVCS; store, primary virtual address, operand 1, MVCP)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)
- Special operation

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second and third instruction halfwords. |
| 7.B | Special-operation exception due to the secondary-space control, bit 5 of control register 0, being zero, to DAT being off, or to the CPU being in the access-register or home-space mode. |
| 8. | Privileged-operation exception due to selected PSW-key-mask bit being zero in the problem state. |
| 9. | Completion due to length zero. |
| 10. | Access exceptions for operands. |

*Figure 10-18. Priority of Execution: MOVE TO PRIMARY and MOVE TO SECONDARY*

**Programming Notes:**

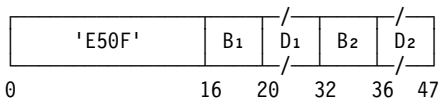
1. MOVE TO PRIMARY and MOVE TO SECONDARY can be used in a loop to move a variable number of bytes of any length. See the programming note under MOVE WITH KEY.
2. MOVE TO PRIMARY and MOVE TO SECONDARY should be used only when movement is between different address spaces. The performance of these instructions on most models may be significantly slower than that of MOVE WITH KEY, MOVE (MVC), or MOVE LONG. In addition, the definition of overlapping operands for MOVE TO PRIMARY and MOVE TO SECONDARY is not compatible with the more precise definitions for MOVE (MVC), MOVE WITH KEY, and MOVE LONG.



LzLabs GmbH. Ex. 1008-526

## MOVE WITH DESTINATION KEY

MVCDK D1(B1),D2(B2) [SSE]

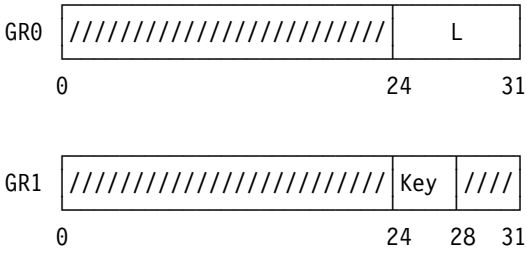



The first operand is replaced by the second operand. The accesses to the destination-operand location are performed by using the key specified in general register 1, and the accesses to the source-operand location are performed by using the PSW key.

The first and second operands are of the same length, which is specified by bits 24-31 of general register 0. Bits 0-23 of general register 0 are ignored.

Bits 24-27 of general register 1 are used as the specified access key. Bits 0-23 and 28-31 of general register 1 are ignored.

The contents of general registers 0 and 1 are as follows:




L specifies the number of bytes to the right of the first byte of each operand. Therefore, the length in bytes of each operand is 1-256, corresponding to a length code in L of 0-255.

The fetch accesses to the second-operand location are performed by using the PSW key, and the store accesses to the first-operand location are performed by using the key specified in general register 1.

Each of the operands is processed left to right. When the operands overlap destructively in real storage, the results in the first-operand location are unpredictable. Except for this unpredictability in the case of destructive overlap, the storage-operand-consistency rules are the same as for the MOVE (MVC) instruction.

**Special Conditions**

In the problem state, the operation is performed only if the access key specified in general register 1 is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the specified access key is valid.

***Condition Code:*** The code remains unchanged.

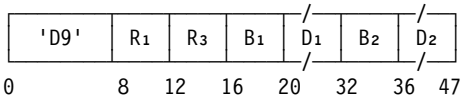
***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)

**Programming Note:** See the programming notes for the MOVE WITH SOURCE KEY instruction.

## MOVE WITH KEY

MVCK D1(R1,B1),D2(B2),R3 [SS]

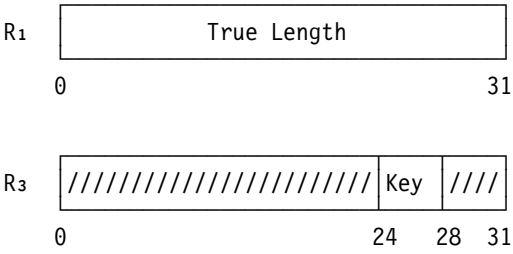



The first operand is replaced by the second operand. The fetch accesses to the second-operand location are performed by using the key specified in the third operand, and the store accesses to the first-operand location are performed by using the PSW key.

Bit positions 24-27 of general register R3 are used as the source access key. Bit positions 0-23 and 28-31 of the register are ignored.

The contents of general register R1 are a 32-bit unsigned value called the true length.

The contents of the general registers just described are as follows:





LzLabs GmbH. Ex. 1008-527

The first and second operands are of the same length, called the effective length. The effective length is equal to the true length or 256, whichever is less. Access exceptions for the first and second operands are recognized only for that portion of the operand within the effective length. When the effective length is zero, no access exceptions are recognized for the first and second operands, and no movement takes place.

Each storage operand is processed left to right. When the storage operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte was fetched. The storage-operand-consistency rules are the same as for the MOVE (MVC) instruction.

As part of the execution of the instruction, the value of the true length is used to set the condition code. If the true length is 256 or less, including zero, the true length and effective length are equal, and condition code 0 is set. If the true length is greater than 256, the effective length is 256, and condition code 3 is set.

**Special Conditions**

In the problem state, the operation is performed only if the source access key is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the source access key is valid.

The priority of the recognition of exceptions and condition codes is shown in Figure 10-19.

***Resulting Condition Code:***

0 True length less than or equal to 256
1 --
2 --
3 True length greater than 256

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second and third instruction halfwords. |
| 8. | Privileged-operation exception due to selected PSW-key-mask bit being zero in the problem state. |
| 9. | Completion due to length zero. |
| 10. | Access exceptions for operands. |

*Figure 10-19. Priority of Execution: MOVE WITH KEY*

**Programming Notes:**

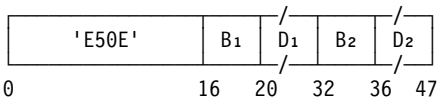
1. MOVE WITH KEY can be used in a loop to move a variable number of bytes of any length, as follows:

```
LOOP    MVCK   D1(R1,B1),D2(B2),R3
        BC     8,END
        AHI    B1,256
        AHI    B2,256
        AHI    R1,-256
        B      LOOP
END     [Any instruction]
```

2. The performance of MOVE WITH KEY on most models may be significantly slower than that of the MOVE (MVC) and MOVE LONG instructions. Therefore, MOVE WITH KEY should not be used if the keys of the source and the target are the same.

# MOVE WITH SOURCE KEY

```
MVCSK     D1(B1),D2(B2)      [SSE]
```




The first operand is replaced by the second operand. The accesses to the source-operand location are performed by using the key specified in general register 1, and the accesses to the destination-operand location are performed by using the PSW key.

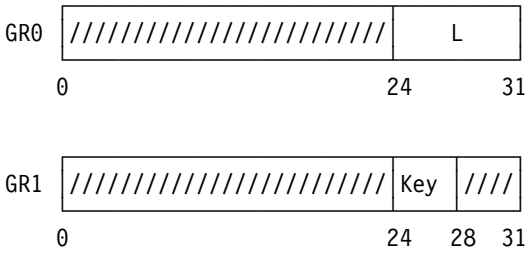
The first and second operands are of the same length, which is specified by bits 24-31 of general register 0. Bits 0-23 of general register 0 are ignored.



LzLabs GmbH. Ex. 1008-528

Bits 24-27 of general register 1 are used as the specified access key. Bits 0-23 and 28-31 of general register 1 are ignored.

The contents of general registers 0 and 1 are as follows:




L specifies the number of bytes to the right of the first byte of each operand. Therefore, the length in bytes of each operand is 1-256, corresponding to a length code in L of 0-255.

The fetch accesses to the second-operand location are performed by using the key specified in general register 1, and the store accesses to the first-operand location are performed by using the PSW key.

Each of the operands is processed left to right. When the operands overlap destructively in real storage, the results in the first-operand location are unpredictable. Except for this unpredictability in the case of destructive overlap, the storage-operand-consistency rules are the same as for the MOVE (MVC) instruction.

**Special Conditions**

In the problem state, the operation is performed only if the access key specified in general register 1 is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the specified access key is valid.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2; store, operand 1)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)

**Programming Notes:**

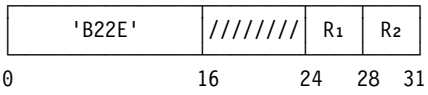
1. When data is to be moved alternately in both directions between two storage areas that are fetch protected by means of different keys, then MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY can be used while leaving the PSW key unchanged; and this may be, on most models, significantly faster than using MOVE WITH KEY along with SET PSW KEY FROM ADDRESS to change the PSW key.
2. MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY should be used only when movement is between storage areas having different keys. The performance of these instructions on most models may be significantly slower than that of the MOVE (MVC) instruction.
3. MOVE WITH SOURCE KEY or MOVE WITH DESTINATION KEY can be used in a loop to move a variable number of bytes as shown in the following example. In the example, the specified access key, the first-operand address, the second-operand address, and the length of each operand are assumed to be in general registers 1-4, respectively, at the beginning of the example. The length of each operand is treated as a 32-bit signed value, and a negative value is treated as zero.

```
        LTR    4,4
        BC     12,END
        AHI    4,-256
        BC     12,LAST
        LA     0,255
LOOP    MVCSK  0(2),0(3)
        LA     2,256(2)
        LA     3,256(3)
        AHI    4,-256
        BC     2,LOOP
LAST    LA     0,255(4)
        MVCSK  0(2),0(3)
END     [Any instruction]
```


LzLabs GmbH. Ex. 1008-529

# PAGE IN

```
PGIN   R1,R2              [RRE]
```




A page-in operation is performed which transfers a 4K-byte block to the real-storage location designated by general register $R_1$ from the expanded-storage block designated by general register $R_2$.

Bits 16-23 of the instruction are ignored.

The contents of general register $R_2$ are a 32-bit unsigned binary integer called the expanded-storage-block number. This number designates the 4K-byte block of expanded storage which is to be transferred. If the expanded-storage-block number designates an inaccessible block in expanded storage, condition code 3 is set.

The contents of general register $R_1$ are a real address which designates a 4K-byte block in main storage. In the 24-bit-addressing mode, bits 8-19 designate the block, and bits 0-7 are ignored. In the 31-bit-addressing mode, bits 1-19 designate the block, and bit 0 is ignored. In both modes, bits 20-31 of the address are ignored.

Because it is a real address, the address designating the main-storage block is not subject to dynamic address translation. PAGE IN is not subject to key-controlled storage protection, but low-address protection does apply. PAGE IN is not subject to program-event recording for storage alteration.

It is unpredictable whether the page-in operation causes change-bit action. However, if the asynchronous-pageout facility is installed, PAGE IN sets the change bit and reference bit to one.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

If the page-in operation is completed with no errors, condition code 0 is set.

If the page-in operation encounters an expanded-storage data error, condition code 1 is set. For an expanded-storage data-error condition, the contents of the entire 4K-byte block in real storage is unpredictable, but this condition does not result in the generation of invalid checking-block codes in real storage.

If the expanded-storage block is not available, that is, the block is not provided or is not currently in the configuration, then condition code 3 is set, and no other action is taken.

**Operation of PAGE IN in a Multiple-CPU Configuration**

The accesses to main storage and to expanded storage by PAGE IN are not necessarily single-access references and are not necessarily performed in a left-to-right direction, as observed by other CPUs and by channel programs.

See also the description under PAGE OUT.

***Resulting Condition Code:***

0 Page-in operation completed
1 Expanded-storage data error
2 --
3 Expanded-storage block not available

***Program Exceptions:***

- Addressing (block designated by general register $R_1$)
- Operation (if the expanded-storage facility is not installed)
- Privileged operation
- Protection (block designated by general register $R_1$; low-address protection)

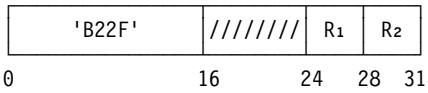
**Programming Note:** The fact that it is unpredictable whether the page-in operation performs change-bit action is usually not a problem because the normal action after executing PAGE IN is to set the storage key with the change bit zero. However, when PAGE IN is being simulated by a host program on behalf of the guest, special precautions must be taken in order for the host not to lose track of the fact that the page has been changed.



LzLabs GmbH. Ex. 1008-530

# PAGE OUT

PGOUT $R_1$,$R_2$ [RRE]




A page-out operation is performed which transfers a 4K-byte block from the real-storage location designated by general register $R_1$ to the expanded-storage block designated by general register $R_2$.

Bits 16-23 of the instruction are ignored.

The contents of general register $R_2$ are a 32-bit unsigned binary integer called the expanded-storage-block number. This number designates the 4K-byte block of expanded storage which is to be replaced. If the expanded-storage-block number designates an inaccessible block in expanded storage, condition code 3 is set.

The contents of general register $R_1$ are a real address which designates a 4K-byte block in main storage. In the 24-bit-addressing mode, bits 8-19 designate the block, and bits 0-7 are ignored. In the 31-bit-addressing mode, bits 1-19 designate the block, and bit 0 is ignored. In both modes, bits 20-31 of the address are ignored.

Because it is a real address, the address designating the main-storage block is not subject to dynamic address translation. PAGE OUT is not subject to key-controlled protection.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Depending on the model, after the data has been written to the expanded-storage block, a read-back-check operation may be performed to determine whether the data was written correctly. If the read-back-check operation determines that the data has been written correctly, condition code 0 is set. If the read-back-check operation encounters an expanded-storage data error, condition code 1 is set.

Most models do not perform the read-back-check operation, and, after the page-out operation is completed, condition code 0 is set.

Regardless of whether condition code 0 or condition code 1 is set, the entire 4K-byte block is written. Errors, if any, in the block after the block is written are preserved. Thus, if a subsequent execution of PAGE IN addresses the same expanded-storage block, the expanded-storage data error will be detected and condition code 1 will be indicated.

If the expanded-storage block is not available, that is, the block is not provided or is not currently in the configuration, then condition code 3 is set, and no other action is taken.

**Operation of PAGE OUT in a Multiple-CPU Configuration**

The accesses to main storage and to expanded storage by PAGE OUT are not necessarily single-access references and are not necessarily performed in a left-to-right direction, as observed by other CPUs and by channel programs.

If two or more CPUs issue PAGE IN or PAGE OUT instructions at approximately the same instant in time, depending on the model, the operations may be performed one at a time, or the operations may be performed concurrently. Concurrent operation may occur even if the instructions address the same expanded-storage block.

When two or more PAGE OUT instructions addressing the same expanded-storage block are executed concurrently, the resulting values in the expanded-storage block for each group of bytes transferred may be from any of the instructions being executed simultaneously. The number of bytes transferred as a group depends on the model.

Similarly, for concurrent execution of a PAGE IN and a PAGE OUT instruction for the same expanded-storage block, the resulting values for each group of bytes transferred as a result of the execution of the PAGE IN instruction may be either the old or new values from the expanded-storage block.

Concurrent operation of paging instructions does not result in expanded-storage data errors.

***Resulting Condition Code:***

0 Page-out operation completed


LzLabs GmbH. Ex. 1008-531

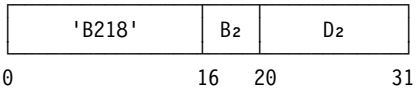
1 Expanded-storage data error
2 --
3 Expanded-storage block not available

***Program Exceptions:***

- Addressing (block designated by general register $R_1$)
- Operation (if the expanded-storage facility is not installed)
- Privileged operation

## PROGRAM CALL

PC D2(B2) [S]




When the program-call-fast facility is installed, the program-call-fast control, bit 28 of control register 0, is one, and the linkage-index (LX) part of the second-operand address has the value 31 (01F hex), it is unpredictable whether this definition or the PROGRAM CALL FAST definition applies. When any of those conditions is not true, this definition applies.

A program-call number specified by the second-operand address is used in a two-level lookup to locate an entry-table entry (ETE). When the address-space-function (ASF) control, bit 15 of control register 0, is zero, a 16-byte ETE is located; otherwise, when the ASF control is one, a 32-byte ETE is located.

The program is authorized to use the ETE when the AND of the PSW-key mask in control register 3 and the authorization key mask in the ETE is nonzero or when the CPU is in the supervisor state.

When a 16-byte ETE is located, or when a 32-byte ETE is located but the PC-type bit, bit 128 of the ETE, is zero, an operation called basic PROGRAM CALL is performed. When a 32-byte ETE is located and the PC-type bit is one, an operation called stacking PROGRAM CALL is performed.

Basic PROGRAM CALL loads the addressing-mode bit, updated instruction address, and problem-state bit from the PSW into general register 14, and it places the PSW-key mask and PASN in general register 3.

Stacking PROGRAM CALL places the entire PSW contents, except with an unpredictable PER mask, and also the PSW-key mask, PASN, SASN, and EAX in a linkage-stack program-call state entry that it forms. A called-space identification, the program-call number, and the contents of general registers 0-15 and access registers 0-15 also are placed in the state entry.

Basic and stacking PROGRAM CALL both replace the addressing-mode bit, instruction address, and problem-state bit in the PSW from the ETE, and both load the entry parameter from the ETE into general register 4.

Basic PROGRAM CALL ORs the entry key mask from the ETE into the PSW-key mask in control register 3. Stacking PROGRAM CALL does the same, or it replaces the PSW-key mask with the entry key mask, as determined by the PSW-key-mask control in the ETE.

Stacking PROGRAM CALL optionally replaces the PSW key in the PSW and the EAX in control register 8 from the ETE, and it sets the address-space-control bits in the PSW, as determined by control bits in the ETE.

The ETE causes a space-switching operation to occur if it contains a nonzero ASN. When the ETE contains a zero ASN, the operation is called PROGRAM CALL to current primary (PC-cp); when the ETE contains a nonzero ASN, the operation is called PROGRAM CALL with space switching (PC-ss). When space switching is specified, the new PASN is loaded into control register 4 from the ETE and is used in a two-level lookup to locate an ASN-second-table entry (ASTE). However, when the ASF control is one, the address of the ASTE may be obtained directly from the ETE. From this ASTE, a new PSTD and AX are loaded into control registers 1 and 4, respectively. When the ASF control is zero, a new LTD is loaded into control register 5 from the ASTE. When the ASF control is one, bits 1-25 of the address of the ASTE are loaded into control register 5 as the new primary-ASTE origin.

In both PC-cp and PC-ss, the SASN and SSTD are set equal to the original PASN and PSTD, respectively. However, the space-switching



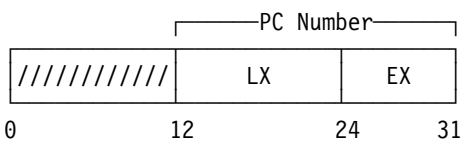
LzLabs GmbH. Ex. 1008-532

stacking PROGRAM CALL operation may instead set the SASN and SSTD equal to the new PASN and PSTD, respectively, as determined by a control bit in the ETE.

In a PC-ss to the base space of the dispatchable unit when the dispatchable unit is subspace active, bits 1-23 and 25-31 of the new PSTD are replaced by the same bits of the STD for the subspace. This occurs before the possible setting of the SSTD equal to the PSTD.

**PROGRAM CALL PC-Number Translation**

The second-operand address is not used to address data; instead, the rightmost 20 bits of the address are used as a PC number and have the following format:

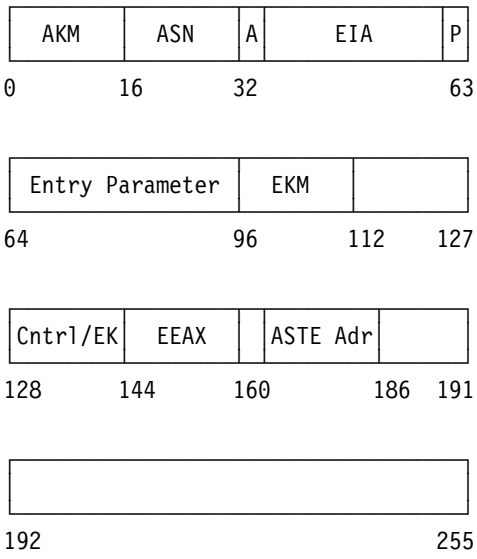



***Linkage Index (LX):*** Bits 12-23 of the second-operand address are the linkage index and are used to select an entry from the linkage table designated by the linkage-table designation. When the ASF control, bit 15 of control register 0, is zero, the linkage-table designation is in control register 5. When the ASF control is one, the linkage-table designation is in the primary ASN-second-table entry (primary ASTE), and the primary-ASTE origin is in control register 5.

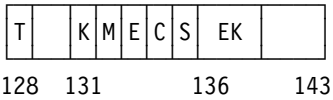
***Entry Index (EX):*** Bits 24-31 of the second-operand address are the entry index and are used to select an entry from the entry table designated by the linkage-table entry.

Bits 0-11 of the second-operand address are ignored.

The linkage-table and entry-table lookup process is depicted in part 1 of Figure 10-21 on page 10-59. The detailed definition of this table-lookup process is in “PC-Number Translation” on page 5-27. The 16-byte entry-table entry (ETE) is identical to the first 16 bytes of the 32-byte ETE. The 32-byte ETE has the following format:




Bits 128-143 of the ETE have the following detailed format:




When bit 32 of the ETE is zero (24-bit addressing mode), bits 33-39 of the ETE must be zeros; otherwise, a PC-translation-specification exception is recognized.

After the ETE has been fetched, if the current PSW specifies the problem state, the current PSW-key mask in control register 3 is tested against the AKM field in the ETE to determine whether the program is authorized to access this entry. The AKM and PSW-key mask are ANDed, and, if the result is zero, a privileged-operation exception is recognized. The PSW-key mask in control register 3 remains unchanged. When PROGRAM CALL is executed in the supervisor state, the AKM field is ignored.

If the result of the AND of the AKM and the PSW-key mask is not zero, or if the CPU is in the supervisor state, the execution of the instruction continues.

If a 16-byte ETE has been fetched, or if a 32-byte ETE has been fetched but bit 128 of the ETE (T) is zero, the basic PROGRAM CALL operation is specified. If a 32-byte ETE has been fetched and


LzLabs GmbH. Ex. 1008-533

bit 128 of the ETE is one, the stacking PROGRAM CALL operation is specified.

**Basic PROGRAM CALL**

The following operations are performed when basic PROGRAM CALL is specified.

Bits 32-62 of the current PSW (the addressing-mode bit and the updated instruction address) are placed in bit positions 0-30 of general register 14. Bit 15 of the PSW (the problem-state bit) is placed in bit position 31 of general register 14.

Bits 32-62 of the ETE (A and the EIA), with a zero appended on the right, are placed in PSW bit positions 32-63 (the addressing-mode bit and the instruction address). Bit 63 of the ETE (P) is placed in PSW bit position 15 (the problem-state bit).

The PSW-key mask, bits 0-15 of control register 3, is placed in bit positions 0-15 of general register 3, and the current PASN, bits 16-31 of control register 4, is placed in bit positions 16-31 of general register 3.

Bits 96-111 of the ETE (the EKM) are ORed with the PSW-key mask, bits 0-15 of control register 3, and the result replaces the PSW-key mask in control register 3.

Bits 64-95 of the ETE (the entry parameter) are loaded into general register 4.

**Stacking PROGRAM CALL**

The following operations are performed when stacking PROGRAM CALL is specified.

The stacking process is performed to form a linkage-stack program-call state entry and place the following information in the state entry: current PSW (with an unpredictable PER mask), PSW-key mask, PASN, SASN, EAX, called-space identification, program-call number, contents of general registers 0-15, and contents of access registers 0-15. This is described in “Stacking Process” on page 5-73. The entry-type code in the state entry is 0000101 binary.

Bits 32-62 of the ETE (A and the EIA), with a zero appended on the right, are placed in PSW bit positions 32-63 (the addressing-mode bit and the instruction address). Bit 63 of the ETE (P) is placed in PSW bit position 15 (the problem-state bit).

When bit 131 of the ETE (K) is zero, bits 8-11 of the PSW (the PSW key) remain unchanged. When bit 131 of the ETE is one, bits 136-139 of the ETE (the EK) replace the PSW key in the PSW.

When bit 132 of the ETE (M) is zero, bits 96-111 of the ETE (the EKM) are ORed with the PSW-key mask, bits 0-15 of control register 3, and the result replaces the PSW-key mask in control register 3. When bit 132 of the ETE is one, bits 96-111 of the ETE replace the PSW-key mask in control register 3.

When bit 133 of the ETE (E) is zero, the EAX, bits 0-15 of control register 8, remains unchanged. When bit 133 of the ETE is one, bits 144-159 of the ETE (the EEAX) replace the EAX in control register 8.

When bit 134 of the ETE (C) is zero, bits 16 and 17 of the PSW (the address-space-control bits) are set to 00 binary (primary-space mode). When bit 134 of the ETE is one, the address-space-control bits in the PSW are set to 01 binary (access-register mode).

Bits 64-95 of the ETE (the entry parameter) are loaded into general register 4.

Key-controlled protection does not apply to references to the linkage stack, but low-address and page protection do apply.

**PROGRAM CALL to Current Primary (PC-cp)**

If bits 16-31 of the ETE (the ASN) are zeros, PROGRAM CALL to current primary (PC-cp) is specified, and the execution of the instruction is completed after the operations described in “PROGRAM CALL PC-Number Translation” and either “Basic PROGRAM CALL” or “Stacking PROGRAM CALL” have been performed and the following operations have been performed.

The current PASN, bits 16-31 of control register 4, is placed in bit positions 16-31 of control register 3 to become the current SASN.

The current PSTD, bits 0-31 of control register 1, is placed in control register 7 to become the current SSTD.



LzLabs GmbH. Ex. 1008-534

The basic PC-cp operation is depicted in parts 1 and 2 of Figure 10-21 on page 10-59. The stacking PC-cp operation is depicted in parts 1 and 3 of the figure.

**PROGRAM CALL with Space Switching (PC-ss)**

If the ASN in the ETE is nonzero, PROGRAM CALL with space switching (PC-ss) is specified, and the execution of the instruction is completed after the operations described in "PROGRAM CALL PC-Number Translation" and either "Basic PROGRAM CALL" or "Stacking PROGRAM CALL" have been performed and the following operations have been performed.

When the ASF control is zero, the ASN in the ETE is translated by means of a two-level table lookup to locate an ASN-second-table entry (ASTE). Otherwise, when the ASF control is one, the ASTE may be located either by means of ASN translation or by means of obtaining its address directly from the ETE, and which of these occurs is unpredictable.

When ASN translation occurs, bits 16-25 of the ETE are used as a 10-bit AFX to index into the ASN first table, and bits 26-31 are used as a 6-bit ASX to index into the ASN second table specified by the AFX. The ASN table-lookup process is described in "ASN Translation" on page 3-18. The exceptions associated with ASN translation are collectively called ASN-translation exceptions. These exceptions and their priority are described in Chapter 6, "Interruptions."

When ASN translation does not occur, bits 161-185 of the ETE, with six zeros appended on the right, are used as the real address of the ASTE. An ASX-translation exception is recognized if bit 0 of the ASTE is one, and an ASN-translation-specification exception may be recognized if any of bits 30, 31, and 60-63 of the ASTE is one. (These exceptions are a subset of the ASN-translation exceptions.)

Bits 16-31 of the ETE (the ASN) are placed in bit positions 16-31 of control register 4 as the new PASN.

Bits 64-95 of the ASTE (the STD) are placed in control register 1 as the new PSTD.

Bits 32-47 of the ASTE (the AX) are placed in bit positions 0-15 of control register 4 as the new authorization index.

When the ASF control is zero, bits 96-127 of the ASTE (the LTD) are placed in control register 5 as the new linkage-table designation. When the ASF control is one, bits 1-25 of the ASTE address are placed in bit positions 1-25 of control register 5 as the new primary-ASTE origin, and zeros are placed in bit positions 0 and 26-31.

In basic PROGRAM CALL, or in stacking PROGRAM CALL when bit 135 of the ETE (S) is zero, the PASN existing before the PASN is replaced from the ETE is placed in bit positions 16-31 of control register 3 to become the current SASN, and the PSTD existing before the PSTD is replaced from the ASTE is placed in control register 7 to become the current SSTD. (The SASN and SSTD are set equal to the old PASN and PSTD, respectively.)

In stacking PROGRAM CALL when bit 135 of the ETE (S) is one, the SASN is replaced by the PASN after the PASN is replaced from the ETE, and the SSTD is replaced by the PSTD after the PSTD is replaced from the ASTE. (The SASN and SSTD are set equal to the new PASN and PSTD, respectively.)

The description in this paragraph applies if the subspace-group facility is installed and the ASF control is one. After the new PSTD has been placed in control register 1 and the new primary-ASTE origin has been placed in control register 5, if (1) the subspace-group-control bit, bit 22, in the PSTD is one, (2) the dispatchable unit is subspace active, and (3) the primary-ASTE origin designates the ASTE for the base space of the dispatchable unit, then bits 1-23 and 25-31 of the PSTD in control register 1 are replaced by bits 1-23 and 25-31 of the STD in the ASTE for the subspace in which the dispatchable unit last had control. This replacement occurs before a replacement of the SSTD in control register 7 by the PSTD. Further details are in "Subspace-Replacement Operations" on page 5-59.

The PC-ss operation is depicted in parts 1, 2, 3, and 4 of Figure 10-21 on page 10-59.


LzLabs GmbH. Ex. 1008-535

**PROGRAM CALL Serialization**

For both the PC-cp and PC-ss operations, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

**Special Conditions**

The basic PROGRAM CALL operation can be performed successfully only when the CPU is in the primary-space mode at the beginning of the operation and the subsystem-linkage control, bit 0 of the linkage-table designation, is one. Stacking PROGRAM CALL can be performed successfully only when the CPU is in the primary-space mode or access-register mode at the beginning of the operation and the subsystem-linkage control is one. In addition, PC-ss can be performed successfully only when the ASN-translation control, bit 12 of control register 14, is one. If any of these rules is violated, a special-operation exception is recognized.

A stack-full or stack-specification exception may be recognized during the stacking process.

When, for PC-ss, the primary space-switch-event-control bit, bit 0 of control register 1, is one either before or after the execution of the instruction, a space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PC-ss operation if a PER event is reported.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-20 on page 10-57.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch or store, except for key-controlled protection, linkage-stack entry)
- Addressing (linkage-table designation in primary ASN-second-table entry, only when address-space-function control is one; linkage-table entry; entry-table entry; ASN-first-table entry, PC-ss only, and only when ASN translation occurs; ASN-second-table entry, PC-ss only)
- AFX translation (PC-ss only, and only when ASN translation occurs)
- ASN-translation specification (PC-ss only)
- ASX translation (PC-ss only)
- EX translation
- LX translation
- PC-translation specification
- Privileged operation (AND of AKM and PSW-key mask is zero in the problem state)
- Space-switch event (PC-ss only)
- Special operation
- Stack full (stacking PC only)
- Stack specification (stacking PC only)
- Subspace replacement (PC-ss only, and only when subspace-group facility is installed and address-space-function control is one)
- Trace



LzLabs GmbH. Ex. 1008-536

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to DAT being off or the CPU being in secondary-space mode or home-space mode. |
| 7.C | Special-operation exception due to the CPU being in access-register mode (only when address-space-function control is zero, and may be recognized instead at 8.B.8). |
| 7.D | Special-operation exception due to subsystem-linkage control in linkage-table designation in control register 5 being zero (only when address-space-function control is zero). |
| 8.A | Trace exceptions. |
| 8.B.1 | Addressing exception for access to linkage-table designation in primary ASN-second-table entry (only when address-space-function control is one). |
| 8.B.2 | Special-operation exception due to subsystem-linkage control in linkage-table designation in primary ASN-second-table entry being zero (only when address-space-function control is one). |
| 8.B.3 | LX-translation exception due to linkage-table entry being outside table. |
| 8.B.4 | Addressing exception for access to linkage-table entry. |
| 8.B.5 | LX-translation exception due to I bit (bit 0) in linkage-table entry being one. |
| 8.B.6 | EX-translation exception due to entry-table entry being outside table. |
| 8.B.7 | Addressing exception for access to entry-table entry. |
| 8.B.8 | Special-operation exception due to the CPU being in access-register mode (basic PC only, and may be recognized at 7.C if address-space-function control is zero). |
| 8.B.9 | PC-translation-specification exception due to invalid combination (bit 32 is zero and bits 33-39 not zeros) in entry-table entry. |
| 8.B.10 | Privileged-operation exception due to zero result from ANDing PSW-key mask and AKM in the problem state. |
| 8.B.11 | Special-operation exception due to ASN-translation control, bit 12 of control register 14, being zero (PC-ss only). |
| 8.B.12 | Addressing exception for access to ASN-first-table entry (PC-ss only, and only when ASN translation occurs). |

*Figure 10-20 (Part 1 of 2). Priority of Execution: PROGRAM CALL*


LzLabs GmbH. Ex. 1008-537

8.B.13 AFX-translation exception due to I bit (bit 0) in ASN-first-table entry being one (PC-ss only, and only when ASN translation occurs).

8.B.14 ASN-translation-specification exception due to invalid ones (bits 28-31 or 26-31, depending on address-space-function control) in ASN-first-table entry (PC-ss only).

8.B.15 Addressing exception for access to ASN-second-table entry (PC-ss only).

8.B.16 ASX-translation exception due to I bit (bit 0) in ASN-second-table entry being one (PC-ss only).

8.B.17 ASN-translation-specification exception due to invalid ones (bits 30, 31, 60-63) in ASN-second-table entry (PC-ss only and optional).

**Note**: Subspace-replacement exceptions, which are not shown in detail in this figure, can occur with any priority after 8.B.17 and before 9.

8.B.18 Access exceptions (fetch) for entry descriptor of the current linkage-stack entry (stacking PC only).

**Note**: Exceptions 8.B.19-8.B.24 can occur only if there is not enough remaining free space in the current linkage-stack section.

8.B.19 Stack-specification exception due to remaining-free-space value in current linkage-stack entry not being a multiple of 8.

8.B.20 Access exceptions (fetch) for second word of the trailer entry of the current section. The entry is presumed to be a trailer entry; its entry-type field is not examined (stacking PC only).

8.B.21 Stack-full exception due to forward-section validity bit in the trailer entry being zero (stacking PC only).

8.B.22 Access exceptions (fetch) for entry descriptor of the header entry of the next section (stacking PC only). This entry is presumed to be a header entry; its entry-type field is not examined.

8.B.23 Stack-specification exception due to not enough remaining free space in the next section (stacking PC only).

8.B.24 Access exceptions (store) for second word of the header entry of the next section. If there is no exception, the header is now called the current entry.

8.B.25 Access exceptions (store) for entry descriptor of the current entry and for the new state entry (stacking PC only).

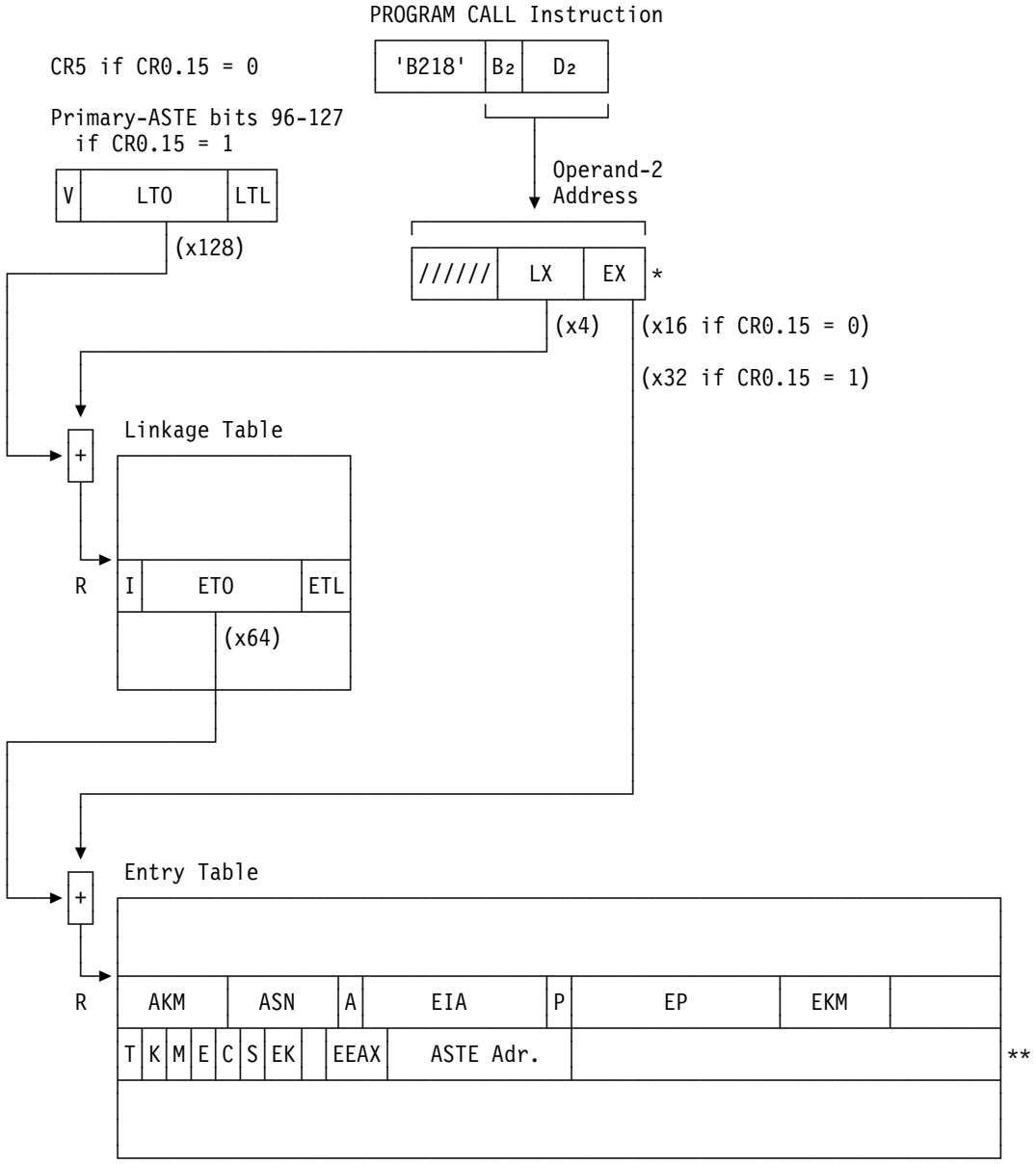
9. Space-switch event (PC-ss only).

*Figure 10-20 (Part 2 of 2). Priority of Execution: PROGRAM CALL*



LzLabs GmbH. Ex. 1008-538

**PC-Number Translation**




R: Address is real.
*: In stacking PC, PC number is placed in linkage stack.
**: Second 16 bytes of ETE exist only if CR0.15 = 1.

*Figure 10-21 (Part 1 of 4). Execution of PROGRAM CALL*


LzLabs GmbH. Ex. 1008-539

**Basic PC-cp and PC-ss**

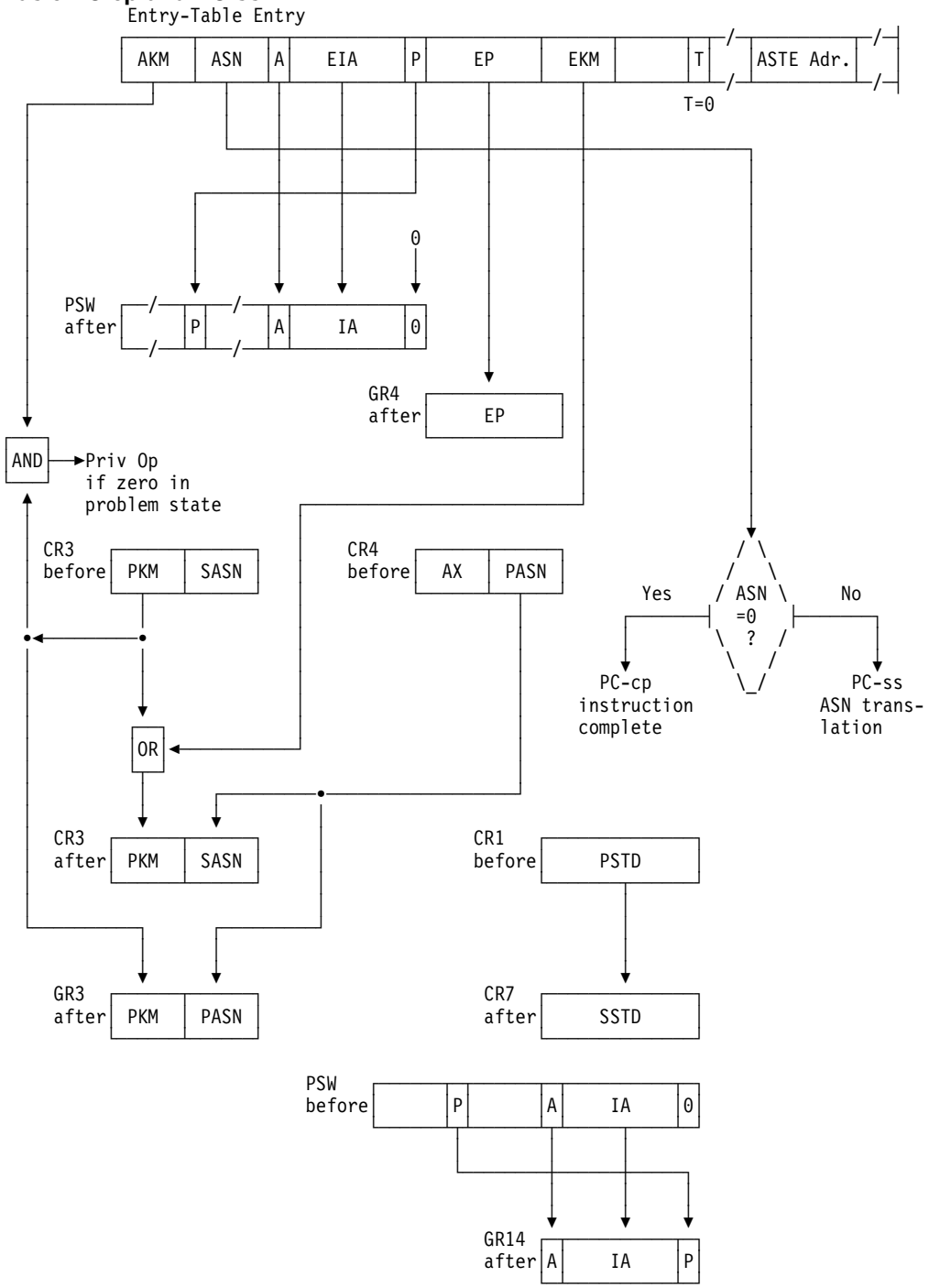
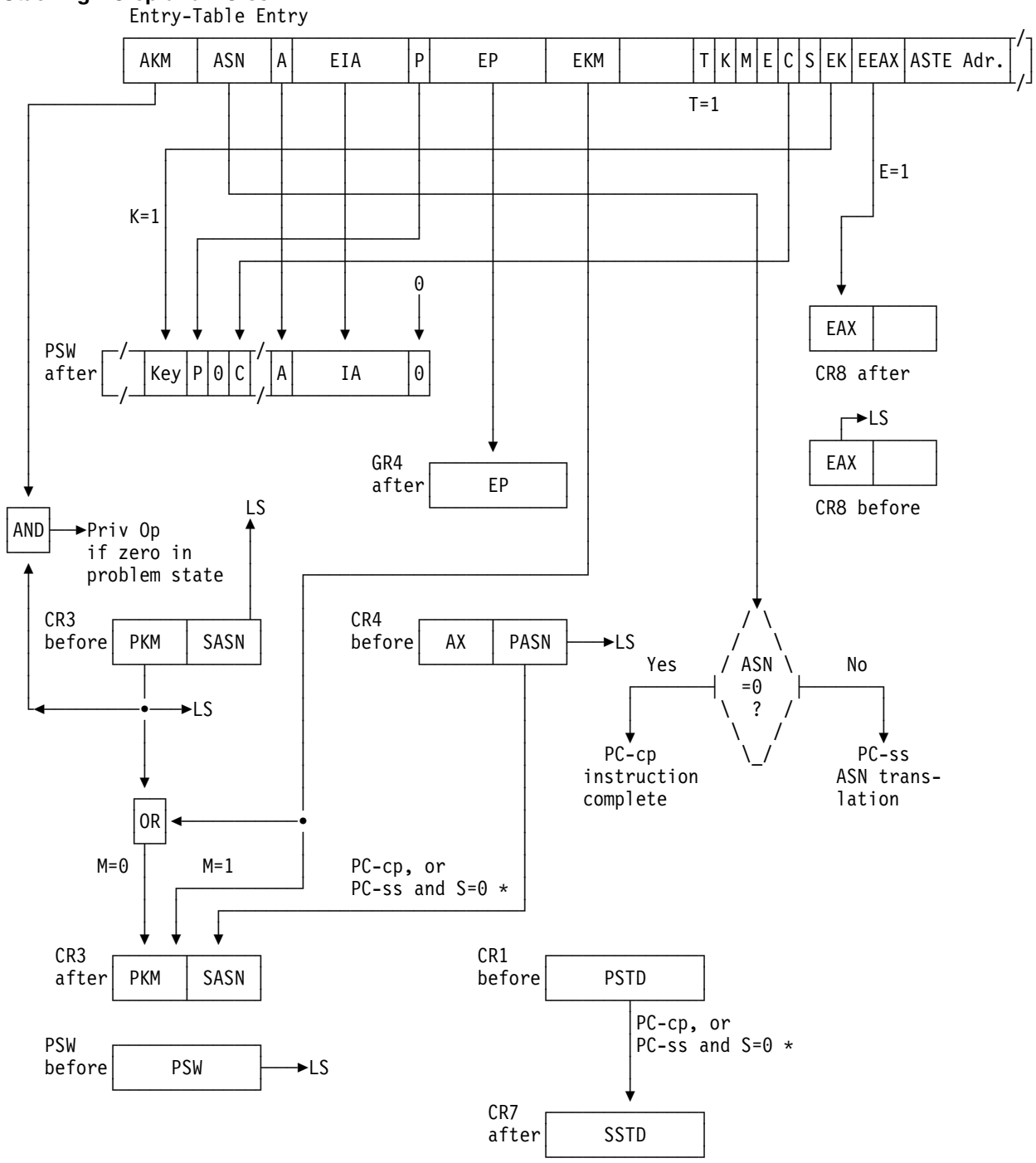



*Figure 10-21 (Part 2 of 4). Execution of PROGRAM CALL*



LzLabs GmbH. Ex. 1008-540

**Stacking PC-cp and PC-ss**



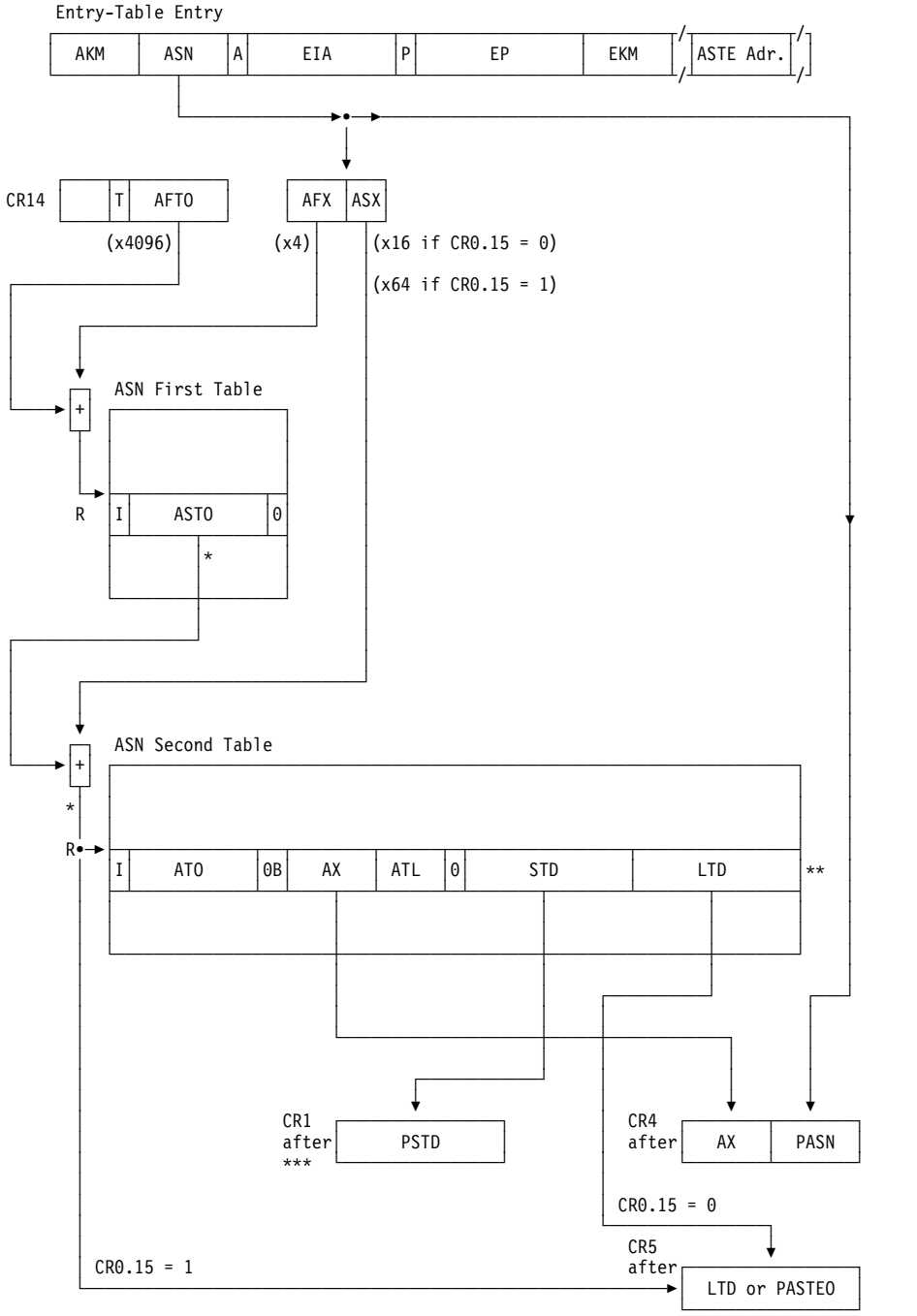

*: If PC-ss and S=1, SASN is replaced by new PASN, and SSTD is replaced by new PSTD.

*Figure 10-21 (Part 3 of 4). Execution of PROGRAM CALL*



LzLabs GmbH. Ex. 1008-541

**ASN Translation for PC-ss**




R: Address is real.
*: If CR0.15 = 1, ASTE address may be obtained by ASN translation or directly from ETE.
**: ASTE is 64 bytes if CR0.15 = 1; last 48 bytes are not shown.
***: If subspace-group facility installed and CR0.15 = 1, bits 1-23 and 25-31 of PSTD may be replaced from a subspace STD.

*Figure 10-21 (Part 4 of 4). Execution of PROGRAM CALL*

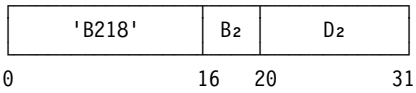
**Programming Note:** To ensure predictable operation of PC-ss when the address-space-function control is one, the ASN-second-table-entry address in the entry-table entry must be the same as the one that would result from ASN translation of the ASN in the entry-table entry.



LzLabs GmbH. Ex. 1008-542

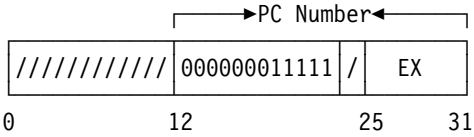
## PROGRAM CALL FAST

PCF D2(B2) [S]



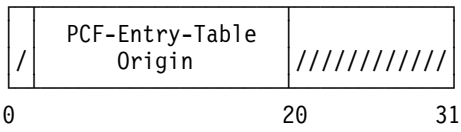

When the program-call-fast facility is installed, the program-call-fast control, bit 28 of control register 0, is one, and bits 12-23 of the second-operand address have the value 31 (01F hex), it is unpredictable whether this definition or the PROGRAM CALL definition applies. When any of those conditions is not true, the PROGRAM CALL definition applies. The program-call-fast facility is not installed on models having z/Architecture installed.

An entry index (EX, which is only bits 1-7 of the EX used by PROGRAM CALL) specified by the second-operand address is used in conjunction with the PCF-entry-table origin to select an entry in the PCF entry table. The second-operand address has the following format:




The PCF-entry-table origin is specified at real locations 196-199 (C4-C7 hex) and has the following format:

Real locations 196-199

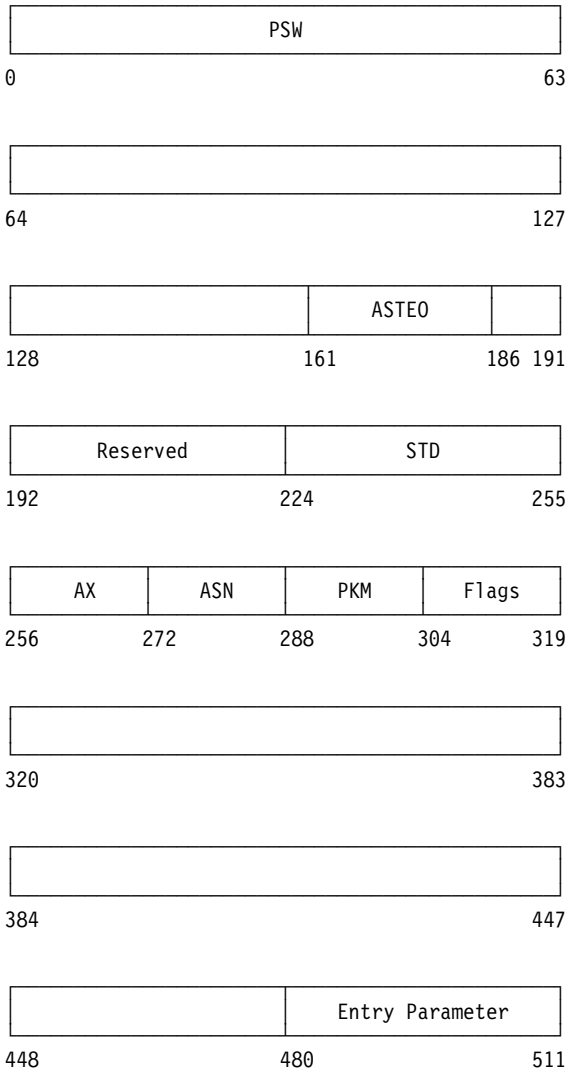



The PCF entry table resides in real storage, is 8K-bytes long on a 4K-byte boundary, and contains 128 64-byte entries.

The 31-bit real address of the PCF-entry-table entry is obtained by appending 12 zeros on the right to the PCF-entry-table origin and adding the EX, with 6 rightmost and 18 leftmost zeros appended. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31} - 1$ to zero. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

Key-controlled protection does not apply to accesses to the PCF-entry-table origin or the PCF entry table.

The PCF-entry-table entry has the following format:




Bits 64-160, 186-223, and 320-479 in the PCF-entry-table entry are reserved for future extensions and should be zeros; otherwise, the program may not operate compatibly in the future.

The PCF-entry-table entry causes a space-switching operation to occur if it contains a nonzero ASN. When the PCF-entry-table entry contains a zero ASN, the operation is called


LzLabs GmbH. Ex. 1008-543

PROGRAM CALL FAST to current primary (PCF-cp); when the PCF-entry-table entry contains a nonzero ASN, the operation is called PROGRAM CALL FAST with space switching (PCF-ss).

**PROGRAM CALL FAST to Current Primary (PCF-cp)**

If the ASN in the PCF-entry-table entry is zero, PROGRAM CALL FAST to current primary is specified, and the execution of the instruction is completed after the following operations have been performed.

A stacking process is performed to form a linkage-stack program-call state entry and place the following information in the state entry: current PSW (with an unpredictable PER mask), PSW-key mask, PASN, SASN, EAX, called-space identification, program-call number, contents of general registers 0-15, and contents of access registers 0-15. This is described in "Stacking Process" on page 5-73. The entry-type code in the state entry is 0000101 binary. The called-space identification in a program-call state entry formed by either PCF-cp or PCF-ss is always all zeros.

Bits 8-63 of the PCF-entry-table entry are placed in PSW bit positions 8-63.

Bits 480-511 of the PCF-entry-table entry (the entry parameter) are loaded into general register 4.

Key-controlled protection does not apply to references to the linkage stack, but low-address and page protection do apply.

**PROGRAM CALL FAST with Space Switching (PCF-ss) Operations**

If the ASN in the PCF-entry-table entry is nonzero, PROGRAM CALL FAST with space switching (PCF-ss) is specified, and the execution of the instruction is completed after the operations specified in "PROGRAM CALL FAST to current primary (PCF-cp)" have been performed and the following operations have been performed.

Bits 272-287 of the PCF-entry-table entry (the ASN) are placed in bit positions 16-31 of control register 4 as the new PASN.

Bits 224-255 of the PCF-entry-table entry (the STD) are placed in control register 1 as the new PSTD. The subspace-group control, bit 22, in the new PSTD is ignored.

Bits 256-271 of the PCF-entry-table entry (the AX) are placed in bit positions 0-15 of control register 4 as the new authorization index.

Bits 161-185 of the PCF-entry-table entry (the ASTEO) are placed in bit positions 1-25 of control register 5 as the new primary-ASTE origin, and zeros are placed in bit positions 0 and 26-31.

The PASN existing before the PASN is replaced from the PCF-entry-table entry is placed in bit positions 16-31 of control register 3 to become the current SASN, and the PSTD existing before the PSTD is replaced from the PCF-entry-table entry is placed in control register 7 to become the current SSTD. (The SASN and SSTD are set equal to the old PASN and PSTD, respectively.)

Bits 288-303 of the PCF-entry-table entry (the PKM) replace the PSW-key mask in bit positions 0-15 of control register 3.

**PROGRAM CALL FAST Serialization**

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

**Special Conditions**

The operation can be performed successfully only when the CPU is in the primary-space mode or access-register mode at the beginning of the operation and the address-space-function control, bit 15 of control register 0, is one. If either of these rules is violated, a special-operation exception is recognized.

Bits 304-319 (flags) of the PCF-entry-table entry are available to control the operation. If any of bits 304-319 is a one, an EX-translation exception is recognized in both the problem and supervisor states. The program-call number is stored in bit positions 12-31 of the word at real location 144, bits 0-10 of the word are set to zeros, and bit 11 of the word is set to one to indicate that the exception was recognized by PROGRAM CALL FAST.



LzLabs GmbH. Ex. 1008-544

A stack-full or stack-specification exception may be recognized during the stacking process.

When, for PCF-ss, the primary space-switch-event-control bit, bit 0 of control register 1, is one either before or after the execution of the instruction, a space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PCF-ss operation if a PER event is indicated.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-22 on page 10-66.

The PSW fields which are to be loaded are not checked for validity before they are loaded. However, after loading, a specification exception is recognized, and a program interruption occurs, when any of the following is true for the newly loaded fields: a zero is in bit position 12, bits 24-31 are not all zeros, or a zero is in bit position 32 and bits 33-39 are not all zeros. In these cases, the operation is completed, and the resulting instruction-length code is 0. The specification exception, which in this case is listed as a program exception in this instruction, is described in "Early Exception Recognition" on page 6-9. It may be considered as occurring early in the process of preparing to execute the following instruction.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch or store, except for key-controlled protection, linkage-stack entry)
- Addressing (PCF-entry-table entry)
- EX translation
- Space-switch event (PCF-ss only)
- Special operation
- Specification
- Stack full
- Stack specification

**Programming Notes:**

1. PROGRAM CALL FAST is contrasted to PROGRAM CALL as follows. PROGRAM CALL FAST does not perform the following operations:
   - ASN tracing
   - ASN translation
   - ASN-translation-control checking
   - Authorization-key-mask checking
   - Extended-authorization-index change
   - PC-number translation
   - PSW-validity checking
   - SASN and SSTD change (not performed for PCF-cp only)
   - Subspace replacement
   - Subsystem-linkage-control checking
   - Translation-mode checking (checking that the final translation mode is the primary-space or access-register mode)
2. Because it is unpredictable whether the PROGRAM CALL FAST definition or the PROGRAM CALL definition applies when the conditions for execution of PROGRAM CALL FAST are met, linkage-table entry 31 must designate an entry table containing entries that will produce the same results as the program-call-fast entry-table entries, in order for a predictable operation to occur. The operation is not entirely predictable since PC-cp sets the SASN and SSTD equal to the PASN and PSTD while PCF-cp leaves the SASN and SSTD unchanged.


LzLabs GmbH. Ex. 1008-545

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B Special-operation exception due to DAT being off, the CPU being in secondary-space mode or home-space mode, or the address-space-function control being zero.

7.C.1 Addressing exception for access to the PCF-entry-table entry.

7.C.2 EX-translation exception due to nonzero flags in the PCF-entry-table entry.

8. Access exceptions (fetch) for entry descriptor of the current linkage-stack entry.

**Note**: Exceptions 9.-14. can occur only if there is not enough remaining free space in the current linkage-stack section.

9. Stack-specification exception due to remaining-free-space value in current linkage-stack entry not being a multiple of 8.

10. Access exceptions (fetch) for second word of the trailer entry of the current section. The entry is presumed to be a trailer entry; its entry-type field is not examined.

11. Stack-full exception due to forward-section validity bit in the trailer entry being zero.

12. Access exceptions (fetch) for entry descriptor of the header entry of the next section. This entry is presumed to be a header entry; its entry-type field is not examined.

13. Stack-specification exception due to not enough remaining free space in the next section.

14. Access exceptions (store) for second word of the header entry of the next section. If there is no exception, the header is now called the current entry.

15. Access exceptions (store) for entry descriptor of the current entry and for the new state entry.

16. Space-switch event (PCF-ss only).

17. Specification exception due to any PSW error of the type that causes an immediate interruption.

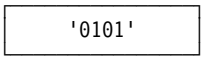
*Figure 10-22. Priority of Execution: PROGRAM CALL FAST*



LzLabs GmbH. Ex. 1008-546

# PROGRAM RETURN

```
PR      [E]
```




The PSW, except for the PER-mask bit and the condition code, saved in the last linkage-stack state entry is restored as the current PSW. The PER mask in the current PSW remains unchanged. The resulting value of the condition code in the current PSW is unpredictable. The contents of general registers 2-14 and access registers 2-14 also are restored from the state entry. When the entry-type code in the entry descriptor of the state entry is 0000101 binary, indicating a program-call state entry, the primary ASN (PASN), secondary ASN (SASN), PSW-key mask (PKM), and extended authorization index (EAX) in the control registers also are restored from the state entry. When the entry-type code is 0000100 binary, indicating a branch state entry, the current PASN, SASN, PKM, and EAX remain unchanged.

The last state entry is located, and information in it is restored, as described in "Unstacking Process" on page 5-75. The state entry is logically deleted from the linkage stack, and the linkage-stack-entry address in control register 15 is replaced by the address of the next preceding state or header entry. This also is described in "Unstacking Process."

When the state entry is a program-call state entry, it causes a space-switching operation to occur if it contains a PASN that is not equal to the current PASN. When the state entry contains a PASN that is equal to the current PASN, the operation is called PROGRAM RETURN to current primary (PR-cp); when the state entry contains a PASN that is not equal to the current PASN, the operation is called PROGRAM RETURN with space switching (PR-ss). PASN translation occurs in PR-ss. SASN translation and authorization may occur in either PR-cp or PR-ss. The terms PR-cp and PR-ss do not apply when the state entry is a branch state entry.

Key-controlled protection does not apply to accesses to the linkage stack, but low-address and page protection do apply.

The sections "PASN Translation," "SASN Translation," "SASN Authorization," and "PROGRAM RETURN Serialization" apply only when the unstacked state entry is a program-call state entry. The functions described in those sections are not performed when the state entry is a branch state entry.

**PASN Translation**

If the new PASN is equal to the old PASN in bit positions 16-31 of control register 4, PASN translation is not performed, and the authorization index (AX), PASN, PSTD, and primary-ASN-second-table-entry (primary-ASTE) origin in the control registers are not changed.

If the new PASN is not equal to the old PASN, the new PASN replaces the PASN in bit positions 16-31 of control register 4 and is translated to locate a 64-byte ASTE. The ASN table-lookup process is described in "ASN Translation" on page 3-18. The exceptions associated with ASN translation are collectively called ASN-translation exceptions. These exceptions and their priority are described in Chapter 6, "Interruptions."

Bits 64-95 of the ASTE are placed in control register 1 as the new PSTD. Bits 32-47 of the ASTE are placed in bit positions 0-15 of control register 4 as the new AX. Bits 1-25 of the ASTE address are placed in bit positions 1-25 of control register 5 as the new primary-ASTE origin, and zeros are placed in bit positions 0 and 26-31.

The description in this paragraph applies if the subspace-group facility is installed and PASN translation has occurred. If (1) the subspace-group-control bit, bit 22, in the new PSTD is one, (2) the dispatchable unit is subspace active, and (3) the new primary-ASTE origin designates the ASTE for the base space of the dispatchable unit, then bits 1-23 and 25-31 of the new PSTD in control register 1 are replaced by bits 1-23 and 25-31 of the STD in the ASTE for the subspace in which the dispatchable unit last had control. This replacement occurs, in the case when the new SASN is equal to the new PASN, before the SSTD is set equal to the PSTD. Further details are in "Subspace-Replacement Operations" on page 5-59.


LzLabs GmbH. Ex. 1008-547

**SASN Translation** The new SASN replaces the SASN in bit positions 16-31 of control register 3.

If the new SASN is equal to the new PASN, the SSTD in control register 7 is set equal to the new PSTD in control register 1. If the new SASN is not equal to the new PASN, the new SASN is translated to locate a 64-byte ASTE. Bits 64-95 of the ASTE are placed in bit positions 0-31 of control register 7 as the new SSTD.

**SASN Authorization**

If the new SASN is not equal to the new PASN, the authority-table origin (ATO) from the ASTE for the new SASN is used as the base for a third table lookup. The new authorization index, bits 0-15 of control register 4, is used, after it has been checked against the authority-table length, as the index to locate the entry in the authority table. The authority-table lookup is described in "ASN Authorization" on page 3-23.

The description in this paragraph applies if the subspace-group facility is installed and SASN translation and authorization have occurred. If (1) the subspace-group-control bit, bit 22, in the new SSTD is one, (2) the dispatchable unit is subspace active, and (3) the ASTE origin obtained by SASN translation designates the ASTE for the base space of the dispatchable unit, then bits 1-23 and 25-31 of the new SSTD in control register 7 are replaced by bits 1-23 and 25-31 of the STD in the ASTE for the subspace in which the dispatchable unit last had control. Further details are in "Subspace-Replacement Operations" on page 5-59.

**PROGRAM RETURN Serialization**

When the unstacked state entry is a program-call state entry, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

**Special Conditions**

The instruction can be executed successfully only when the CPU is in the primary-space mode or access-register mode at the beginning of the operation and the address-space-function control, bit 15 of control register 0, is one. In addition, the ASN-translation process can be performed, for either the PASN or the SASN, only when the ASN-translation control, bit 12 of control register 14, is one. If any of these rules is violated, a special-operation exception is recognized.

A stack-empty, stack-operation, stack-specification, or stack-type exception may be recognized during the unstacking process.

When, for PR-ss, the primary space-switch-event control, bit 0 of control register 1, is one either before or after the execution of the instruction, a space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PR-ss operation if a PER event is reported.

The PSW which is to be loaded by the instruction is not checked for validity before it is loaded. However, after loading, a specification exception is recognized, and a program interruption occurs, when the newly loaded PSW contains a zero in bit position 12, when the contents of bit positions 0, 2-4, and 24-31 are not all zeros, or when bit position 32 contains a zero and the contents of bit positions 33-39 are not all zeros. In these cases, the operation is completed, and the resulting instruction-length code is 0. The specification exception, which in this case is listed as a program exception in this instruction, is described in "Early Exception Recognition" on page 6-9. It may be considered as occurring early in the process of preparing to execute the following instruction.

If a space-switch event is indicated and the PSW that was loaded by the instruction is invalid because of a reason described in the preceding paragraph, it is unpredictable whether the resulting instruction-length code is 0 or 1, or 0 or 2 if EXECUTE was used.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-23 on page 10-69.

***Resulting Condition Code:*** The code is unpredictable.



LzLabs GmbH. Ex. 1008-548

***Program Exceptions:***

- Access (fetch and store, except key-controlled protection, linkage-stack entry)
- Addressing (authority-table entry, if SASN translation occurs)
- ASN translation (if PASN or SASN translation occurs)
- Secondary authority (if SASN translation occurs)
- Space-switch event
- Special operation
- Specification
- Stack empty
- Stack operation
- Stack specification
- Stack type
- Subspace replacement (if subspace-group facility is installed and PASN or SASN translation occurs)
- Trace

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7. | Special-operation exception due to DAT being off, the CPU being in secondary-space mode or home-space mode, or the address-space-function control, bit 15 of control register 0, being zero. |
| 8.A | Trace exceptions. |
| 8.B.1 | Access exceptions (fetch) for entry descriptor of the current linkage-stack entry. |
| 8.B.2 | Stack-type exception due to current entry not being a state entry or header entry.<br><br>**Note:** Exceptions 8.B.3-8.B.7 can occur only if the current entry is a header entry. |
| 8.B.3 | Stack-operation exception due to unstack-suppression bit in the header entry being one. |
| 8.B.4 | Access exceptions (fetch) for second word of the header entry. |
| 8.B.5 | Stack-empty exception due to backward stack-entry validity bit in the header entry being zero. |
| 8.B.6 | Access exceptions (fetch) for entry descriptor of preceding entry, which is the entry designated by the backward stack-entry address in the header entry. |
| 8.B.7 | Stack-specification exception due to preceding entry being a header entry. |

*Figure 10-23 (Part 1 of 2). Priority of Execution: PROGRAM RETURN*


LzLabs GmbH. Ex. 1008-549

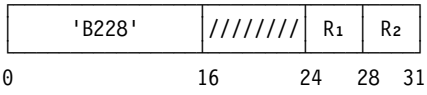
| | |
|---|---|
| 8.B.8 | Stack-type exception due to preceding entry not being a state entry. |
| 8.B.9 | Stack-operation exception due to unstack-suppression bit being one in the state entry. |
| 8.B.10 | Access exceptions (fetch) for the state entry, and access exceptions (store) for entry descriptor of entry preceding the state entry.<br><br>**Note:** Exceptions 8.B.11-8.B.15 and the event 9 can occur only if the state entry is a program-call state entry. |
| 8.B.11 | Special-operation exception due to the ASN-translation control, bit 12 of control register 14, being zero (if PASN or SASN translation occurs). |
| 8.B.12 | ASN-translation exceptions (if PASN or SASN translation occurs).<br><br>**Note:** Subspace-replacement exceptions for replacement of bits in either the PSTD or the SSTD, which are not shown in detail in this figure, can occur with any priority after 8.B.12 and before 9. |
| 8.B.13 | Secondary-authority exception due to authority-table entry being outside table (if SASN translation occurs). |
| 8.B.14 | Addressing exception for access to authority-table entry (if SASN translation occurs). |
| 8.B.15 | Secondary-authority exception due to S bit in authority-table entry being zero (if SASN translation occurs). |
| 9. | Space-switch event (PR-ss only). |
| 10. | Specification exception due to any PSW error of the type that causes an immediate interruption. |

*Figure 10-23 (Part 2 of 2). Priority of Execution: PROGRAM RETURN*

**Programming Note:** Because PROGRAM CALL cannot be executed successfully in the secondary-space or home-space mode, PROGRAM RETURN is not intended to load a PSW specifying one of these translation modes. PROGRAM RETURN, unlike SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST, does not recognize a space-switch event because of loading a PSW that specifies the home-space mode.

## PROGRAM TRANSFER

PT R1,R2 [RRE]



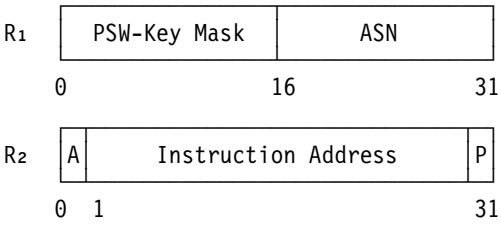

The contents of general register $R_1$ are used as the new values for the PSW-key mask, the PASN, and the SASN. The contents of general register $R_2$ are used as the new values for the problem-state bit, addressing-mode bit, and instruction address in the current PSW.

Bits 16-23 of the instruction are ignored.



LzLabs GmbH. Ex. 1008-550

General registers $R_1$ and $R_2$ have the following format:




When the contents of bit positions 16-31 of general register $R_1$ are equal to the current PASN, the operation is called PROGRAM TRANSFER to current primary (PT-cp); when the fields are not equal, the operation is called PROGRAM TRANSFER with space switching (PT-ss).

The contents of general register $R_2$ are used to update the problem-state bit, the addressing-mode bit, and the instruction address in the current PSW. Bit 31 of general register $R_2$ is placed in the problem-state bit position, PSW bit position 15, unless the operation would cause PSW bit 15 to change from one to zero (problem state to supervisor state). If such a change would occur, a privileged-operation exception is recognized. Bits 0-30 of general register $R_2$ replace the addressing-mode bit and the instruction address, bits 32-62, in the current PSW. Bit 63 of the PSW is set to zero.

Bits 0-15 of general register $R_1$ are ANDed with the PSW-key mask, bits 0-15 of control register 3, and the result replaces the PSW-key mask.

In both the PT-ss and PT-cp instructions, the ASN specified by bits 16-31 of general register $R_1$ replaces the SASN in control register 3, and the SSTD in control register 7 is replaced by the final contents of control register 1.

**PROGRAM TRANSFER to Current Primary (PT-cp)**

The PROGRAM TRANSFER to-current-primary (PT-cp) operation is depicted in part 1 of Figure 10-25 on page 10-74. The PT-cp operation is completed when the common portion of the PROGRAM TRANSFER operation, described above, is completed. The authorization index, PASN, primary STD, and contents of control register 5 (linkage-table designation or primary-ASN-second-table-entry origin) are not changed by PT-cp.

**PROGRAM TRANSFER with Space Switching (PT-ss)**

If the ASN in bit positions 16-31 of general register $R_1$ is not equal to the current PASN, a PROGRAM TRANSFER with space switching (PT-ss) operation is specified, and the ASN is translated by means of a two-level table lookup.

The PT-ss operation is depicted in parts 1 and 2 of Figure 10-25 on page 10-74. The PT-ss operation is completed as follows.

In PT-ss, the contents of bit positions 16-31 of general register $R_1$ are used as an ASN, which is translated by means of a two-level table lookup.

Bits 16-25 of general register $R_1$ are a 10-bit AFX that is used to select an entry from the ASN first table. Bits 26-31 are a six-bit ASX that is used to select an entry from the ASN second table. The ASN table-lookup process is described in “ASN Translation” on page 3-18. The exceptions associated with ASN translation are collectively called “ASN-translation exceptions.” These exceptions and their priority are described in Chapter 6, “Interruptions.”

The authority-table origin from the ASN-second-table entry (ASTE) is used as the base for a third table lookup. The current authorization index, bits 0-15 of control register 4, is used, after it has been checked against the authority-table length, as the index to locate the entry in the authority table. The authority-table lookup is described in “ASN Authorization” on page 3-23.

The PT-ss operation is completed by placing bits 64-95 of the ASTE in both the PSTD and SSTD positions, bit positions 0-31 of control registers 1 and 7, respectively. The contents of bit positions 32-47 of the ASTE replace the authorization index in bit positions 0-15 of control register 4. When the address-space-function (ASF) control, bit 15 of control register 0, is zero, the contents of bit positions 96-127 of the ASTE replace the LTD in bit positions 0-31 of control register 5. When the ASF control is one, bits 1-25 of the ASTE address are placed in bit positions 1-25 of control register 5 as the new primary-ASTE origin, and zeros are placed in bit positions 0 and 26-31. The ASN, bits 16-31 of general register $R_1$, replaces the SASN


LzLabs GmbH. Ex. 1008-551

and PASN in bit positions 16-31 of control registers 3 and 4.

The description in this paragraph applies if the subspace-group facility is installed and the ASF control is one. After the new PSTD has been placed in control register 1 and the new primary-ASTE origin has been placed in control register 5, if (1) the subspace-group-control bit, bit 22, in the PSTD is one, (2) the dispatchable unit is subspace active, and (3) the primary-ASTE origin designates the ASTE for the base space of the dispatchable unit, then bits 1-23 and 25-31 of the PSTD in control register 1 are replaced by bits 1-23 and 25-31 of the STD in the ASTE for the subspace in which the dispatchable unit last had control. This replacement occurs before a replacement of the SSTD in control register 7 by the PSTD. Further details are in "Subspace-Replacement Operations" on page 5-59.

**PROGRAM TRANSFER Serialization**

For both the PT-cp and PT-ss operations, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

**Special Conditions**

The instruction can be executed only when the CPU is in the primary-space mode and the subsystem-linkage control, bit 0 of the linkage-table designation, is one. If the CPU is in the real mode, secondary-space mode, access-register mode, or home-space mode, or if the subsystem-linkage control is zero, a special-operation exception is recognized.

Bit 31 of general register $R_2$ is placed in the problem-state bit position, PSW bit position 15, unless the operation would cause PSW bit 15 to change from one to zero (problem state to supervisor state). If such a change would occur, a privileged-operation exception is recognized.

The instruction is completed only if bits 0-7 of general register $R_2$ specify a valid combination of PSW bits 32-39. If bit 0 of general register $R_2$ is zero and bits 1-7 are not all zeros, a specification exception is recognized.

In addition to the above requirements, when a PT-ss instruction is specified, the ASN-translation control, bit 12 of control register 14, must be one; otherwise, a special-operation exception is recognized.

When, for PT-ss, the primary space-switch-event-control bit, bit 0 of control register 1, is one either before or after the execution of the instruction, a space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PT-ss operation if a PER event is reported.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-24 on page 10-73.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (linkage-table designation in primary ASN-second-table entry, only when address-space-function control is one; authority-table entry, PT-ss only)
- ASN translation (PT-ss only)
- Primary authority (PT-ss only)
- Privileged operation (attempt to set the supervisor state when in the problem state)
- Space-switch event (PT-ss only)
- Special operation
- Specification
- Subspace replacement (PT-ss only, and only when subspace-group facility is installed and address-space-function control is one)
- Trace



LzLabs GmbH. Ex. 1008-552

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B Special-operation exception due to DAT being off or the CPU being in secondary-space mode, access-register mode, or home-space mode.

7.C Special-operation exception due to subsystem-linkage control in linkage-table designation in control register 5 being zero (only when address-space-function control is zero).

8.A Trace exceptions.

8.B.1 Addressing exception for access to linkage-table designation in primary ASN-second-table entry (only when address-space-function control is one).

8.B.2 Special-operation exception due to subsystem-linkage control in linkage-table designation in primary ASN-second-table entry being zero (only when address-space-function control is one).

8.B.3 Privileged-operation exception due to attempt to set the supervisor state when in the problem state.

8.B.4 Specification exception due to invalid value in bit positions 0-7 of general register $R_2$.

8.B.5 Special-operation exception due to ASN-translation control, bit 12 of control register 14, being zero (PT-ss only).

8.B.6 ASN-translation exceptions (PT-ss only).

**Note:** Subspace-replacement exceptions, which are not shown in detail in this figure, can occur with any priority after 8.B.6 and before 9.

8.B.7 Primary-authority exception due to authority-table entry being outside table (PT-ss only).

8.B.8 Addressing exception for access to authority-table entry (PT-ss only).

8.B.9 Primary-authority exception due to P bit in authority-table entry being zero (PT-ss only).

9. Space-switch event (PT-ss only).

*Figure 10-24. Priority of Execution: PROGRAM TRANSFER*

**Programming Notes:**

1. The operation of PROGRAM TRANSFER (PT) is such that it may be used to restore the CPU to the state saved by a previous basic PROGRAM CALL operation. This restoration is accomplished by issuing PT 3,14. Though general registers 3 and 14 are not restored to their original values, the PASN, PSW-key mask, problem-state bit, addressing mode, and instruction address are restored, and the authorization index, PSTD, and LTD or primary-ASN-second-table-entry origin are made consistent with the restored PASN. The SASN is not saved by PROGRAM CALL or restored by PROGRAM TRANSFER; PROGRAM TRANSFER sets the SASN equal to the restored PASN.


LzLabs GmbH. Ex. 1008-553

2. With proper authority, and while being executed in a common area, PROGRAM TRANSFER may be used to change the primary address space to any desired space. The secondary address space is also changed to be the same as the new primary address space.
3. Unlike the RR-format branch instructions, a value of zero in the $R_2$ field for PROGRAM TRANSFER designates general register 0, and branching occurs.

**PT-cp and PT-ss**

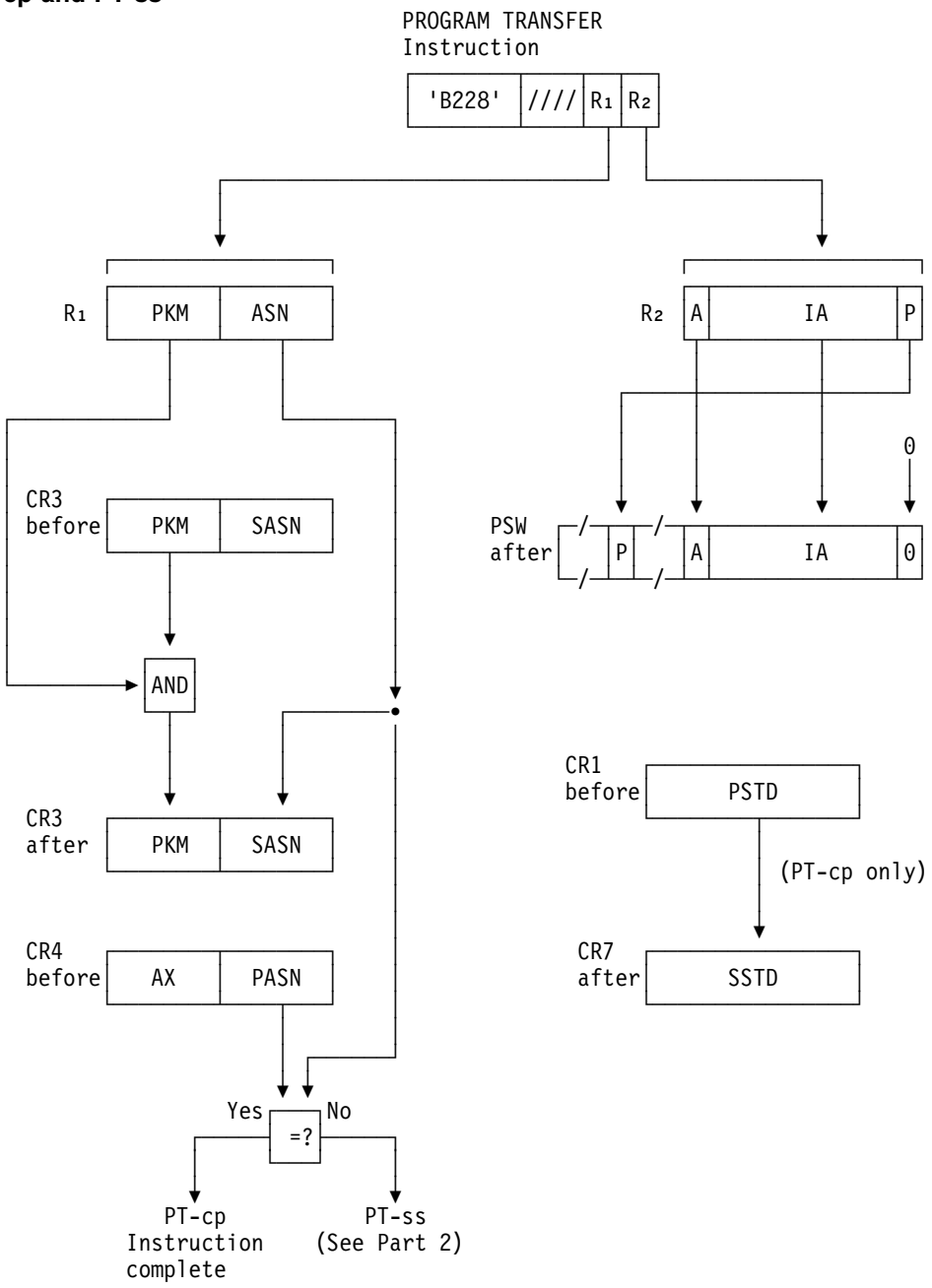
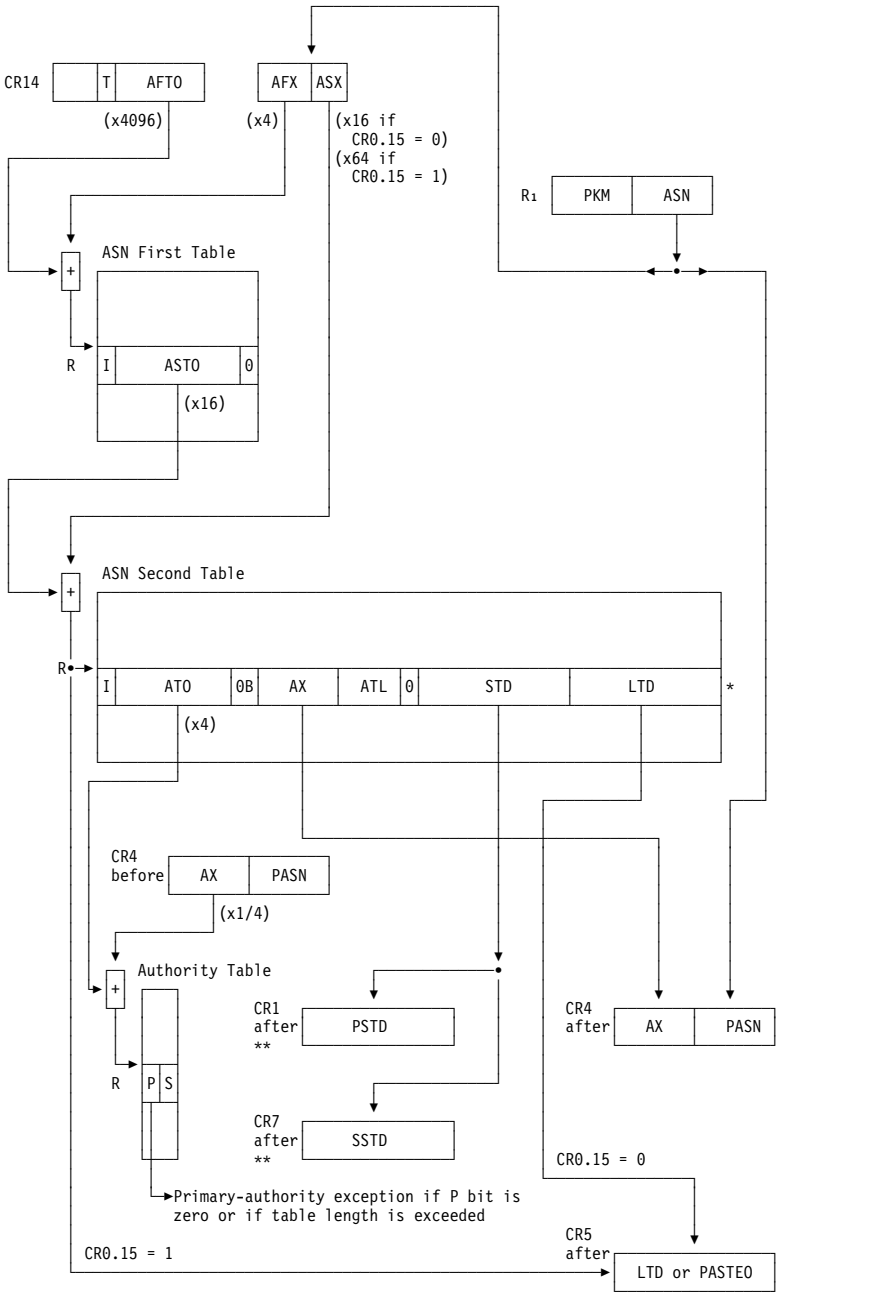



*Figure 10-25 (Part 1 of 2). Execution of PROGRAM TRANSFER*



LzLabs GmbH. Ex. 1008-554

**PT-ss**



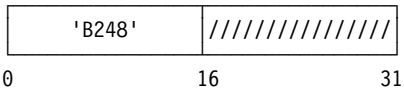

R: Address is real.
*: ASTE is 64 bytes if CR0.15 = 1; last 48 bytes are not shown.
**: If subspace-group facility installed and CR0.15 = 0, bits 1-23 and 25-31 of PSTD and SSTD may be replaced from a subspace STD.

*Figure 10-25 (Part 2 of 2). Execution of PROGRAM TRANSFER*


LzLabs GmbH. Ex. 1008-555

## PURGE ALB

```
PALB   [RRE]
```




The ART-lookaside buffer (ALB) of this CPU is cleared of entries. No change is made to the contents of addressable storage or registers.

Bits 16-31 of the instruction are ignored.

The ALB appears cleared of its original contents beginning with the execution of the next sequential instruction. The operation is not signaled to any other CPU.

A serialization function is performed.

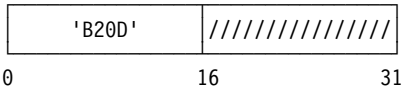
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Privileged operation

## PURGE TLB

```
PTLB                   [S]
```




The translation-lookaside buffer (TLB) of this CPU is cleared of entries. No change is made to the contents of addressable storage or registers.

Bits 16-31 of the instruction are ignored.

The TLB appears cleared of its original contents beginning with the fetching of the next sequential instruction. The operation is not signaled to any other CPU.

A serialization function is performed.

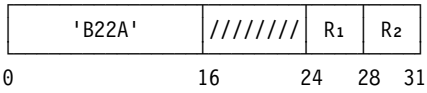
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Privileged operation

## RESET REFERENCE BIT EXTENDED

```
RRBE   R1,R2           [RRE]
```




The reference bit in the storage key for the 4K-byte block that is addressed by the contents of general register $R_2$ is set to zero. The contents of general register $R_1$ are ignored.

Bits 16-23 of the instruction are ignored.

In the 24-bit addressing mode, bits 8-19 of general register $R_2$ designate a 4K-byte block in real storage, and bits 0-7 and 20-31 of the register are ignored. In the 31-bit addressing mode, bits 1-19 of general register $R_2$ designate a 4K-byte block in real storage, and bits 0 and 20-31 of the register are ignored.

Because it is a real address, the address designating the storage block is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The remaining bits of the storage key, including the change bit, are not affected.

The condition code is set to reflect the state of the reference and change bits before the reference bit is set to zero.

***Resulting Condition Code:***

0 Reference bit zero; change bit zero
1 Reference bit zero; change bit one
2 Reference bit one; change bit zero
3 Reference bit one; change bit one

***Program Exceptions:***

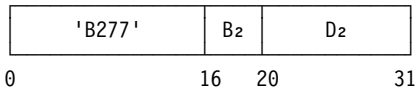
- Addressing (address specified by general register $R_2$)
- Privileged operation



LzLabs GmbH. Ex. 1008-556

## RESUME PROGRAM

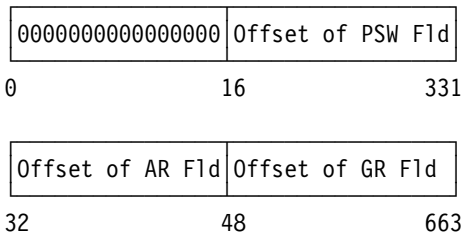
RP D2(B2) [S]




Certain fields in the current PSW and the contents of access register and general register $B_2$ are replaced from fields in the second operand. The offsets of the fields in the second operand are specified in a parameter list that immediately follows the instruction in the instruction address space.

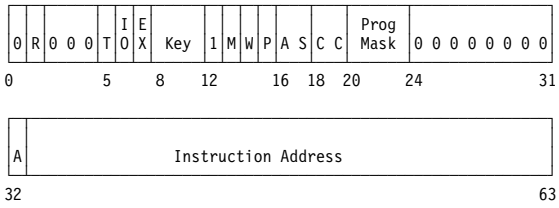
The instruction address space is the address space from which instructions are fetched. It is composed of real addresses if DAT is off.

The parameter list has the following format:




Bits 16-31 of the parameter list are an unsigned binary integer that is the offset in bytes from the beginning of the second operand to a field that has the format of a PSW and from which fields in the current PSW will be replaced. Bits 32-47 and 48-63 similarly are offsets to four-byte fields from which the contents of access register $B_2$ and general register $B_2$, respectively, will be replaced. Bits 0-15 must be zeros; otherwise, a specification exception is recognized.

A PSW has the following format:




Fields in the current PSW are replaced from the corresponding fields in the PSW field in the second operand. Those fields are as follows:

| PSW Bits | Field Name |
|---|---|
| 16 and 17 | Address-space control (AS) |
| 18 and 19 | Condition code (CC) |
| 20-23 | Program mask |
| 32 | Addressing mode (A) |
| 33-63 | Instruction address |

The remaining fields in the PSW field in the second operand are ignored.

Unassigned fields in the PSW may be assigned in the future and may then be among those restored by RESUME PROGRAM. Therefore, these fields in the PSW field in the second operand should contain zeros; otherwise, the program may not operate compatibly in the future.

The fields in the second operand are fetched before the contents of access register $B_2$ and general register $B_2$ are changed.

When RESUME PROGRAM is the target of an EXECUTE instruction, the parameter list immediately follows the RESUME PROGRAM instruction, not the EXECUTE instruction.

The references to the parameter list are storage-operand fetches, not instruction fetches.

**Special Conditions**

The instruction is completed only if bits 32-63 of the PSW field in the second operand are valid for placement in the current PSW. If bit 32 is zero and bits 33-39 are not all zeros, or if bit 63 is one, a specification exception is recognized.

When DAT is on, the address-space-function control, bit 15 of control register 0, must be one when the operation is to set the access-register mode; otherwise, a special-operation exception is recognized. Also, the CPU must be in the supervisor state when the operation is to set the home-space mode; otherwise, a privileged-operation exception is recognized. When DAT is off, the values of bits 16 and 17 of the PSW field in the second operand are not tested.


LzLabs GmbH. Ex. 1008-557

When the CPU is in the home-space mode either before or after the operation, but not both before and after the operation, a space-switch-event program interruption occurs after the operation is completed if any of the following is true: (1) the primary space-switch-event control, bit 0 of control register 1, is one; (2) the home space-switch-event control, bit 0 of control register 13, is one; or (3) a PER event is to be indicated.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-26 on page 10-79.

***Resulting Condition Code:*** The code is set as specified by the new condition code loaded.

***Program Exceptions:***

- Access (fetch, parameter list and operand 2)
- Operation (if the resume-program facility is not installed)
- Privileged operation (attempt to set the home-space mode when in the problem state)
- Space-switch event
- Special operation
- Specification
- Trace

**Programming Notes:**

1. As described in "Instruction Fetching" on page 5-82, the bytes of an instruction may be fetched piecemeal, and the instruction may be fetched multiple times for a single execution. Therefore, the results are unpredictable when instructions are fetched for execution from storage that is being changed by another CPU or a channel program. This warning is particularly applicable when RESUME PROGRAM is the target of EXECUTE since the EXECUTE instruction may be refetched in order to generate, from its B, X, and D fields, the address of the parameter list used by RESUME PROGRAM. If EXECUTE is refetched, there is not necessarily a test for whether storage still contains either the EXECUTE instruction or the RESUME PROGRAM instruction.
2. The storage-operand references for RESUME PROGRAM may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)



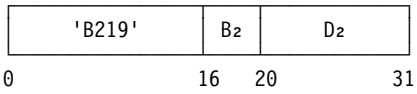
LzLabs GmbH. Ex. 1008-558

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Operation exception if the resume-program facility is not installed. |
| 8.A | Trace exceptions. |
| 8.B.1 | Access exceptions for parameter list. |
| 8.B.2 | Specification exception due to bits 0-15 of parameter list not being all zeros. |
| 8.B.3 | Access exceptions for second operand. |
| 8.B.4 | Special-operation exception due to attempt to set the access-register mode when the address-space-function control, bit 15 of control register 0, is zero. |
| 8.B.5 | Privileged-operation exception due to attempt to set the home-space mode when in the problem state. |
| 8.B.6 | Specification exception due to invalid values in bit positions 32-39 and 63 of PSW in second operand. |
| 9. | Space-switch event. |

*Figure 10-26. Priority of Execution: RESUME PROGRAM*

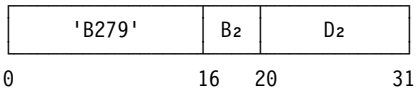
## SET ADDRESS SPACE CONTROL

SAC $D_2(B_2)$ [S]



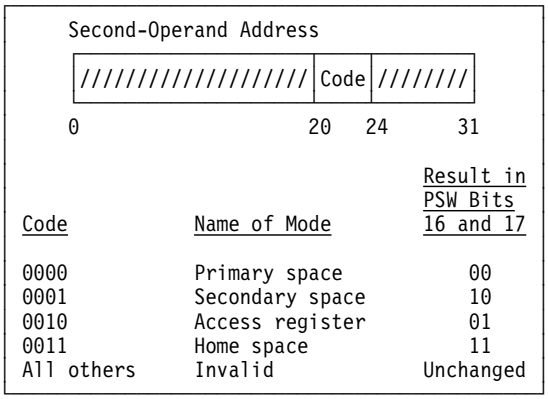

## SET ADDRESS SPACE CONTROL FAST

SACF $D_2(B_2)$ [S]




Bits 20-23 of the second-operand address are used as a code to set the address-space-control bits in the PSW. The second-operand address is not used to address data; instead, bits 20-23 form the code. Bits 0-19 and 24-31 of the second-operand address are ignored. Bits 20 and 21 of the second-operand address must be zeros; otherwise, a specification exception is recognized.

The following figure summarizes the operation of SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST:




| Code | Name of Mode | Result in PSW Bits 16 and 17 |
|---|---|---|
| 0000 | Primary space | 00 |
| 0001 | Secondary space | 10 |
| 0010 | Access register | 01 |
| 0011 | Home space | 11 |
| All others | Invalid | Unchanged |

The address-space-function control, bit 15 of control register 0, must be one when the operation is to set the access-register mode; otherwise, a special-operation exception is recognized. Also, the CPU must be in the supervisor state when the operation is to set the home-space mode; otherwise, a privileged-operation exception is recognized.


LzLabs GmbH. Ex. 1008-559

For SET ADDRESS SPACE CONTROL, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed. This function is not performed for SET ADDRESS SPACE CONTROL FAST.

**Special Conditions**

For SET ADDRESS SPACE CONTROL, the operation is performed only when the secondary-space control, bit 5 of control register 0, is one and DAT is on. When either the secondary-space control is zero or DAT is off, a special-operation exception is recognized. The same rules apply also to SET ADDRESS SPACE CONTROL FAST, except that whether the secondary-space control is tested is unpredictable.

When the CPU is in the home-space mode either before or after the operation, but not both before and after the operation, a space-switch-event program interruption occurs after the operation is completed if any of the following is true: (1) the primary space-switch-event control, bit 0 of control register 1, is one; (2) the home space-switch-event control, bit 0 of control register 13, is one; or (3) a PER event is to be indicated.

The priority of recognition of program exceptions for the instructions is shown in Figure 10-27.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Operation (if the set-address-space-control-fast facility is not installed, SACF only)
- Privileged operation (attempt to set the home-space mode in the problem state)
- Space-switch event
- Special operation
- Specification

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B.1 | Operation exception if the set-address-space-control-fast facility is not installed (SACF only). |
| 7.B.2 | Special-operation exception due to DAT being off. |
| 7.C | Special-operation exception due to the secondary-space control, bit 5 of control register 0, being zero. May be omitted for SET ADDRESS SPACE CONTROL FAST. |
| 8. | Privileged-operation exception due to attempt to set home-space mode when in problem state. |
| 9. | Special-operation exception due to the address-space-function control, bit 15 of control register 0, being zero on an attempt to set access-register mode. |
| 10. | Specification exception due to non-zero value in bit positions 20 and 21 of second-operand address. |
| 11. | Space-switch event. |

*Figure 10-27. Priority of Execution: SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST*

**Programming Notes:**

1. SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST are defined in such a way that the mode to be set can be placed directly in the displacement field of the instruction or can be specified from the same bit positions of a general register as those in which the mode is saved by INSERT ADDRESS SPACE CONTROL.
2. SET ADDRESS SPACE CONTROL FAST may provide better performance than SET ADDRESS SPACE CONTROL, depending on the model.
3. Because SET ADDRESS SPACE CONTROL FAST does not perform the serialization function, it does not cause copies of prefetched instructions to be discarded. To ensure predictable results after SET ADDRESS SPACE CONTROL FAST is used to switch to or from the home-space mode, the program must



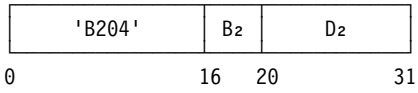
LzLabs GmbH. Ex. 1008-560

cause prefetched instructions to be discarded before an instruction is executed in a location that does not contain the same instruction in both the primary and home address spaces. The operations that cause prefetched instructions to be discarded are described in "Instruction Fetching" on page 5-82.

4. If a program stores into the instruction stream at a location following a subsequent SET ADDRESS SPACE CONTROL FAST instruction, and the SET ADDRESS SPACE CONTROL FAST instruction changes the translation mode either from or to either the access-register mode or the home-space mode, a copy of a prefetched instruction may be executed instead of the value that was stored. To avoid this situation, either SET ADDRESS SPACE CONTROL must be used instead of SET ADDRESS SPACE CONTROL FAST or some other means must be used to cause prefetched instructions to be discarded after the conceptual store occurs.

## SET CLOCK

SCK $D_2(B_2)$ [S]




The current value of the TOD clock is replaced by the contents of the doubleword designated by the second-operand address, and the clock enters the stopped state.

The doubleword operand replaces the contents of the clock, as determined by the resolution of the clock. Only those bits of the operand are set in the clock that correspond to the bit positions which are updated by the clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock. In some models, starting at or to the right of bit position 52, the rightmost bits of the second operand are ignored, and the corresponding positions of the clock which are implemented are set to zeros. When the extended-TOD-clock facility is installed, zeros are also placed in positions to the right of bit position 63 of the clock.

After the clock value is set, the clock enters the stopped state. The clock leaves the stopped state to enter the set state and resume incrementing under control of the TOD-clock-sync control, bit 2 of control register 0, of the CPU which most recently caused the clock to enter the stopped state. When the bit is zero, the clock enters the set state at the completion of the instruction. When the bit is one, the clock remains in the stopped state until the bit is set to zero, until another CPU executes a SET CLOCK instruction affecting the clock, or until any other running TOD clock in the configuration is incremented to a value of all zeros in bit positions 32 through the rightmost bit position that is incremented when the clock is running. If an external time reference (ETR) is installed, a signal from the ETR may be used to set the set state from the stopped state.

The value of the clock is changed and the clock is placed in the stopped state only if the manual TOD-clock control of any CPU in the configuration is set to the enable-set position or the TOD-clock-control-override control, bit 10 of control register 14, is one. The TOD-clock-control-override control is available if the TOD-clock-control-override facility it installed. If the TOD-clock control of all CPUs is set to the secure position and the TOD-clock-control-override facility is not installed or the TOD-clock-control-override control is zero, the value and state of the clock are not changed. Whether the clock is set or remains unchanged is distinguished by condition codes 0 and 1, respectively.

When the clock is not operational, the value and state of the clock are not changed, regardless of the settings of the TOD-clock control and the TOD-clock-control-override control, and condition code 3 is set.

**Special Conditions**

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Clock value set
1 Clock value secure
2 --
3 Clock in not-operational state


LzLabs GmbH. Ex. 1008-561

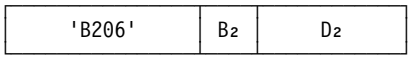
***Program Exceptions:***

- Access (fetch, operand 2)
- Privileged operation
- Specification

**Programming Note:** In an installation with more than one CPU, each CPU may have a separate TOD clock, or more than one CPU may share a TOD clock, depending on the model. When multiple TOD clocks exist, special procedures are required to synchronize the clocks. See "TOD-Clock Synchronization" on page 4-34.

## SET CLOCK COMPARATOR

SCKC $D_2(B_2)$ [S]




The current value of the clock comparator is replaced by the contents of the doubleword designated by the second-operand address.

Only those bits of the operand are set in the clock comparator that correspond to the bit positions to be compared with the TOD clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock comparator.

**Special Conditions**

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

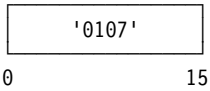
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2)
- Privileged operation
- Specification

## SET CLOCK PROGRAMMABLE FIELD

SCKPF [E]




Bits 16-31 of general register 0 are placed in the corresponding bit positions of the TOD programmable register. Zeros are placed in bit positions 0-15 of the TOD programmable register.

**Special Conditions**

Bits 0-15 of general register 0 must be zeros; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

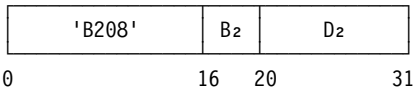
***Program Exceptions:***

- Privileged operation
- Operation (if the extended-TOD-clock facility is not installed)
- Specification

**Programming Note:** Each CPU has a TOD programmable register. The values in the TOD programmable registers should be unique within a multiple-configuration system.

## SET CPU TIMER

SPT $D_2(B_2)$ [S]




The current value of the CPU timer is replaced by the contents of the doubleword designated by the second-operand address.

Only those bits of the operand are set in the CPU timer that correspond to the bit positions to be updated; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the CPU timer.



LzLabs GmbH. Ex. 1008-562

**Special Conditions**

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

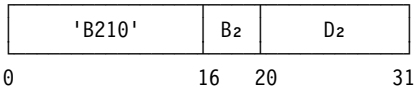
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2)
- Privileged operation
- Specification

# SET PREFIX

SPX D2(B2) [S]




The contents of the prefix register are replaced by the contents of bit positions 1-19 of the word at the location designated by the second-operand address. The ART-lookaside buffer (ALB) and translation-lookaside buffer (TLB) of this CPU are cleared of entries.

After the second operand is fetched, the value is tested for validity before it is used to replace the contents of the prefix register. Bits 1-19 of the operand with 12 rightmost zeros appended are used as an absolute address of the 4K-byte new prefix area in storage. The prefix value is treated as a 31-bit address, regardless of the addressing mode specified by bit 32 of the current PSW. The 4K-byte block within the new prefix area is accessed; if it is not available in the configuration, an addressing exception is recognized, and the operation is suppressed. The access to the block is not subject to protection; however, the access may cause the reference bit to be set to one.

If the operation is completed, the new prefix is used for any interruptions following the execution of the instruction and for the execution of subsequent instructions. The contents of bit positions 0 and 20-31 of the second operand are ignored.

The ART-lookaside buffer (ALB) and translation-lookaside buffer (TLB) are cleared of entries. The ALB and TLB appear cleared of their original contents, beginning with the fetching of the next sequential instruction.

A serialization function is performed before or after the second operand is fetched and again after the operation is completed.

**Special Conditions**

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

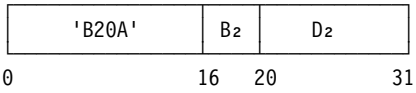
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2)
- Addressing (new prefix area)
- Privileged operation
- Specification

# SET PSW KEY FROM ADDRESS

SPKA D2(B2) [S]




The four-bit PSW key, bits 8-11 of the current PSW, is replaced by bits 24-27 of the second-operand address.

The second-operand address is not used to address data; instead, bits 24-27 of the address form the new PSW key. Bits 0-23 and 28-31 of the second-operand address are ignored.

**Special Conditions**

In the problem state, the execution of the instruction is subject to control by the PSW-key mask in control register 3. When the bit in the PSW-key mask corresponding to the PSW-key value to be set is one, the instruction is executed successfully. When the selected bit in the PSW-key mask is zero, a privileged-operation exception is recognized. In the supervisor state, any value for the PSW key is valid.


LzLabs GmbH. Ex. 1008-563

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

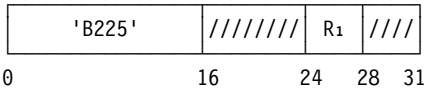
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)

**Programming Notes:**

1. The format of SET PSW KEY FROM ADDRESS permits the program to set the PSW key either from the general register designated by the $B_2$ field or from the $D_2$ field in the instruction itself.
2. When one program requests another program to access a location designated by the requesting program, SET PSW KEY FROM ADDRESS can be used by the called program to verify that the requesting program is authorized to make this access, provided the storage location of the called program is not protected against fetching. The called program can perform the verification by replacing the PSW key with the requesting-program PSW key before making the access and subsequently restoring the called-program PSW key to its original value. Caution must be exercised, however, in handling any resulting protection exceptions since such exceptions may cause the operation to be terminated. See TEST PROTECTION and the associated programming notes for an alternative approach to the testing of addresses passed by a calling program.

# SET SECONDARY ASN

SSAR $R_1$ [RRE]




The ASN specified in bit positions 16-31 of general register $R_1$ replaces the secondary ASN in control register 3, and the segment-table designation corresponding to that ASN replaces the SSTD in control register 7.

Bits 16-23 and 28-31 of the instruction are ignored.

The contents of bit positions 16-31 of general register $R_1$ are called the new ASN. The contents of bit positions 0-15 of the register are ignored.

First the new ASN is compared with the current PASN. If the new ASN is equal to the PASN, the operation is called SET SECONDARY ASN to current primary (SSAR-cp). If the new ASN is not equal to the current PASN, the operation is called SET SECONDARY ASN with space switching (SSAR-ss). The SSAR-cp and SSAR-ss operations are depicted in Figure 10-29 on page 10-86.

**SET SECONDARY ASN to Current Primary (SSAR-cp)**

The new ASN replaces the SASN, bits 16-31 of control register 3; the PSTD, bits 0-31 of control register 1, replaces the SSTD, bits 0-31 of control register 7; and the operation is completed.

**SET SECONDARY ASN with Space Switching (SSAR-ss)**

The new ASN is translated by means of the ASN translation tables, and then the current AX, bits 0-15 of control register 4, is used to test whether the program is authorized to access the specified ASN.

The new ASN is translated by means of a two-level table lookup. Bits 0-9 of the new ASN (bits 16-25 of the register) are a 10-bit AFX which is used to select an entry from the ASN first table. Bits 10-15 of the new ASN (bits 26-31 of the register) are a six-bit ASX which is used to select an entry from the ASN second table. The two-level lookup is described in “ASN Translation” on page 3-18. The exceptions associated with ASN translation are collectively called “ASN-translation exceptions.” These exceptions and their priority are described in Chapter 6, “Interruptions.”

The ASN-second-table entry (ASTE) obtained as a result of the second lookup contains the segment-table designation and the authority-table origin and length associated with the ASN.

The authority-table origin from the ASTE is used as a base for a third table lookup. The current authorization index, bits 0-15 of control register 4, is used, after it has been checked against the authority-table length, as the index to locate the



LzLabs GmbH. Ex. 1008-564

entry in the authority table. The authority-table lookup is described in "ASN Authorization" on page 3-23.

The new ASN, bits 16-31 of general register $R_1$, replaces the SASN, bits 16-31 of control register 3. The segment-table designation, bits 64-95 of the ASTE, replaces the SSTD, bits 0-31 of control register 7.

The description in this paragraph applies if the subspace-group facility is installed and the address-space-function control, bit 15 of control register 0, is one. After the new SSTD has been placed in control register 7, if (1) the subspace-group-control bit, bit 22, in the SSTD is one, (2) the dispatchable unit is subspace active, and (3) the ASTE obtained by ASN translation is the ASTE for the base space of the dispatchable unit, then bits 1-23 and 25-31 of the SSTD in control register 7 are replaced by bits 1-23 and 25-31 of the STD in the ASTE for the subspace in which the dispatchable unit last had control. Further details are in "Subspace-Replacement Operations" on page 5-59.

**SET SECONDARY ASN Serialization**

For both the SSAR-cp and SSAR-ss operations, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

**Special Conditions**

The operation is performed only when the ASN-translation control, bit 12 of control register 14, is one and DAT is on. When either the ASN-translation-control bit is zero or DAT is off, a special-operation exception is recognized.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-28.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (authority-table entry, SSAR-ss only)
- ASN translation (SSAR-ss only)
- Secondary authority (SSAR-ss only)
- Special operation
- Subspace replacement (SSAR-ss only, and only when subspace-group facility is installed and address-space-function control is one)
- Trace

```
1.-6.  Exceptions with the same priority as the priority of program-
       interruption conditions for the general case.

7.A    Access exceptions for second instruction halfword.

7.B    Special-operation exception due to DAT being off, or the ASN-
       translation control, bit 12 of control register 14, being
       zero.

8.A    Trace exceptions.

8.B.1  ASN-translation exceptions (SSAR-ss only).

       Note:  Subspace-replacement exceptions, which are not shown
       in detail in this figure, can occur with any priority after
       8.B.1.

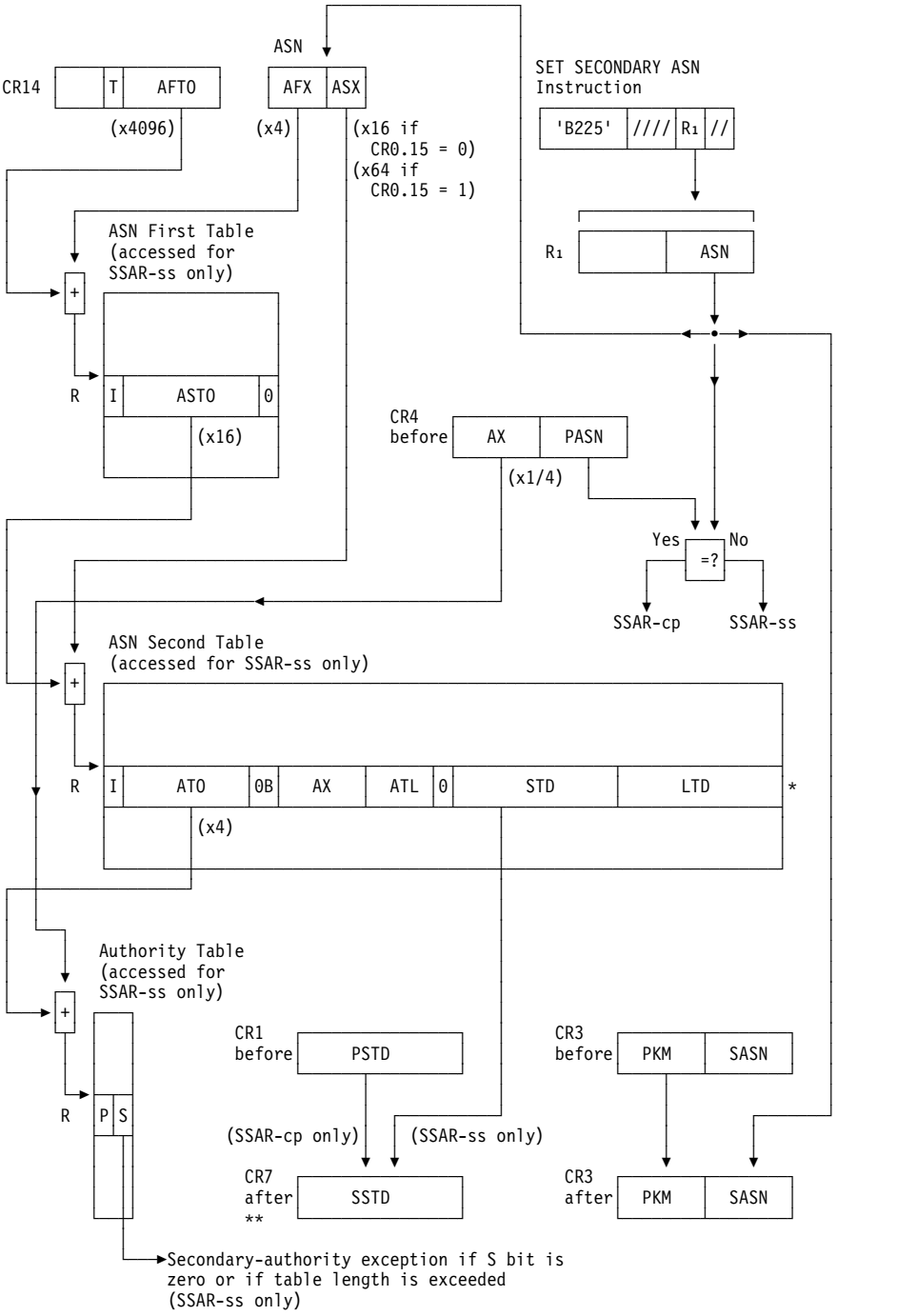
8.B.2  Secondary-authority exception due to authority-table entry
       being outside table (SSAR-ss only).

8.B.3  Addressing exception for access to authority-table entry
       (SSAR-ss only).

8.B.4  Secondary-authority exception due to S bit in authority-
       table entry being zero (SSAR-ss only).
```

*Figure 10-28. Priority of Execution: SET SECONDARY ASN*


LzLabs GmbH. Ex. 1008-565




R: Address is real.
*: ASTE is 64 bytes if CR0.15 = 1; last 48 bytes are not shown.
**: For SSAR-ss only, if subspace-group facility installed and CR0.15 = 1, bits 1-23 and 25-31 of SSTD may be replaced from a subspace STD.

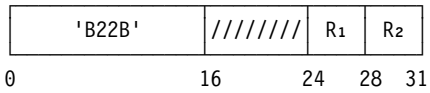
*Figure 10-29. Execution of SET SECONDARY ASN*



LzLabs GmbH. Ex. 1008-566

## SET STORAGE KEY EXTENDED

SSKE R1,R2 [RRE]




The storage key for the 4K-byte block that is addressed by the contents of general register R2 is replaced by bits from general register R1.

Bits 16-23 of the instruction are ignored.

In the 24-bit addressing mode, bits 8-19 of general register R2 designate a 4K-byte block in real storage, and bits 0-7 and 20-31 of the register are ignored. In the 31-bit addressing mode, bits 1-19 of general register R2 designate a 4K-byte block in real storage, and bits 0 and 20-31 of the register are ignored.

Because it is a real address, the address designating the storage block is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The new seven-bit storage-key value is obtained from bit positions 24-30 of general register R1. The contents of bit positions 0-23 and 31 of the register are ignored.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

For any store access, by any CPU or channel program, completed to the designated 4K-byte block either before or after the execution of this instruction, the associated setting of the reference and change bits to one in the storage key for the block also is completed before or after, respectively, the execution of this instruction.

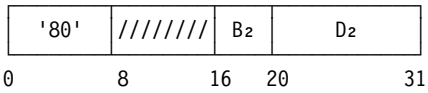
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (address specified by general register R2)
- Privileged operation

## SET SYSTEM MASK

SSM D2(B2) [S]




Bits 0-7 of the current PSW are replaced by the byte at the location designated by the second-operand address.

Bits 8-15 of the instruction are ignored.

**Special Conditions**

When the SSM-suppression-control bit, bit 1 of control register 0, is one and the CPU is in the supervisor state, a special-operation exception is recognized.

The value to be loaded into the PSW is not checked for validity before loading. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, if the contents of bit positions 0 and 2-4 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2. The specification exception, which is listed as a program exception for this instruction, is described in “Early Exception Recognition” on page 6-9. This exception may be considered as caused by execution of this instruction or as occurring early in the process of preparing to execute the subsequent instruction.

The operation is suppressed on all addressing and protection exceptions.

***Condition Code:*** The code remains unchanged.

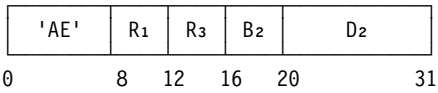
***Program Exceptions:***

- Access (fetch, operand 2)
- Privileged operation
- Special operation
- Specification


LzLabs GmbH. Ex. 1008-567

## SIGNAL PROCESSOR

SIGP R1,R3,D2(B2) [RS]




An eight-bit order code and, if called for, a 32-bit parameter are transmitted to the CPU designated by the CPU address contained in the third operand. The result is indicated by the condition code and may be detailed by status assembled in the first-operand location.

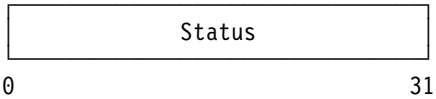
The second-operand address is not used to address data; instead, bits 24-31 of the address contain the eight-bit order code. Bits 0-23 of the second-operand address are ignored. The order code specifies the function to be performed by the addressed CPU. The assignment and definition of order codes appear in "CPU Signaling and Response" on page 4-45.

The 16-bit unsigned binary integer contained in bit positions 16-31 of general register R3 forms the CPU address. Bits 0-15 of the register are ignored.

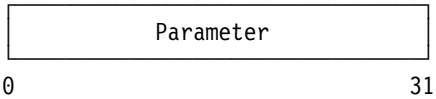
The general register containing the 32-bit parameter is R1 or R1+1, whichever is the odd-numbered register. It depends on the order code whether a parameter is provided and for what purpose it is used.

The operands just described have the following formats:

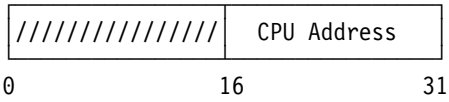
General register designated by R1:



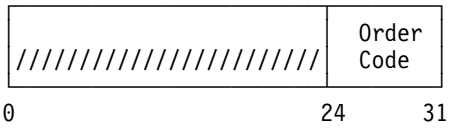

General register designated by R1 or R1 + 1, whichever is the odd-numbered register:




General register designated by R3:




Second-operand address:




A serialization function is performed before the operation begins and again after the operation is completed.

When the order code is accepted and no nonzero status is returned, condition code 0 is set. When status information is generated by this CPU or returned by the addressed CPU, the status is placed in general register R1, and condition code 1 is set.

When the access path to the addressed CPU is busy, or the addressed CPU is operational but in a state where it cannot respond to the order code, condition code 2 is set.

When the addressed CPU is not operational (that is, it is not provided in the installation, it is not in the configuration, it is in any of certain customer-engineer test modes, or its power is off), condition code 3 is set.

***Resulting Condition Code:***

0 Order code accepted
1 Status stored
2 Busy
3 Not operational

***Program Exceptions:***

- Privileged operation

**Programming Notes:**

1. A more detailed discussion of the condition-code settings for SIGNAL PROCESSOR is contained in "CPU Signaling and Response" on page 4-45.
2. To ensure that presently written programs will be executed properly when new facilities using additional bits are installed, only zeros should appear in the unused bit positions of the



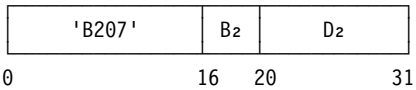
LzLabs GmbH. Ex. 1008-568

second-operand address and in bit positions 0-15 of general register $R_3$.

3. Certain SIGNAL PROCESSOR orders are provided with the expectation that they will be used primarily in special circumstances. Such orders may be implemented with the aid of an auxiliary maintenance or service processor, and, thus, the execution time may take several seconds. Unless all of the functions provided by the order are required, combinations of other orders, in conjunction with appropriate programming support, can be expected to provide a specific function more rapidly. The emergency-signal, external-call, and sense orders are the only orders which are intended for frequent use. The following orders are intended for infrequent use, and performance therefore may be much slower than for frequently used orders: restart, set prefix, store status at address, store extended status at address, start, stop, stop and store status, set architecture, and all the reset orders. An alternative to the set-prefix order, for faster performance when the receiving CPU is not already stopped, is the use of the emergency-signal or external-call order, followed by the execution of a SET PREFIX instruction on the addressed CPU. Clearing the TLB of entries is ordinarily accomplished more rapidly through the use of the emergency-signal or external-call order, followed by execution of the PURGE TLB instruction on the addressed CPU, than by use of the set-prefix order.

# STORE CLOCK COMPARATOR

STCKC $D_2(B_2)$ [S]




The current value of the clock comparator is stored at the doubleword location designated by the second-operand address.

Zeros are provided for the rightmost bit positions of the clock comparator that are not compared with the TOD clock.

**Special Conditions**

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

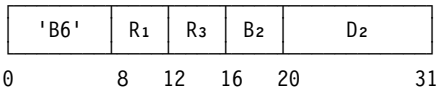
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Privileged operation
- Specification

# STORE CONTROL

STCTL $R_1,R_3,D_2(B_2)$ [RS]




The set of control registers starting with control register $R_1$ and ending with control register $R_3$ is stored at the locations designated by the second-operand address.

The storage area where the contents of the control registers are placed starts at the location designated by the second-operand address and continues through as many storage words as the number of control registers specified. The contents of the control registers are stored in ascending order of their register numbers, starting with control register $R_1$ and continuing up to and including control register $R_3$, with control register 0 following control register 15. The contents of the control registers remain unchanged.

**Special Conditions**

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

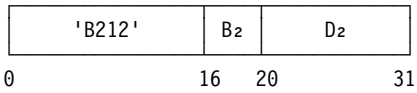
***Program Exceptions:***

- Access (store, operand 2)
- Privileged operation
- Specification


LzLabs GmbH. Ex. 1008-569

## STORE CPU ADDRESS

```
STAP   D2(B2)            [S]
```




The 16-bit unsigned binary integer by which this CPU is identified in a multiprocessing configuration is stored at the halfword location designated by the second-operand address.

**Special Conditions**

The operand must be designated on a halfword boundary; otherwise, a specification exception is recognized.

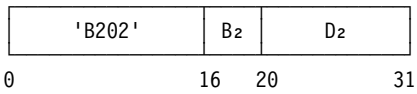
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Privileged operation
- Specification

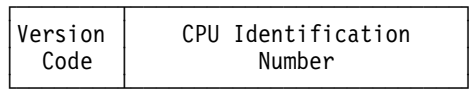
## STORE CPU ID

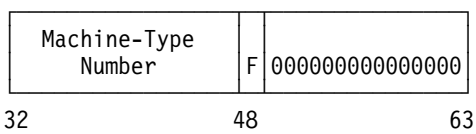
```
STIDP  D2(B2)            [S]
```




Information identifying either (1) the CPU or (2) the configuration and the logical partition in which the program is being executed is stored at the doubleword location designated by the second-operand address.

The information stored has the following format:




Bit positions 0-7 contain the version code. The format and significance of the version code depend on the model.

Bit positions 8-31 contain the CPU identification number, consisting of six four-bit digits. Some or all of these digits are selected from the physical serial number stamped on the CPU.

The format bit (F) in bit position 48 specifies the format of the first two digits of the CPU identification number.

When the format bit is zero, the contents of the CPU-identification-number field, in conjunction with the machine-type number, permit unique identification of the CPU. When the format bit is one, the CPU-identification number identifies the system configuration as opposed to an individual CPU within the configuration, and it identifies the logical partition in which the program is being executed.

Bit positions 32-47 contain the machine-type number of the CPU. Bit positions 49-63 contain zeros.

**Special Conditions**

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Privileged operation
- Specification

**Programming Notes:**

1. The program should allow for the possibility that the CPU identification number may contain the digits A-F as well as the digits 0-9.
2. When the format bit in bit position 48 of the second operand is zero, the CPU identification number, in conjunction with the machine-type number, provides a unique CPU identification that can be used in associating results with an individual machine.

   When the format bit is one, the CPU-identification number identifies the system configuration as opposed to an individual CPU within the configuration, and it



LzLabs GmbH. Ex. 1008-570

identifies the requesting logical partition. The CPU identity can be obtained using the STORE CPU ADDRESS or the STORE SYSTEM INFORMATION instruction.

3. In versions of this publication prior to SA22-7201-03, the machine-type-number field was called the model-number field.

4. When the version code is nonzero, it is usually indicative of the model number of the model and the number of CPUs contained in the model. The version-code values for a machine type are described in the "Functional Characteristics" or "System Overview" manual for the machine type.

   The STORE SYSTEM INFORMATION instruction can be used to determine the model number and the number of CPUs in the model.

5. The CPU identification number has the hex format:

   - "Annnnn" in the basic mode, or
   - "LPnnnn" in the LPAR (logically-partitioned) mode, when the format bit is zero, or
   - "PPnnnn" in the LPAR mode, when the format bit is one.

   Where:

   - A is the CPU address of the CPU.
   - L is a logical CPU address.
   - P is a logical-partition identifier.
   - PP is the user partition identifier (UPID). The UPID is an eight-bit unsigned binary integer bound to a logical partition.
   - n is a digit derived from the serial number of the CPU.

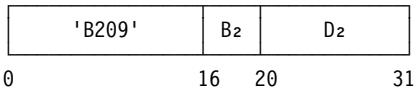
   The terminology above that is not defined in this publication is defined in the machine manuals.

6. The format bit is always stored as a zero in the basic mode.

7. Model z800 and z900 machines always store the format bit as a zero.

# STORE CPU TIMER

STPT D2(B2) [S]




The current value of the CPU timer is stored at the doubleword location designated by the second-operand address.

Zeros are provided for the rightmost bit positions that are not updated by the CPU timer.

**Special Conditions**

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

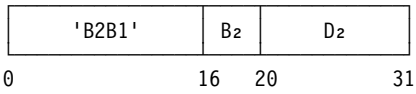
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Privileged operation
- Specification

# STORE FACILITY LIST

STFL D2(B2) [S]




A list of bits providing information about facilities is stored in the word at real address 200. The bits have meanings as follows.

**Bit Meaning when Bit Is One**

0 The instructions marked "N3" in the instruction-summary figures in Chapters 7 and 10 are installed.

1 The z/Architecture architectural mode is installed.

2 The z/Architecture architectural mode is active. When this bit is zero, the ESA/390 architectural mode is active.


LzLabs GmbH. Ex. 1008-571

| 3 INVALIDATE DAT TABLE ENTRY (IDTE) is installed in the z/Architecture architectural mode. Unless bit 4 is one, IDTE simply purges all TLBs.

| 4 INVALIDATE DAT TABLE ENTRY (IDTE) performs the invalidation-and-clearing operation by selectively clearing combined region-and-segment-table entries when a segment-table entry or entries are invalidated. IDTE also performs the clearing-by-ASCE operation. Bit 3 is one if bit 4 is one.

| 5 INVALIDATE DAT TABLE ENTRY (IDTE) performs the invalidation-and-clearing operation by selectively clearing combined region-and-segment-table entries when a region-table entry or entries are invalidated. Bits 3 and 4 are ones if bit 5 is one.

16 The extended-translation facility 2 is installed.

| 17 The message-security assist is installed.

| 18 The long-displacement facility is installed in the z/Architecture architectural mode.

| 19 The long-displacement facility has high performance. Bit 18 is one if bit 19 is one.

| 20 The HFP-multiply-add/subtract facility is installed.

| A bit is set to one regardless of the current architectural mode if its meaning is true. A meaning applies to the current architectural mode unless it is said to apply to a specific architectural mode.

| Bits 6-15 and 21-31 are reserved for indication of new facilities. The bits are currently stored as zeros but may be stored as ones in the future.

The second-operand address is ignored but should be zero to permit possible future extensions.

Key-controlled and low-address protection do not apply.

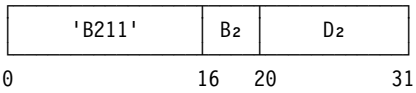
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

| • Operation (if z/Architecture is not installed)
• Privileged operation

# STORE PREFIX

STPX D2(B2) [S]




The contents of the prefix register are stored at the word location designated by the second-operand address. Zeros are provided for bit positions 0 and 20-31.

**Special Conditions**

The operand must be designated on a word boundary; otherwise, a specification exception is recognized.

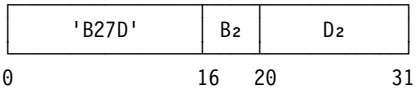
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Privileged operation
- Specification

# STORE SYSTEM INFORMATION

STSI D2(B2) [S]




Depending on a function code in general register 0, either an identification of the level of the configuration executing the program is placed in general register 0 or information about a component or components of a configuration is stored in a system-information block (SYSIB). When information about a component or components is requested, the information is specified by further contents of general register 0 and by contents of general register 1. The SYSIB, if any, is designated by the second-operand address.

The machine is considered to provide one, two, or three levels of configuration. The levels are:

1. The basic machine, which is the machine as if it were operating in the basic mode.
2. A logical partition, which is provided if the machine is operating in the LPAR, or logically-partitioned, mode. A logical partition is pro-



LzLabs GmbH. Ex. 1008-572

vided by the LPAR hypervisor, which is a part of the machine. A basic machine exists even when the machine is operating in the LPAR mode.

3. A virtual machine, which is provided by a virtual-machine (VM) control program that is executed either by the basic machine or in a logical partition. A virtual machine may itself execute a VM control program that provides a higher-level (more removed from the basic machine) virtual machine, which also is considered a level-3 configuration.

The terms basic mode, LPAR mode, logical partition, hypervisor, and virtual machine, and any other terms related specifically to those terms, are not defined in this publication; they are defined in the machine manuals.

A program being executed by a level-1 configuration (the basic machine) can request information about that configuration. A program being executed by a level-2 configuration (in a logical partition) can request information about the logical partition and about the underlying basic machine. A program being executed by a level-3 configuration (a virtual machine) can request information about the virtual machine and about the one or two underlying levels; a basic machine is always underlying, and a logical partition may or may not be between the basic machine and the virtual machine. When information about a virtual machine is requested, information is provided about the configuration executing the program and about any underlying level or levels of virtual machine. In any of these cases, information is provided about a level only if the level implements the instruction.

The function code determining the operation is an unsigned binary integer in bit positions 0-3 of general register 0 and is as follows:

| Function Code | Information Requested |
|---|---|
| 0 | Current-configuration-level number |
| 1 | Information about level 1 (the basic machine) |
| 2 | Information about level 2 (a logical partition) |
| 3 | Information about level 3 (a virtual machine) |
| 4-15 | None; codes are reserved |

**Invalid Function Code**

The level of the configuration executing the program is called the current level. The configuration level specified by a nonzero function code is called the specified level. When the specified level is numbered higher than the current level, then the function code is called invalid, the condition code is set to 3, and no other action (including checking) is performed.

**Valid Function Code**

When the function code is equal to or less than the number of the current level, it is called valid. In this case, bits 4-23 of general register 0 and bits 0-15 of general register 1 must be zero; otherwise, a specification exception is recognized.

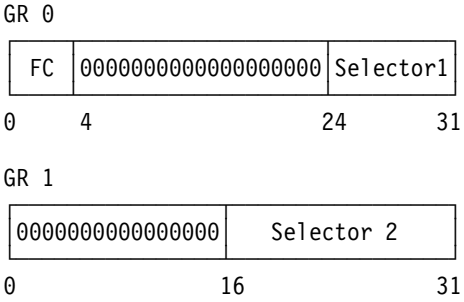
When the function code is 0, an unsigned binary integer identifying the current configuration level (1 for basic machine, 2 for logical partition, or 3 for virtual machine) is placed in bit positions 0-3 of general register 0, the condition code is set to 0, and no further action is performed.

When the function code is valid and nonzero, general registers 0 and 1 contain additional specifications about the information requested, as follows:

- Bit positions 24-31 of general register 0 contain an unsigned binary integer, called *selector 1,* that specifies a component or components of the specified configuration.
- Bit positions 16-31 of general register 1 contain an unsigned binary integer, called *selector 2,* that specifies the type of information requested.

The contents of general registers 0 and 1 are as follows:


LzLabs GmbH. Ex. 1008-573




When the function code is valid and nonzero, information may be stored in a system-information block (SYSIB) beginning at the second-operand location. The SYSIB is 4K bytes and must begin at a 4K-byte boundary; otherwise, a specification exception may be recognized, depending on selector 1 and selector 2 and on whether access exceptions are recognized due to references to the SYSIB (see "Special Conditions").

Selector 1 can have values as follows:

| Selector 1 | Information Requested |
|---|---|
| 0 | None; selector is reserved |
| 1 | Information about the specified configuration level |
| 2 | Information about one or more CPUs in the specified configuration level |
| 3-255 | None; selectors are reserved |

When selector 1 is 1, selector 2 can have values as follows:

| Selector 2 when Selector 1 Is 1 | Information Requested |
|---|---|
| 0 | None; selector is reserved |
| 1 | Information about the specified configuration level |
| 2-65,535 | None; selectors are reserved |

When selector 1 is 2, selector 2 can have values as follows:

| Selector 2 when Selector 1 Is 2 | Information Requested |
|---|---|
| 0 | None; selector is reserved |
| 1 | Information about the CPU executing the program in the specified configuration level |
| 2 | Information about all CPUs in the specified configuration level |
| 3-65,535 | None; selectors are reserved |

Only certain combinations of the function code, selector 1, and selector 2 are valid, as shown in Figure 10-30.

| Function Code | Selector 1 | Selector 2 | Information Requested about |
|---|---|---|---|
| 0 | - | - | Current-configuration-level number |
| 1 | 1 | 1 | Basic-machine configuration |
| 1 | 2 | 1 | Basic-machine CPU |
| 1 | 2 | 2 | Basic-machine CPUs |
| 2 | 2 | 1 | Logical-partition CPU |
| 2 | 2 | 2 | Logical-partition CPUs |
| 3 | 2 | 2 | Virtual-machine CPUs |

**Explanation:**

\- Ignored.

*Figure 10-30. Valid Function-Code, Selector-1, and Selector-2 Combinations for STORE SYSTEM INFORMATION*

When the specified function-code, selector-1, and selector-2 combination is invalid (is other than as shown in Figure 10-30), or if it is valid but the requested information is not available because the specified level does not implement or does not fully implement the instruction or because a necessary part of the level is uninstalled or not initialized, and provided that an exception is not recognized (see "Special Conditions"), the condition code is set to 3. When the function code is nonzero, the combination is valid, the requested information is available, and there is no exception, the requested information is stored in a system-information block (SYSIB) at the second-operand address.



LzLabs GmbH. Ex. 1008-574

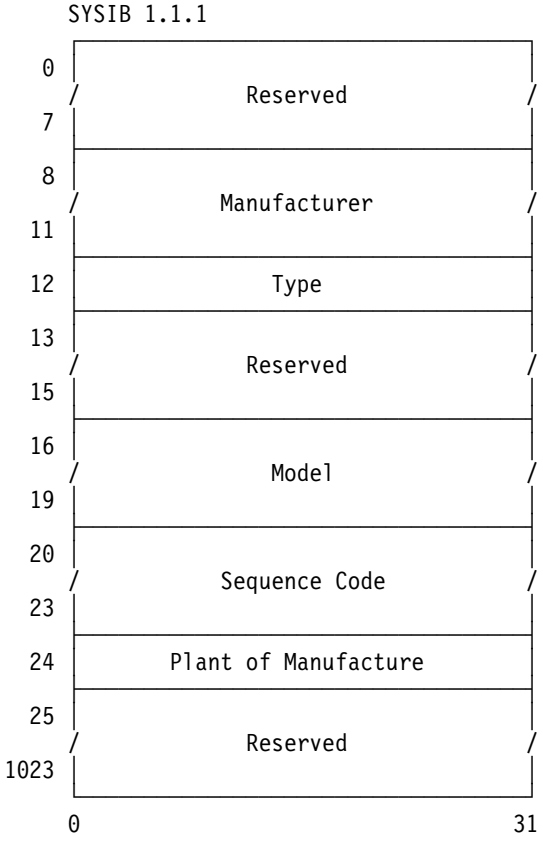
Some or all of the SYSIB may be fetched before it is stored.

A SYSIB may be identified in references by means of "SYSIB fc.s1.s2," where "fc," "s1," and "s2" are the values of a function code, selector 1, and selector 2, respectively.

Following sections describe the defined SYSIBs by means of figures and related text. In the figures, the offsets shown on the left are word values. "The configuration" refers to the configuration level specified by the function code (the configuration level about which information is requested).

**SYSIB 1.1.1 (Basic-Machine Configuration)**

SYSIB 1.1.1 has the following format:




***Reserved:*** The contents of words 0-7, 13-15, and 25-63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

***Manufacturer:*** Words 8-11 contain the 16-character (0-9 or uppercase A-Z) EBCDIC name of the manufacturer of the configuration. The name is left justified with trailing blanks if necessary.

***Type:*** Word 12 contains the four-character (0-9) EBCDIC type number of the configuration. (This is called the machine-type number in the definition of STORE CPU ID.)

***Model:*** Words 16-19 contain the 16-character (0-9 or uppercase A-Z) EBCDIC model identification of the configuration. The model identification is left justified with trailing blanks if necessary. (This is called the model number in programming note 4 on page 10-91 of STORE CPU ID.)

***Sequence Code:*** Words 20-23 contain the 16-character (0-9 or uppercase A-Z) EBCDIC sequence code of the configuration. The sequence code is right justified with leading EBCDIC zeros if necessary.

***Plant of Manufacture:*** Word 24 contains the four-character (0-9 or uppercase A-Z) EBCDIC code that identifies the plant of manufacture for the configuration. The code is left justified with trailing blanks if necessary.

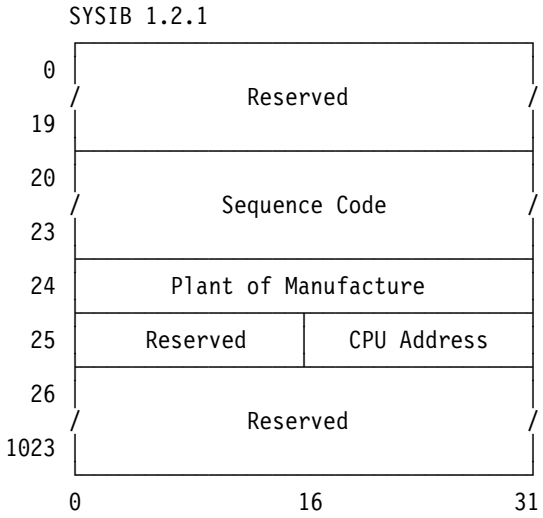
**Programming Note:** The fields of the SYSIB 1.1.1 are similar to those of the node descriptor described in the publication *Common I/O-Device Commands and Self Description,* SA22-7204. However, the contents of the SYSIB fields may not be identical to the contents of the corresponding node-descriptor fields because the SYSIB fields:

- Allow more characters.
- Are more flexible regarding the type of characters allowed.
- Provide information that is justified differently within the field.
- May not use the same method to determine the contents of fields such as the sequence-code field.


LzLabs GmbH. Ex. 1008-575

**SYSIB 1.2.1 (Basic-Machine CPU)**

SYSIB 1.2.1 has the following format:




***Reserved:*** The contents of words 0-19, bytes 0 and 1 of word 25, and words 26-63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

***Sequence Code:*** Words 20-23 contain the 16-character (0-9 or uppercase A-Z) EBCDIC sequence code of the configuration. The code is right justified with leading EBCDIC zeros if necessary.

***Plant of Manufacture:*** Word 24 contains the four-character (0-9 or uppercase A-Z) EBCDIC code that identifies the plant of manufacture for the configuration. The code is left justified with trailing blanks if necessary.

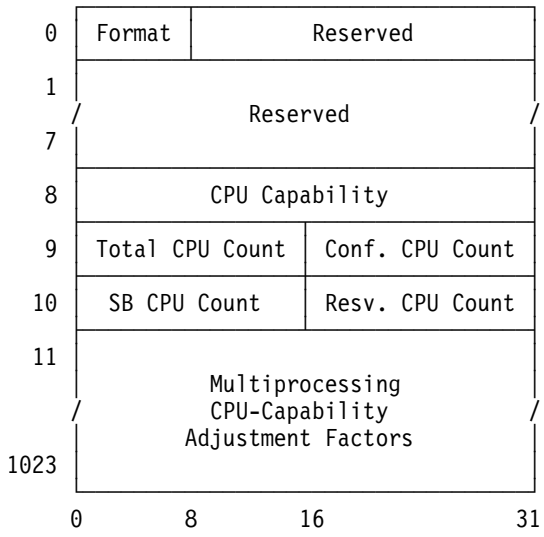
***CPU Address:*** Bytes 2 and 3 of word 25 contain the CPU address by which this CPU is identified in a multiprocessing configuration. The CPU address is a 16-bit unsigned binary integer. The CPU address is the same as is stored by STORE CPU ADDRESS when the program is executed by a machine operating in the basic mode.

**Programming Note:** Multiple CPUs in the same configuration have the same sequence code, and it is necessary to use other information, such as the CPU address, to establish a unique CPU identity. The sequence code returned for a basic-machine CPU and a logical-partition CPU are identical and have the same value as the sequence code returned for the basic-machine configuration.

**SYSIB 1.2.2 (Basic-Machine CPUs)**

The format field in byte 0 of word 0 determines the format of the SYSIB.

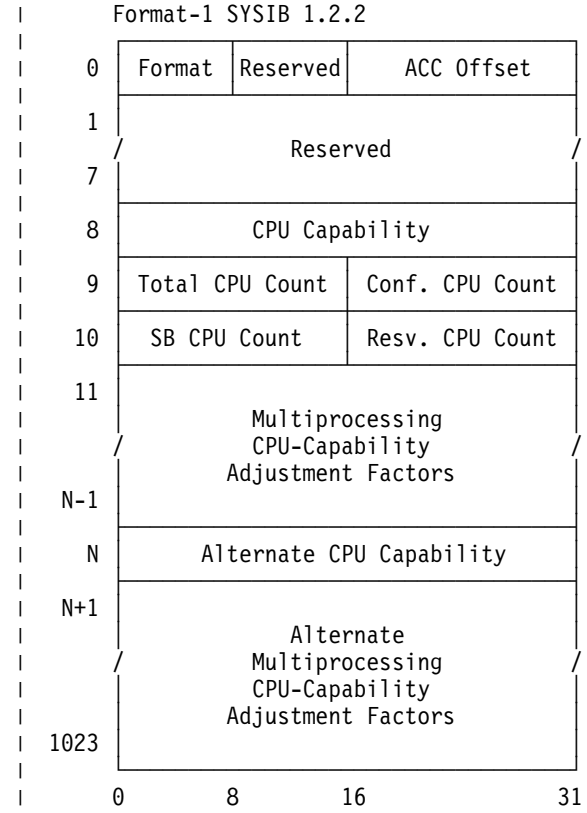
When the format field has a value of zero, SYSIB 1.2.2 has a format-0 layout as follows:




When the format field has a value of one, SYSIB 1.2.2 has a format-1 layout as follows:



LzLabs GmbH. Ex. 1008-576




***Reserved:*** When the format field contains a value of zero, the contents of bytes 1-3 of word 0 and words 1-7 are reserved and stored as zeros. When the format field contains a value of one, the contents of byte 1 of word 0 and words 1-7 are reserved and stored as zeros.

When less than 64 words are needed to contain the information for all the CPUs, the portion of the SYSIB following the adjustment-factor list in a format-0 SYSIB or the alternate-adjustment-factor list in a format-1 SYSIB, up to word 63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

When 64 or more words are needed to contain the information for all the CPUs, the portion of the SYSIB following the adjustment-factor list in a format-0 SYSIB or the alternate-adjustment-factor list in a format-1 SYSIB, up to word 1023 are reserved and may be stored as zeros or may remain unchanged.

***Format:*** Byte 0 of word 0 contains an 8-bit unsigned binary integer that specifies the format of SYSIB 1.2.2.

***Alternate-CPU-Capability Offset:*** When the format field has a value of one, bytes 2-3 of word 0 contain a 16-bit unsigned binary integer that specifies the offset in bytes of the alternate-CPU-capability field in the SYSIB.

***CPU Capability:*** Word 8 contains a 32-bit unsigned binary integer that specifies the capability of one of the CPUs in the configuration. There is no formal description of the algorithm used to generate this integer. The integer is used as an indication of the capability of the CPU relative to the capability of other CPU models.

The capability value applies to each of the CPUs in the configuration. That is, all CPUs in the configuration have the same capability.

***Total CPU Count:*** Bytes 0 and 1 of word 9 contain a 16-bit unsigned binary integer that specifies the total number of CPUs in the configuration. This number includes all CPUs in the configured state, the standby state, or the reserved state.

***Configured CPU Count:*** Bytes 2 and 3 of word 9 contain a 16-bit unsigned binary integer that specifies the number of CPUs that are in the configured state. A CPU is in the configured state when it is in the configuration and available to be used to execute programs.

***Standby CPU Count:*** Bytes 0 and 1 of word 10 contain a 16-bit unsigned binary integer that specifies the number of CPUs that are in the standby state. A CPU is in the standby state when it is in the configuration, is not available to be used to execute programs, and can be made available by issuing instructions to place it in the configured state.

***Reserved CPU Count:*** Bytes 2 and 3 of word 10 contain a 16-bit unsigned binary integer that specifies the number of CPUs that are in the reserved state. A CPU is in the reserved state when it is in the configuration, is not available to be used to execute programs, and cannot be made available by issuing instructions to place it in the configured state. (It may be possible to place a reserved CPU in the standby or configured state by means of manual actions.)


LzLabs GmbH. Ex. 1008-577

***Multiprocessing CPU-Capability Adjustment Factors:*** Beginning with bytes 0 and 1 of word 11, the SYSIB contains a series of contiguous two-byte fields, each containing a 16-bit unsigned binary integer that is an adjustment factor (percentage) for the value contained in the CPU-capability field.

The number of adjustment-factor fields is one less than the number of CPUs specified in the total-CPU-count field. The adjustment-factor fields correspond to configurations with increasing numbers of CPUs in the configured state. The first adjustment-factor field corresponds to a configuration with two CPUs in the configured state. Each successive adjustment-factor field corresponds to a configuration with a number of CPUs in the configured state that is one more than that for the preceding field.

***Alternate CPU Capability:*** When the format field has a value of one, word N contains a 32-bit unsigned binary integer that specifies the announced capability of one of the CPUs in the configuration. There is no formal description of the algorithm used to generate this integer. The integer is used as an indication of the announced capability of the CPU relative to the announced capability of other CPU models.

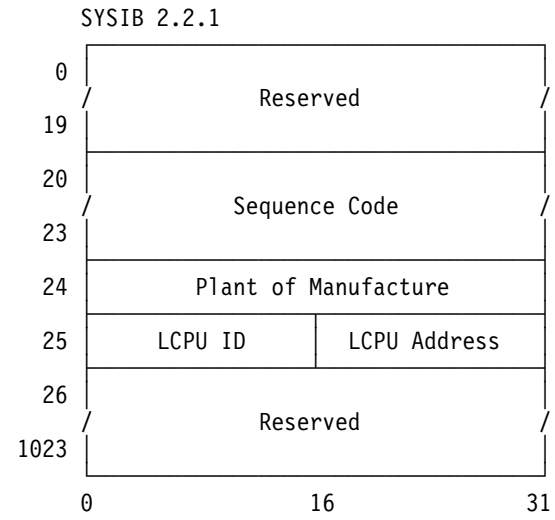
The alternate-capability value applies to each of the CPUs in the configuration. That is, all CPUs in the configuration have the same alternate capability.

***Alternate Multiprocessing CPU-Capability Adjustment Factors:*** Beginning with bytes 0 and 1 of word N+1, the SYSIB contains a series of contiguous two-byte fields, each containing a 16-bit unsigned binary integer that is an adjustment factor (percentage) for the value contained in the alternate-CPU-capability field.

The number of alternate-adjustment-factor fields is one less than the number of CPUs specified in the total-CPU-count field. The alternate-adjustment-factor fields correspond to configurations with increasing numbers of CPUs in the configured state. The first alternate-adjustment-factor field corresponds to a configuration with two CPUs in the configured state. Each successive alternate-adjustment-factor field corresponds to a configuration with a number of CPUs in the configured state that is one more than that for the preceding field.

**SYSIB 2.2.1 (Logical-Partition CPU)**

SYSIB 2.2.1 has the following format:




***Reserved:*** The contents of words 0-19 and 26-63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

***Sequence Code:*** Words 20-23 contain the 16-character (0-9 or uppercase A-Z) EBCDIC sequence code of the configuration. The code is right justified with leading EBCDIC zeros if necessary.

***Plant of Manufacture:*** Word 24 contains the four-character (0-9 or uppercase A-Z) EBCDIC code that identifies the plant of manufacture for the configuration. The code is left justified with trailing blanks if necessary.

***Logical-CPU ID:*** Bytes 0 and 1 of word 25 contain a 16-bit unsigned binary integer that can be used in conjunction with the logical-CPU address to distinguish the logical CPU from the other logical CPUs provided by the same LPAR hypervisor.

***Logical-CPU Address:*** Bytes 2 and 3 of word 25 contain the logical-CPU address by which this logical CPU is identified within the level-2 configuration. The logical-CPU address is a 16-bit unsigned binary integer.



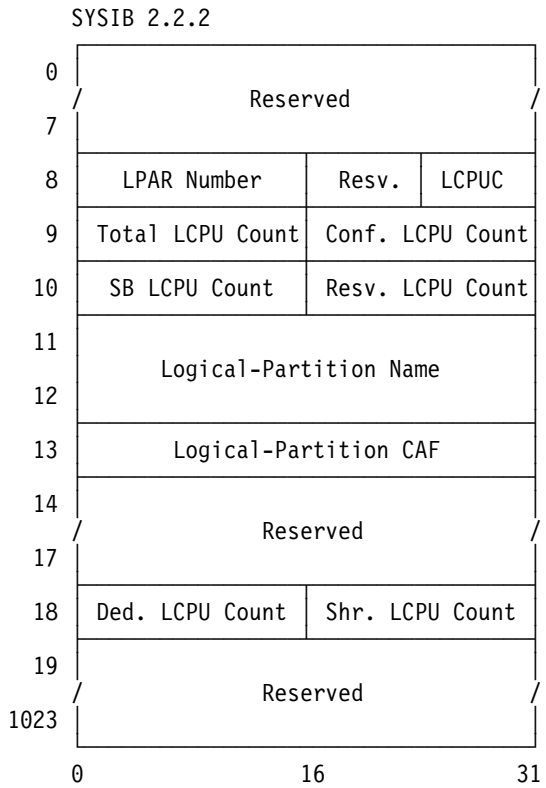
LzLabs GmbH. Ex. 1008-578

The logical-CPU-address field contains the same information as is stored by STORE CPU ADDRESS when the machine is operating in the LPAR mode.

**Programming Note:** Multiple logical CPUs in the same level-2 configuration have the same logical-CPU sequence code, and it is necessary to use other information, such as the logical-CPU address, to establish a unique logical-CPU identity. The sequence code returned for a basic-machine CPU and a logical-partition CPU are identical and have the same value as the sequence code returned for the basic-machine configuration.

**SYSIB 2.2.2 (Logical-Partition CPUs)**

SYSIB 2.2.2 has the following format:




***Reserved:*** The contents of words 0-7, byte 2 of word 8, words 14-17, and words 19-63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

***Logical-Partition Number:*** Bytes 0 and 1 of word 8 contain a 16-bit unsigned binary integer which is the number of the level-2 configuration. This number distinguishes the configuration from all other level-2 configurations provided by the same LPAR hypervisor.

***Logical-CPU Characteristics (LCPUC):*** The contents of byte 3 of word 8 describe the characteristics of the logical CPUs that are provided for the level-2 configuration. The bits and their meanings are as follows:

*Bit* *Meaning*

0 Dedicated: When one, bit 0 indicates that one or more of the logical CPUs for this level-2 configuration are provided using level-1 CPUs that are dedicated to this level-2 configuration and are not used to provide logical CPUs for any other level-2 configuration. The number of logical CPUs that are provided using dedicated level-1 CPUs is specified by the dedicated-LCPU-count value in bytes 0 and 1 of word 18.

When zero, bit 0 indicates that none of the logical CPUs for this level-2 configuration are provided using level-1 CPUs that are dedicated to this level-2 configuration.

1 Shared: When one, bit 1 indicates that one or more of the logical CPUs for this level-2 configuration are provided using level-1 CPUs that can be used to provide logical CPUs for other level-2 configurations. The number of logical CPUs that are provided using shared level-1 CPUs is specified by the shared-LCPU-count value in bytes 2 and 3 of word 18.

When zero, bit 1 indicates that none of the logical CPUs for this level-2 configuration are provided using shared level-1 CPUs.

2 Utilization Limit: When one, bit 2 indicates that the amount of use of the level-1 CPUs that are used to provide the logical CPUs for this level-2 configuration is limited.

When zero, bit 2 indicates that the amount of use of the level-1 CPUs that are used to provide the logical CPUs for this level-2 configuration is unlimited.

3-7 Reserved.

***Total Logical-CPU Count:*** Bytes 0 and 1 of word 9 contain a 16-bit unsigned binary integer that specifies the total number of logical CPUs


LzLabs GmbH. Ex. 1008-579

that are provided for this level-2 configuration. This number includes all of the logical CPUs that are in the configured state, the standby state, or the reserved state.

***Configured Logical-CPU Count:*** Bytes 2 and 3 of word 9 contain a 16-bit unsigned binary integer that specifies the number of logical CPUs for this level-2 configuration that are in the configured state.

A logical CPU is in the configured state when it is in the level-2 configuration and is available to be used to execute programs.

***Standby Logical-CPU Count:*** Bytes 0 and 1 of word 10 contain a 16-bit unsigned binary integer that specifies the number of logical CPUs for this level-2 configuration that are in the standby state.

A logical CPU is in the standby state when it is in the level-2 configuration, is not available to be used to execute programs, and can be made available by issuing instructions to place it in the configured state.

***Reserved Logical-CPU Count:*** Bytes 2 and 3 of word 10 contain a 16-bit unsigned binary integer that specifies the number of CPUs for this level-2 configuration that are in the reserved state.

A logical CPU is in the reserved state when it is in the level-2 configuration, is not available to be used to execute programs, and cannot be made available by issuing instructions to place it in the configured state. (It may be possible to place the reserved CPU in the standby or configured state through manual actions.)

***Logical-Partition Name:*** Words 11-12 contain the 8-character EBCDIC name of this level-2 configuration. The name is left justified with trailing blanks if necessary.

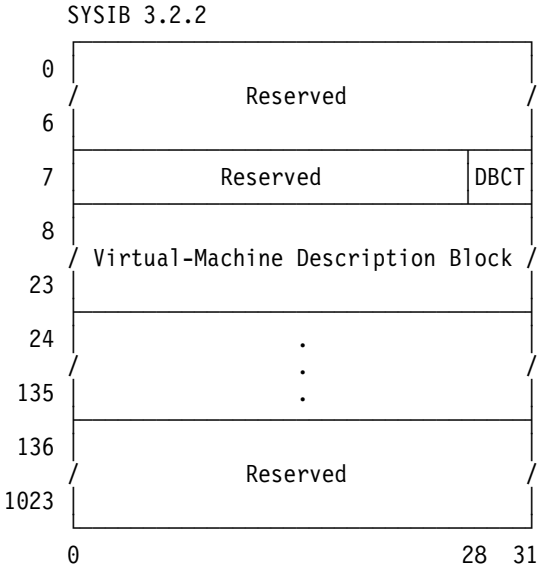
***Logical-Partition Capability Adjustment Factor (CAF):*** Word 13 contains a 32-bit unsigned binary integer, called an adjustment factor, with a value of 1000 or less. The adjustment factor specifies the amount of the underlying level-1-configuration capability that is allowed to be used for this level-2 configuration by the LPAR hypervisor. The fraction of level-1-configuration capability is determined by dividing the CAF value by 1000.

***Dedicated Logical-CPU Count:*** Bytes 0 and 1 of word 18 contain a 16-bit unsigned binary integer that specifies the number of configured-state logical CPUs for this level-2 configuration that are provided using dedicated level-1 CPUs. (See the description of bit 0 of the logical-CPU-characteristics field.)

***Shared Logical-CPU Count:*** Bytes 2 and 3 of word 18 contain a 16-bit unsigned binary integer that specifies the number of configured-state logical CPUs for this level-2 configuration that are provided using shared level-1 CPUs. (See the description of bit 1 of the logical-CPU-characteristics field.)

**SYSIB 3.2.2 (Virtual-Machine CPUs)**

SYSIB 3.2.2 has the following format:




***Reserved:*** The contents of words 0-6, bits 0-27 of word 7, and words 136-1023 are reserved and are stored as zeros.

***Description-Block Count (DBCT):*** Bits 28-31 of word 7 contain a four-bit unsigned binary integer that specifies the number (up to eight) of virtual-machine description blocks that are stored in the SYSIB beginning at word 8.

***Virtual-Machine Description Blocks:*** Words 8-135 contain from one to eight 64-byte virtual-machine description blocks, depending on the number of nested level-3 configurations, if any, and their processing characteristics.



LzLabs GmbH. Ex. 1008-580

When a level-3 configuration is provided by a virtual-machine control program and the control program is being executed by a level-3 configuration provided by another virtual-machine control program, the level-3 configurations are said to be "nested." Level-3 configurations can be nested in this way for several levels.

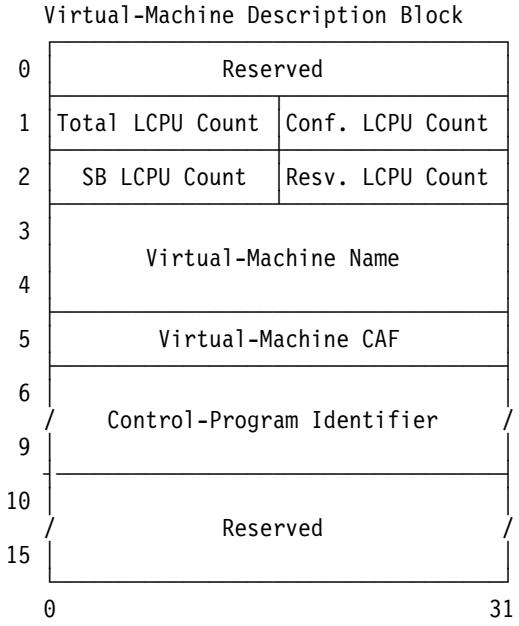
The collection of nested level-3 configurations that is in the path between a program being executed by a level-3 configuration and the basic machine is called a "level-3-configuration stack." The level-3 configurations in a stack are consecutively numbered. The level-3 configuration provided by a virtual-machine control program being executed by either a level-2 configuration or a level-1 configuration is the lowest-numbered (0) level-3 configuration in the stack. The level-3 configuration that is executing the program containing this instruction is the highest numbered (N) level-3 configuration in the stack.

If more than one virtual-machine description block is stored in words 8-135 of the SYSIB, the blocks are stored according to the following rules:

- The collection of level-3 configurations described is a contiguous subset of the total collection of level-3 configurations in the level-3-configuration stack. The subset always includes the highest-numbered level-3 configuration in the stack. One or more level-3 configurations at the bottom of the stack may not be described because STORE SYSTEM INFORMATION is not implemented by the highest of those configurations or the limit of eight description blocks would be exceeded.
- The highest-numbered level-3 configuration in the level-3-configuration stack is always described by the first description block in the SYSIB. The lowest-numbered level-3 configuration in the stack, of those that are included in the subset that is described, is described by the last description block in the SYSIB.

The contents of the SYSIB subsequent to the virtual-machine description blocks and prior to word 136 are reserved and are stored as zeros.

The virtual-machine description block has the following format:




***Reserved:*** The contents of words 0 and 10-15 are reserved and are stored as zeros.

***Total Logical-CPU Count:*** Bytes 0 and 1 of word 1 contain a 16-bit unsigned binary integer that specifies the total number of logical CPUs that are provided for this level-3 configuration. This number includes all of the logical CPUs that are in the configured state, the standby state, and the reserved state.

***Configured Logical-CPU Count:*** Bytes 2 and 3 of word 1 contain a 16-bit unsigned binary integer that specifies the number of logical CPUs for this level-3 configuration that are in the configured state.

A logical CPU is in the configured state when it is in the level-3 configuration and is available to be used to execute programs.

***Standby Logical-CPU Count:*** Bytes 0 and 1 of word 2 contain a 16-bit unsigned binary integer that specifies the number of logical CPUs for this level-3 configuration that are in the standby state.

A logical CPU is in the standby state when it is in the level-3 configuration, is not available to be used to execute programs, and can be made available by issuing instructions to place it in the configured state.


LzLabs GmbH. Ex. 1008-581

***Reserved Logical-CPU Count:*** Bytes 2 and 3 of word 2 contain a 16-bit unsigned binary integer that specifies the number of CPUs for this level-3 configuration that are in the reserved state.

A logical CPU is in the reserved state when it is in the level-3 configuration, is not available to be used to execute programs, and cannot be made available by issuing instructions to place it in the configured state. (It may be possible to place the logical CPU in the standby or configured state through manual actions.)

***Virtual-Machine Name:*** Words 3-4 contain the eight-character EBCDIC name of this level-3 configuration. The name is left justified with trailing blanks if necessary.

***Virtual-Machine Capability Adjustment Factor (CAF):*** Word 5 contains a 32-bit unsigned binary integer, called an adjustment factor, with a value of 1000 or less. The adjustment factor specifies the amount of the underlying level-1-, level-2-, or level-3-configuration capability that is allowed to be used for this level-3 configuration by the virtual-machine control program. The fraction of the underlying capability is determined by dividing the CAF value by 1000.

***Control-Program Identifier:*** Words 6-9 contain the 16-character EBCDIC identifier of the virtual-machine control program that provides this level-3 configuration. This identifier may include qualifiers such as version number and release level. The identifier is left justified with trailing blanks if necessary.

**Special Conditions**

The condition code is set to 3 if the function code in bit positions 0-3 of general register 0 is greater than the current-level number.

Bits 4-23 of general register 0 and 0-15 of general register 1 must be zero; otherwise, a specification exception is recognized.

When the function code is valid and nonzero, the following special conditions apply in an unpredictable order:

- The second operand must be designated on a 4K-byte boundary; otherwise, a specification exception is recognized.
- If the function-code, selector-1, and selector-2 combination is invalid, or if it is valid but the requested information is not available, the condition code is set to 3.

The priority of the recognition of exceptions and condition codes is shown in Figure 10-31 on page 10-103.

***Resulting Condition Code:***

0 Requested configuration-level number placed in general register 0 or requested SYSIB information stored
1 --
2 --
3 Requested SYSIB information not available

***Program Exceptions:***

- Access (store, operand 2, only if function code nonzero)
- Operation (if store-system-information facility not installed)
- Privileged operation
- Specification

**Programming Note:** The storage-operand references for STORE SYSTEM INFORMATION may be multiple-access references. (See “Storage-Operand Consistency” on page 5-87.)



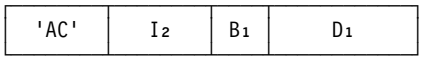
LzLabs GmbH. Ex. 1008-582

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B.1 | Operation exception if the store-system-information facility is not installed. |
| 7.B.2 | Privileged-operation exception for privileged instruction. |
| 8. | Condition code 3 due to function code greater than current-level number. |
| 9. | Specification exception due to bits 4-23 of general register 0 or bits 0-15 of general register 1 not zero. |
| 10. | Condition code 0 due to function code 0. |
| 11.A | Specification exception due to second-operand address not designating a 4K-byte boundary. |
| 11.B | Condition code 3 due to invalid function-code, selector-1, and selector-2 combination or requested information not available. |
| 12. | Access exceptions (store) for system-information block. |
| 13. | Condition code 0 due to information stored in system-information block. |

*Figure 10-31. Priority of Execution: STORE SYSTEM INFORMATION*

## STORE THEN AND SYSTEM MASK

STNSM $D_1(B_1),I_2$ [SI]




Bits 0-7 of the current PSW are stored at the first-operand location. Then the contents of bit positions 0-7 of the current PSW are replaced by the logical AND of their original contents and the second operand.

### Special Conditions

The operation is suppressed on addressing and protection exceptions.

***Condition Code:*** The code remains unchanged.

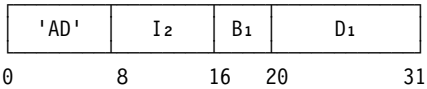
***Program Exceptions:***

- Access (store, operand 1)
- Privileged operation

**Programming Note:** STORE THEN AND SYSTEM MASK permits the program to set selected bits in the system mask to zeros while retaining the original contents for later restoration. For example, it may be necessary that a program, which has no record of the present status, disable program-event recording for a few instructions.

## STORE THEN OR SYSTEM MASK

STOSM $D_1(B_1),I_2$ [SI]




Bits 0-7 of the current PSW are stored at the first-operand location. Then the contents of bit positions 0-7 of the current PSW are replaced by the logical OR of their original contents and the second operand.


LzLabs GmbH. Ex. 1008-583

**Special Conditions**

The value to be loaded into the PSW is not checked for validity before loading. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, if the contents of bit positions 0 and 2-4 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2. The specification exception, which is listed as a program exception for this instruction, is described in "Early Exception Recognition" on page 6-9. It may be considered as occurring early in the process of preparing to execute the following instruction.

The operation is suppressed on addressing and protection exceptions.

***Condition Code:*** The code remains unchanged.

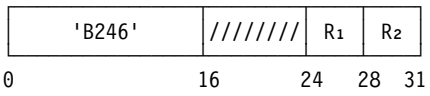
***Program Exceptions:***

- Access (store, operand 1)
- Privileged operation
- Specification

**Programming Note:** STORE THEN OR SYSTEM MASK permits the program to set selected bits in the system mask to ones while retaining the original contents for later restoration. For example, the program may enable the CPU for I/O interruptions without having available the current status of the external-mask bit.

# STORE USING REAL ADDRESS

```
STURA     R1,R2     [RRE]
```




The contents of general register $R_1$ are stored at the real-storage location addressed by the contents of general register $R_2$.

Bits 16-23 of the instruction are ignored.

In the 24-bit addressing mode, bits 8-31 of general register $R_2$ designate a real-storage location on a word boundary, and bits 0-7 of the register are ignored. In the 31-bit addressing mode, bits 1-31 of general register $R_2$ designate a real-storage location on a word boundary, and bit 0 of the register is ignored.

Because it is a real address, the address designating the storage word is not subject to dynamic address translation.

**Special Conditions**

The contents of general register $R_2$ must designate a location on a word boundary; otherwise, a specification exception is recognized.

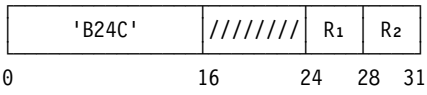
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Addressing (address specified by general register $R_2$)
- Privileged operation
- Protection (store, operand 2, key-controlled protection and low-address protection)
- Specification

# TEST ACCESS

```
TAR       R1,R2     [RRE]
```




The access-list-entry token (ALET) in access register $R_1$ is tested for exceptions recognized during access-register translation (ART). The extended authorization index (EAX) used is bits 0-15 of general register $R_2$. The ALET is also tested for whether it designates the dispatchable-unit access list or the primary-space access list and for whether it is 00000000 or 00000001 hex.

When $R_1$ is 0, the actual contents of access register 0 are used in ART, instead of the 00000000 hex that is usually used.

Bits 16-31 of general register $R_2$ are ignored. Bits 16-23 of the instruction are ignored.

The operation does not depend on the translation mode — bits 5, 16, and 17 of the PSW are ignored.

When the ALET specified by means of the $R_1$ field is other than 00000000 or 00000001 hex, the ART process is applied to the ALET. The EAX speci-



LzLabs GmbH. Ex. 1008-584

fied by means of the $R_2$ field is called the effective EAX, and it is the EAX which is used by ART. When a condition exists that would normally cause one of the exceptions shown in the following table, the instruction is completed by setting condition code 3.

| Exception Name | Cause |
|---|---|
| ALET specification | ALET bits 0-6 not all zeros |
| ALEN translation | Access-list entry (ALE) outside list or invalid (bit 0 is one) |
| ALE sequence | ALE sequence number (ALESN) in ALET not equal to ALESN in ALE |
| ASTE validity | ASN-second-table entry (ASTE) invalid (bit 0 is one) |
| ASTE sequence | ASTE sequence number (ASTESN) in ALE not equal to ASTESN in ASTE |
| Extended authority | ALE private bit not zero, ALE authorization index (ALEAX) not equal to effective EAX, and secondary bit selected by effective EAX either outside authority table or zero |

When ART is completed without one of the above conditions being present, the instruction is completed by setting condition code 1 or 2, depending on whether the effective access list is the dispatchable-unit access list or the primary-space access list, respectively. The effective access list is the dispatchable-unit access list if bit 7 of the ALET is zero, or it is the primary-space access list if bit 7 is one. ART, including the obtaining of the effective access-list designation, is described in "Access-Register-Translation Process" on page 5-48.

When the ALET is 00000000 hex, the instruction is completed by setting condition code 0. When the ALET is 00000001 hex, the instruction is completed by setting condition code 3.

**Special Conditions**

The operation is performed only when the address-space-function control, bit 15 of control register 0, is one. When the address-space-function control is zero, a special-operation exception is recognized.

An addressing exception is recognized when the address used by ART to fetch the effective access-list designation or the ALE, ASTE, or authority-table entry designates a location which is not available in the configuration. When it is necessary to access the authority table — when the private bit in the ALE is not zero and the ALEAX in the ALE is not equal to the effective EAX — an ASN-translation-specification exception may be recognized when bits 30, 31, and 60-63 of the ASTE are not all zeros.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-32 on page 10-106.

***Resulting Condition Code:***

0 Access-list-entry token (ALET) is 00000000 hex
1 ALET designates the dispatchable-unit access list and does not cause exceptions in access-register translation (ART)
2 ALET designates the primary-space access list and does not cause exceptions in ART
3 ALET is 00000001 hex or causes exceptions in ART

***Program Exceptions:***

- Addressing (effective access-list designation, access-list entry, ASN-second-table entry, or authority-table entry)
- ASN-translation specification
- Special operation

**Programming Notes:**

1. TEST ACCESS permits a called program to check whether an ALET passed from the calling program is authorized for use by means of the calling program's EAX. The calling program's EAX can be obtained from the last linkage-stack state entry by means of EXTRACT STACKED STATE. The called program can thus avoid performing an operation for the calling program, through the use of the called program's EAX, which the calling program is not authorized to perform by means of its own EAX.


LzLabs GmbH. Ex. 1008-585

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Special-operation exception due to address-space-function control, bit 15 of control register 0, being zero. |
| 8. | Condition code 0 due to access-list-entry-token (ALET) being 00000000 hex. |
| 9. | Condition code 3 due to ALET being 00000001 hex or ALET bits 0-6 not being all zeros. |
| 10. | Addressing exception for access to effective access-list designation. |
| 11. | Condition code 3 due to access-list entry (ALE) being outside the list. |
| 12. | Addressing exception for access to ALE. |
| 13. | Condition code 3 due to ALE being invalid (bit 0 is one) or access-list-entry sequence number (ALESN) in the ALET not being equal to the ALESN in the ALE. |
| 14. | Addressing exception for access to ASN-second-table entry (ASTE). |
| 15. | Condition code 3 due to ASTE being invalid (bit 0 is one) or ASTE sequence number (ASTESN) in the ALE not being equal to the ASTESN in the ASTE. |
| 16. | ASN-translation-specification exception due to bits 30, 31, and 60-63 of ASTE not being all zeros (only if authority-table access is required). (Optional.) |
| 17. | Condition code 3 due to authority-table entry being outside table. |
| 18. | Addressing exception for access to authority-table entry. |
| 19. | Condition code 3 due to ALE private bit not being zero, ALE authorization index (ALEAX) not being equal to effective extended authorization index (EAX), and secondary bit selected by effective EAX being zero. |
| 20. | Condition code 1 if ALET bit 7 is zero; otherwise, condition code 2. |

*Figure 10-32. Priority of Execution: TEST ACCESS*

2. When an ALET equal to 00000000 hex is passed during a program linkage performed by PROGRAM CALL with space switching (PC-ss), and the ALET conceptually designates the calling program's primary address space and the called program's secondary address space, the ALET must be changed to 00000001 hex before it is used by the called program. Condition code 0 of TEST ACCESS indicates a 00000000 hex ALET so that the ALET can be changed to 00000001 hex by the called program.
3. PROGRAM CALL to current primary (PC-cp) sets the secondary address space equal to the primary address space. PC-ss sets the secondary address space equal to the calling



LzLabs GmbH. Ex. 1008-586

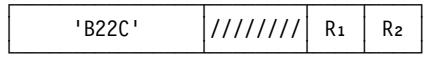
program's primary address space, except that stacking PC-ss sets it equal to the called program's primary address space when the secondary-ASN control in the entry-table entry used is one. In all these cases, a passed 00000001 hex ALET that conceptually designates the calling program's secondary address space is not usable by the called program, even after any transformation (unless the operation was PC-cp and the calling program's PASN and SASN are equal). This is why TEST ACCESS sets condition code 3 when the tested ALET is 00000001 hex.

4. After a PC-ss, a passed ALET that conceptually designates an entry in the primary-space access list of the calling program is not usable by the called program. This is why TEST ACCESS sets condition code 2, instead of condition code 1, when the tested ALET designates the primary-space access list.

5. The control program may manage the ASN-second-table entry in a way that causes a correctable ASTE-validity or ASTE-sequence exception situation to exist; that is, a situation which, if it were to cause a program interruption during access-register translation, would be corrected by the control program so that access-register translation could be completed successfully. In this case, the program should not use TEST ACCESS directly but should instead use a control-program service that uses TEST ACCESS and that corrects the situation, if possible, when condition code 3 is set. MVS/ESA provides the TESTART macro instruction for use instead of the direct use of TEST ACCESS.

## TEST BLOCK

```
TB      R1,R2             [RRE]
```




The storage locations and storage key of a 4K-byte block are tested for usability, and the result of the test is indicated in the condition code. The test for usability is based on the susceptibility of the block to the occurrence of invalid checking-block code. The contents of general register $R_1$ are ignored.

Bits 16-23 of the instruction are ignored.

The block tested is addressed by the contents of general register $R_2$.

A complete testing operation is necessarily performed only when the initial contents of general register 0 are zero. The contents of general register 0 are set to zero at the completion of the operation.

If the block is found to be usable, the 4K bytes of the block are cleared to zeros, the contents of the storage key are unpredictable, and condition code 0 is set. If the block is found to be unusable, the data and the storage key are set, as far as is possible by the model, to a value such that subsequent fetches to the area do not cause a machine-check condition, and condition code 1 is set.

In the 24-bit addressing mode, bits 8-19 of general register $R_2$ designate a 4K-byte block in real storage, and bits 0-7 and 20-31 of the register are ignored. In the 31-bit addressing mode, bits 1-19 of general register $R_2$ designate a 4K-byte block in real storage, and bits 0 and 20-31 of the register are ignored.

The address of the block is a real address, and the accesses to the block designated by the second-operand address are not subject to key-controlled, access-list-controlled, and page protection. Low-address protection does apply. The operation is terminated on addressing and protection exceptions. If termination occurs, the condition code and the contents of general register 0 are unpredictable. The contents of the storage block and its associated storage key are not changed when these exceptions occur.

Depending on the model, the test for usability may be performed (1) by alternately storing and reading out test patterns to the data and storage key in the block or (2) by reference to an internal record of the usability of the blocks which are available in the configuration, or (3) by using a combination of both mechanisms.

In models in which an internal record is used, the block is indicated as unusable if a solid failure has been previously detected, or if intermittent failures in the block have exceeded the threshold implemented by the model. In such models, depending on the criteria, attempts to store may or may not


LzLabs GmbH. Ex. 1008-587

occur. Thus, if block 0 is not usable, and no store occurs, low-address protection may or may not be indicated.

In models in which test patterns are used, TEST BLOCK may be interruptible. When an interruption occurs after a unit of operation, other than the last one, the condition code is unpredictable, and the contents of general register 0 may contain a record of the state of intermediate steps. When execution is resumed after an interruption, the condition code is ignored, but the contents of general register 0 may be used to determine the resumption point.

If (1) TEST BLOCK is executed with an initial value other than zero in general register 0, or (2) the interrupted instruction is resumed after an interruption with a value in general register 0 or a value in the storage block or its associated storage key other than the corresponding value which was present at the time of the interruption, or (3) the block or its associated storage key is accessed by another CPU or by the channel subsystem during the execution of the instruction, then the contents of the storage block, its associated storage key, and general register 0 are unpredictable, along with the resultant condition-code setting.

Invalid checking-block-code errors initially found in the block or encountered during the test do not normally result in machine-check conditions. The test-block function is implemented in such a way that the frequency of machine-check interruptions due to the instruction execution is not significant. However, if, during the execution of TEST BLOCK for an unusable block, that block is accessed by another CPU (or by the channel subsystem), error conditions may be reported both to this CPU and to the other CPU (or to the channel subsystem).

A serialization function is performed before the block is accessed and again after the operation is completed (or partially completed).

The priority of the recognition of exceptions and condition codes is shown in Figure 10-33.

***Resulting Condition Code:***

0 Block usable
1 Block not usable
2 --
3 --

***Program Exceptions:***

- Addressing (fetch and store, operand 2)
- Privileged operation
- Protection (store, operand 2, low-address protection only)

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword. |
| 7.B | Privileged-operation exception. |
| 8. | Addressing exception due to block not being available in the configuration.* |
| 9.A | Condition code 1, block not usable. |
| 9.B | Protection exception due to low-address protection.* |
| 10. | Condition code 0, block usable and set to zeros. |

**Explanation:**

* The operation is terminated on addressing and protection exceptions, and the condition code may be unpredictable.

*Figure 10-33. Priority of Execution: TEST BLOCK*

**Programming Notes:**

1. The execution of TEST BLOCK on most models is significantly slower than that of the MOVE LONG instruction with padding; therefore, the instruction should not be used for the normal case of clearing storage.
2. The program should use TEST BLOCK at initial program loading and as part of the vary-storage-online procedure to determine if blocks of storage exist which should not be used.
3. The program should use TEST BLOCK when an uncorrected error is reported in either the data or storage key of a block. This is because in the execution of TEST BLOCK the attempt is made, as far as is possible on the model, to leave the contents of a block in a state such that subsequent prefetches or unintended references to the block do not cause machine-check conditions. The program may use the resulting condition code in this case to determine if the block can be reused. (The



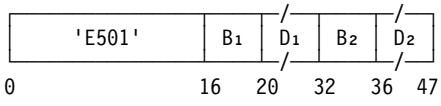
LzLabs GmbH. Ex. 1008-588

block could be indicated as usable if, for example, the error were an externally generated error or an indirect storage error.) This procedure should be followed regardless of whether the indirect-storage-error indication is reported.

4. The model may or may not be successful in removing the errors from a block when TEST BLOCK is executed. The program therefore should take every reasonable precaution to avoid referencing an unusable block. For example, the program should not place the page-frame real address of an unusable block in an attached and valid page-table entry.
5. On some models, machine checks may be reported for a block even though the block is not referenced by the program. When a machine check is reported for a storage-key error in a block which has been marked as unusable by the program, it is possible that SET STORAGE KEY EXTENDED may be more effective than TEST BLOCK in validating the storage key.
6. The storage-operand references for TEST BLOCK may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

## TEST PROTECTION

TPROT $D_1(B_1),D_2(B_2)$ [SSE]




The location designated by the first-operand address is tested for protection exceptions by using the access key specified by bits 24-27 of the second-operand address.

The second-operand address is not used to address data; instead, bits 24-27 of the address form the access key to be used in testing. Bits 0-23 and 28-31 of the second-operand address are ignored.

The first-operand address is a logical address. When the CPU is in the access-register mode (when DAT is on and PSW bits 16 and 17 are 01 binary), the first-operand address is subject to translation by means of both the access-register-translation (ART) and the dynamic-address-translation (DAT) processes. ART applies to the access register designated by the $B_1$ field, and it obtains the segment-table designation to be used by DAT. When DAT is on but the CPU is not in the access-register mode, the first-operand address is subject to translation by DAT. In this case, DAT uses the segment-table designation contained in control register 1, 7, or 13 when the CPU is in the primary-space, secondary-space, or home-space mode, respectively. When DAT is off, the first-operand address is a real address not subject to translation by either ART or DAT.

When the CPU is in the access-register mode and a segment-table designation cannot be obtained by ART because of a situation that would normally cause one of the exceptions shown in the following table, the instruction is completed by setting condition code 3.

| Exception Name | Cause |
|---|---|
| ALET specification | Access-list-entry-token (ALET) bits 0-6 not zeros |
| ALEN translation | Access-list entry (ALE) outside list or invalid (bit 0 is one) |
| ALE sequence | ALE sequence number (ALESN) in ALET not equal to ALESN in ALE |
| ASTE validity | ASN-second-table entry (ASTE) invalid (bit 0 is one) |
| ASTE sequence | ASTE sequence number (ASTESN) in ALE not equal to ASTESN in ASTE |
| Extended authority | ALE private bit not zero, ALE authorization index (ALEAX) not equal to extended authorization index (EAX), and secondary bit selected by EAX either outside authority table or zero |

When the access register contains 00000000 hex or 00000001 hex, ART obtains the segment-table designation from control register 1 or 7, respectively, without accessing the access list. When the $B_1$ field designates access register 0, ART treats the access register as containing 00000000


LzLabs GmbH. Ex. 1008-589

hex and does not examine the actual contents of the access register.

When ART is completed successfully, the operation is continued through the performance of DAT.

When DAT is on and the first-operand address cannot be translated because of a situation that would normally cause a page-translation or segment-translation exception, the instruction is completed by setting condition code 3.

When translation of the first-operand address can be completed, or when DAT is off, the storage key for the block designated by the first-operand address is tested against the access key specified in bit positions 24-27 of the second-operand address, and the condition code is set to indicate whether store and fetch accesses are permitted, taking into consideration all applicable protection mechanisms. Thus, for example, if low-address protection is active and the first-operand effective address is less than 512, then a store access is not permitted. Access-list-controlled protection, page protection, storage-protection override, and fetch-protection override also are taken into account.

The contents of storage, including the change bit, are not affected. Depending on the model, the reference bit for the first-operand address may be set to one, even for the case in which the location is protected against fetching.

**Special Conditions**

When the CPU is in the access-register mode, an addressing exception is recognized when the address used by ART to fetch the effective access-list designation or the ALE, ASTE, or authority-table entry designates a location which is not available in the configuration. When it is necessary to access the authority table — when the private bit in the ALE is not zero and the ALEAX in the ALE is not equal to the EAX — an ASN-translation-specification exception may be recognized when bits 30, 31, and 60-63 of the ASTE are not all zeros.

When DAT is on, an addressing exception is recognized when the address of the segment-table entry or page-table entry or the operand real address after translation designates a location which is not available in the configuration. Also, a translation-specification exception is recognized when the segment-table entry or page-table entry has a format error, that is, when any of the reasons listed in "Translation-Specification Exception" on page 6-34 applies. When DAT is off, only the addressing exception due to the operand real address applies.

For all of the above cases, the operation is suppressed.

***Resulting Condition Code:***

0 Fetching permitted; storing permitted
1 Fetching permitted; storing not permitted
2 Fetching not permitted; storing not permitted
3 Translation not available

***Program Exceptions:***

- Addressing (effective access-list designation, access-list entry, ASN-second-table entry, authority-table entry, segment-table entry, page-table entry, or operand 1)
- ASN-translation specification
- Privileged operation
- Translation specification

**Programming Notes:**

1. TEST PROTECTION permits a program to check the validity of an address passed from a calling program without incurring program exceptions. The instruction sets a condition code to indicate whether fetching or storing is permitted at the location designated by the first-operand address of the instruction. The instruction takes into consideration all of the protection mechanisms in the machine: access-list controlled, page, key-controlled, and low-address protection, storage-protection override, and fetch-protection override. Additionally, since segment-translation and page-translation exception conditions may be a program substitute for a protection violation, these conditions are used to set the condition code rather than cause a program exception.

   When the CPU is in the access-register mode, TEST PROTECTION additionally permits the program to check the usability of an access-list-entry token (ALET) in an access register without incurring program exceptions. The ALET is checked for validity (absence of an ALET-specification, ALEN-translation, and ALE-sequence exception condition) and for



LzLabs GmbH. Ex. 1008-590

being authorized for use by the program (absence of an ASTE-validity, ASTE-sequence, and extended-authority exception condition).

2. See the programming notes under SET PSW KEY FROM ADDRESS for more details and for an alternative approach to testing validity of addresses passed by a calling program. The approach using TEST PROTECTION has the advantage of a test which does not result in interruptions; however, the test and use are separated in time and may not be accurate if the possibility exists that the storage key of the location in question can change between the time it is tested and the time it is used.

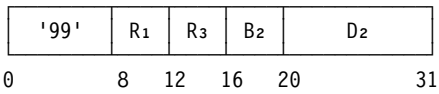
3. In the handling of dynamic address translation, TEST PROTECTION is similar to LOAD REAL ADDRESS in that the instructions do not cause segment-translation and page-translation exceptions. Instead, these exception conditions are indicated by means of a condition-code setting. Similarly, access-register translation sets a condition code for certain exception conditions when performed during either of the two instructions. Conditions which result in condition codes 1, 2, and 3 for LOAD REAL ADDRESS result in condition code 3 for TEST PROTECTION. The instructions also differ in several other respects. The first-operand address of TEST PROTECTION is a logical address and thus is not subject to dynamic address translation when DAT is off. The second-operand address of LOAD REAL ADDRESS is a virtual address which is always translated. TEST PROTECTION may use the TLB for translation of the address, whereas LOAD REAL ADDRESS may use it only if z/Architecture is installed.

   Access-register translation applies to TEST PROTECTION only when the CPU is in the access-register mode (DAT is on), whereas it applies to LOAD REAL ADDRESS when PSW bits 16 and 17 are 01 binary regardless of whether DAT is on or off. When condition code 3 is set because of an exception condition in access-register translation, LOAD REAL ADDRESS, but not TEST PROTECTION, returns in a general register the program-interruption code assigned to the exception. When access-register translation is performed, both TEST PROTECTION and LOAD REAL ADDRESS may use the ART-lookaside buffer (ALB).

   When DAT is off for LOAD REAL ADDRESS, the translation-specification exception for an invalid value of bits 8-12 of control register 0 occurs after instruction fetching as part of the execution portion of the instruction. This exception condition cannot occur for TEST PROTECTION since the operand address is a logical address and does not result in examination of control register 0 when DAT is off. When DAT is on, the exception would be recognized during instruction fetching. Since the instruction-fetching portion of an instruction is common for all instructions, descriptions of access exceptions associated with instruction fetching do not appear in the individual instruction definitions.

## TRACE

TRACE $R_1,R_3,D_2(B_2)$ [RS]




When explicit tracing is on (bit 31 of control register 12 is one), the second operand, which is a 32-bit word in storage, is fetched, and bit 0 of the word is examined. If bit 0 of the second operand is zero, a trace entry is formed at the real-storage location designated by control register 12.

If explicit tracing is off (bit 31 of control register 12 is zero), or if bit 0 of the second operand is one, no trace entry is formed, and no trace exceptions are recognized.

The trace entry is composed of an entry-type identifier, a count of the number of general registers whose contents are placed in the entry, bits 16-63 of the TOD clock, the second operand, and the contents of a range of general registers. The general registers are stored in ascending order of their register numbers, starting with general register $R_1$ and continuing up to and including general register $R_3$, with general register 0 following general register 15. The trace table and the trace-entry formats are described in “Tracing” on page 4-10.


LzLabs GmbH. Ex. 1008-591

When a trace entry is made, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

**Special Conditions**

A privileged-operation exception is recognized in the problem state, even when explicit tracing is off or bit 0 of the second operand is one.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized. It is unpredictable whether the specification exception is recognized when explicit tracing is off.

It is unpredictable whether access exceptions are recognized for the second operand when explicit tracing is off.

***Condition Code:*** The code remains unchanged.

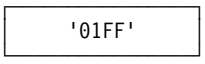
***Program Exceptions:***

- Access (fetch, operand 2)
- Privileged operation
- Specification
- Trace

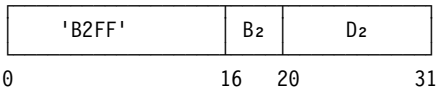
**Programming Note:** Bits 1-15 of the second operand are reserved for model-dependent functions and should therefore be set to zeros.

## TRAP

TRAP2 [E]




TRAP4 $D_2(B_2)$ [S]




A trap operation is performed if the CPU is in the primary-space or access-register mode and the TRAP-enabled bit in byte 47 of the dispatchable-unit control table (DUCT) is one. Otherwise, a special-operation exception is recognized.

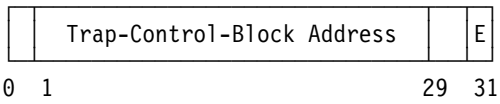
The trap operation obtains a trap-control-block address from the DUCT and then a trap-save-area address and a trap-program address from the trap control block. State information is stored in the trap save area. Then the trap-control-block address is loaded into general register 15. Finally, the current PSW is updated by setting the addressing-mode bit to one (31-bit mode) and the address-space-control bits to zeros (primary-space mode) and by replacing the instruction address with the trap-program address.

For TRAP4, the second-operand address is not used to address data; instead, it is stored in the trap save area.

**Dispatchable-Unit Control Table**

Bytes 44-47 (word 10) of the dispatchable-unit-control table (DUCT) are used by this instruction. The contents of those bytes are as follows:

DUCT Bytes 44-47




The fields in bytes 44-47 of the DUCT are allocated as follows:

***Trap-Control-Block Address:*** Bits 1-28, with three zeros appended on the right, form the 31-bit home virtual address of the trap control block. This address is treated as a 31-bit home virtual address regardless of the current addressing mode and regardless of the current value of the address-space-control bits. This address, with a zero appended on the left, is placed in general register 15 after the contents of that register have been saved in the trap save area.

***TRAP-Enabled Bit (E):*** Bit 31 specifies, when one, that the trap operation is to be performed. TRAP recognizes a special-operation exception if bit 31 is zero.

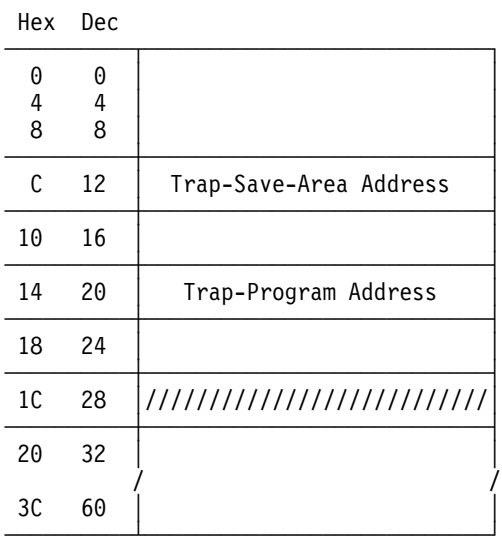
Bits 0, 29, and 30 of bytes 44-47 are ignored, but they should be zeros to permit possible future extensions.



LzLabs GmbH. Ex. 1008-592

### Trap Control Block

The trap control block is 64 bytes aligned on a doubleword boundary. The format of the trap control block is:




The fields in the trap control block are allocated as follows:

***Trap-Save-Area Address:*** Bits 1-28 of bytes 12-15, with three zeros appended on the right, form the 31-bit home virtual address of the trap save area. This address is treated as a 31-bit home virtual address regardless of the current addressing mode and regardless of the current value of the address-space-control bits. Bits 0 and 29-31 of bytes 12-15 are ignored.

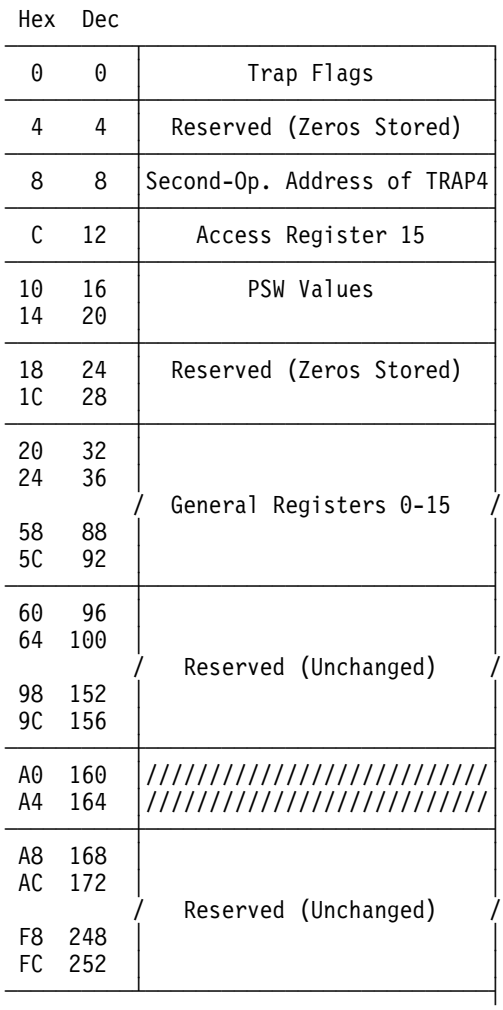
***Trap-Program Address:*** Bits 1-31 of bytes 20-23 form the 31-bit primary virtual address of the trap program. Bit 0 of bytes 20-23 is ignored.

Bytes 0-11, 16-19, 24-27, and 32-63 of the trap control block are reserved and should contain zeros. Bytes 28-31 are available for use by programming.

### Trap Save Area

The trap save area is 256 bytes aligned on a doubleword boundary.

The trap operation stores information into the trap save area as follows:




The fields in the trap save area are allocated as follows:

***Trap Flags:*** Information identifying the instruction(s) causing the trap operation is stored in byte positions 0-3. The detailed format of bytes 0-3 is as follows:

| Flag Bits | Meaning |
|---|---|
| 0 | TRAP was target of EXECUTE |
| 1 | TRAP is TRAP4 (not TRAP2) |
| 2-12 | Reserved, zeros stored |
| 13-14 | Instruction-length code (ILC) |
| 15-31 | Reserved, zeros stored |

Bit 0 of bytes 0-3 is set to one if TRAP was the target of an EXECUTE instruction.

Bit 1 of bytes 0-3 is set to one if TRAP is TRAP4 (not TRAP2).


LzLabs GmbH. Ex. 1008-593

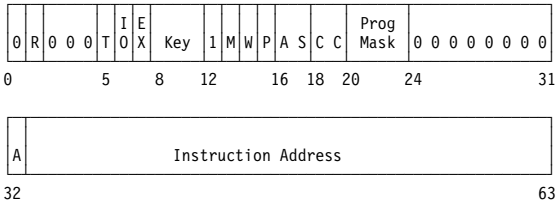
Bits 13 and 14 are the instruction-length code (ILC) that specifies the length of the TRAP instruction, or the length of the EXECUTE instruction if TRAP was the target of EXECUTE.

Bits 2-12 and 15-31 are reserved and are stored as zeros.

***Second-Operand Address of TRAP4:*** For TRAP4, the second-operand address, generated under the control of the current addressing mode and with a zero appended on the left, is stored in byte positions 8-11. For TRAP2, all zeros are stored in those byte positions.

***Access Register 15:*** The contents of access register 15 are stored in byte positions 12-15.

***PSW Values:*** Certain information from the current PSW is stored in byte positions 16-23. The PSW has the following format:




Bits 0-63 of bytes 16-23 correspond one-to-one with bits 0-63 of the PSW. For some bit positions of bytes 16-23, the corresponding PSW bits are stored. For the other bit positions of bytes 16-23, unpredictable values are stored. Information is stored in bytes 16-23 as follows:

| Bits | Value |
|---|---|
| 0 | Zero |
| 1 | Unpredictable |
| 2-4 | Zero |
| 5-11 | Unpredictable |
| 12 | One |
| 13 | Unpredictable |
| 14 | Wait state (W) |
| 15 | Problem state (P) |
| 16-17 | Address-space control (AS) |
| 18-19 | Condition code (CC) |
| 20-23 | Program mask |
| 24-31 | Zero |
| 32 | Addressing mode (A) |
| 33-63 | Instruction address |

In summary, bits 0, 2-4, and 24-31 are zero, bit 12 is one, bits 1, 5-11, and 13 are unpredictable, and the other bits are set with variable information from the PSW.

The wait-state, problem-state, address-space-control, condition-code, program-mask, and addressing-mode values specify the state of the CPU before the TRAP instruction was executed. The instruction-address value is the updated instruction address, which is the address of the instruction following TRAP, or the address of the instruction following EXECUTE if TRAP was the target of EXECUTE.

***General Registers 0-15:*** The contents of general registers 0-15 are stored in byte positions 32-95. They are stored in ascending order of register numbers, starting with register 0 and continuing up to and including register 15.

Bytes 24-31 are reserved, and zeros are stored in these byte positions. Bytes 96-255 remain unchanged. Bytes 96-159 and 168-255 are reserved. Bytes 160-167 are available for use by programming.

**Special Conditions**

The CPU must be in the primary-space mode or access-register mode, and bit 31 in bytes 44-47 of the dispatchable-unit control table must be one; otherwise, a special-operation exception is recognized.

All protection mechanisms apply in the usual way to the accesses to the trap control block and trap save area. Access exceptions may or may not be recognized for sections of the trap control block and trap save area that are not referenced by the TRAP instruction.

The trap-program address in the trap control block is not tested before it replaces the instruction address in the PSW. An odd address will cause a specification exception to be recognized as part of the execution of the next instruction.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-34 on page 10-115.



LzLabs GmbH. Ex. 1008-594

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, trap control block; store, trap save area)
- Addressing (dispatchable-unit control table)
- Operation (if the trap facility is not installed)
- Special operation
- Trace

**Programming Notes:**

1. It is intended that TRAP instructions will overlay instructions in an application program in order to give control to a trap program, which might be a program for performing fix-ups of data used by the application program, such as dates that may be a "Year-2000" problem. TRAP2 can overlay a two-byte instruction, and TRAP4 can overlay a four-byte instructions or the first four bytes of a six-byte instruction. The trap program is to simulate the overlaid instruction and perform fix-ups as appropriate, and it is then to return control to the application program.
2. The trap program can use the RESUME PROGRAM instruction to return control to the application program. For example, the trap program can restore the contents of all registers except access and general registers 15, and then, using those registers (or at least the general register) to address the trap save area, can restore the contents of those registers and also PSW fields from the trap save area.
3. The trap control block and trap save area are in the home address space, and the trap program is in the primary address space. The trap-control-block address placed in general register 15 by TRAP can be useful to the trap program if (1) the primary address space and home address space are the same address space, (2) the trap control block and trap save area are at the same locations in the primary address space as in the home address space, or (3) the trap program can use access registers to access the home address space.
4. The storage-operand references for TRAP may be multiple-access references. (See "Storage-Operand Consistency" on page 5-87.)

| | |
|---|---|
| 1.-6. | Exceptions with the same priority as the priority of program-interruption conditions for the general case. |
| 7.A | Access exceptions for second instruction halfword (TRAP4 only). |
| 7.B.1 | Operation exception if the trap facility is not installed. |
| 7.B.2.A | Special-operation exception due to the CPU not being in the primary-space mode or access-register mode. |
| 7.B.2.B.1 | Addressing exception for access to dispatchable-unit control table. |
| 7.B.2.B.2 | Special-operation exception due to bit 31 in bytes 44-47 of dispatchable-unit control table being zero. |
| 8.A | Trace exceptions. |
| 8.B.1 | Access exceptions (fetch) for trap control block. |
| 8.B.2 | Access exceptions (store) for trap save area. |

*Figure 10-34. Priority of Execution: TRAP*


LzLabs GmbH. Ex. 1008-595

LzLabs GmbH. Ex. 1008-596

# Chapter 11. Machine-Check Handling


Machine-Check Detection . . . . . 11-2
Correction of Machine Malfunctions . . . . . 11-2
Error Checking and Correction . . . . . 11-2
CPU Retry . . . . . 11-2
Effects of CPU Retry . . . . . 11-3
Checkpoint Synchronization . . . . . 11-3
Handling of Machine Checks during Checkpoint Synchronization . . . . . 11-3
Checkpoint-Synchronization Operations 11-3
Checkpoint-Synchronization Action . . . 11-4
Channel-Subsystem Recovery . . . . . 11-4
Unit Deletion . . . . . 11-4
Handling of Machine Checks . . . . . 11-5
Validation . . . . . 11-5
Invalid CBC in Storage . . . . . 11-6
Programmed Validation of Storage . . . 11-7
Invalid CBC in Storage Keys . . . . . 11-7
Invalid CBC in Registers . . . . . 11-10
Check-Stop State . . . . . 11-11
System Check Stop . . . . . 11-11
Machine-Check Interruption . . . . . 11-11
Exigent Conditions . . . . . 11-11
Repressible Conditions . . . . . 11-12
Interruption Action . . . . . 11-12
Point of Interruption . . . . . 11-14
Machine-Check-Interruption Code . . . . . 11-15
Subclass . . . . . 11-16
System Damage . . . . . 11-16
Instruction-Processing Damage . . . . 11-17
System Recovery . . . . . 11-17
Timing-Facility Damage . . . . . 11-17
External Damage . . . . . 11-18
Vector-Facility Failure . . . . . 11-18
Degradation . . . . . 11-18
Warning . . . . . 11-18
Channel Report Pending . . . . . 11-18
Service-Processor Damage . . . . . 11-19
Channel-Subsystem Damage . . . . . 11-19
Subclass Modifiers . . . . . 11-19
Vector-Facility Source . . . . . 11-19
Backed Up . . . . . 11-19
Delayed Access Exception . . . . . 11-19
Ancillary Report . . . . . 11-19
Synchronous Machine-Check-Interruption Conditions 11-20
Processing Backup . . . . . 11-20
Processing Damage . . . . . 11-20
Storage Errors . . . . . 11-20
Storage Error Uncorrected . . . . . 11-21
Storage Error Corrected . . . . . 11-21
Storage-Key Error Uncorrected . . . . 11-21
Storage Degradation . . . . . 11-21
Indirect Storage Error . . . . . 11-21
Machine-Check Interruption-Code Validity Bits . . . . . 11-22
PSW-MWP Validity . . . . . 11-22
PSW Mask and Key Validity . . . . . 11-22
PSW Program-Mask and Condition-Code Validity . . . . . 11-22
PSW-Instruction-Address Validity . . . 11-22
Failing-Storage-Address Validity . . . 11-22
External-Damage-Code Validity . . . . 11-22
Floating-Point-Register Validity . . . . 11-23
General-Register Validity . . . . . 11-23
Control-Register Validity . . . . . 11-23
Storage Logical Validity . . . . . 11-23
Access-Register Validity . . . . . 11-23
Extended-Floating-Point-Register Validity . . . . . 11-23
CPU-Timer Validity . . . . . 11-23
Clock-Comparator Validity . . . . . 11-23
Machine-Check Extended Interruption Information . . . . . 11-24
Register Save Areas . . . . . 11-24
Machine-Check Extended Save Area . . 11-24
External-Damage Code . . . . . 11-24
Failing-Storage Address . . . . . 11-25
Handling of Machine-Check Conditions . . 11-25
Floating Interruption Conditions . . . . . 11-25
Floating Machine-Check-Interruption Conditions . . . . . 11-26
Floating I/O Interruptions . . . . . 11-26
Machine-Check Masking . . . . . 11-26
Channel-Report-Pending Subclass Mask . . . . . 11-26
Recovery Subclass Mask . . . . . 11-26
Degradation Subclass Mask . . . . . 11-26
External-Damage Subclass Mask . . . 11-26
Warning Subclass Mask . . . . . 11-26
Machine-Check Logout . . . . . 11-27
Summary of Machine-Check Masking . . . 11-27


The machine-check-handling mechanism provides extensive equipment-malfunction detection to ensure the integrity of system operation and to permit automatic recovery from some malfunc-

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-597

tions. Equipment malfunctions and certain external disturbances are reported by means of a machine-check interruption to assist in program-damage assessment and recovery. The interruption supplies the program with information about the extent of the damage and the location and nature of the cause. Equipment malfunctions, errors, and other situations which can cause machine-check interruptions are referred to as machine checks.

## Machine-Check Detection

Machine-check-detection mechanisms may take many forms, especially in control functions for arithmetic and logical processing, addressing, sequencing, and execution. For program-addressable information, detection is normally accomplished by encoding redundancy into the information in such a manner that most failures in the retention or transmission of the information result in an invalid code. The encoding normally takes the form of one or more redundant bits, called check bits, appended to a group of data bits. Such a group of data bits and the associated check bits are called a checking block. The size of the checking block depends on the model.

The inclusion of a single check bit in the checking block allows the detection of any single-bit failure within the checking block. In this arrangement, the check bit is sometimes referred to as a "parity bit." In other arrangements, a group of check bits is included to permit detection of multiple errors, to permit error correction, or both.

For checking purposes, the contents of the entire checking block, including the redundancy, are called the checking-block code (CBC). When a CBC completely meets the checking requirements (that is, no failure is detected), it is said to be valid. When both detection and correction are provided and a CBC is not valid but satisfies the checking requirements for correction (the failure is correctable), it is said to be near-valid. When a CBC does not satisfy the checking requirements (the failure is uncorrectable), it is said to be invalid.

## Correction of Machine Malfunctions

Four mechanisms may be used to provide recovery from machine-detected malfunctions: error checking and correction, CPU retry, channel-subsystem recovery, and unit deletion.

Machine failures which are corrected successfully may or may not be reported as machine-check interruptions. If reported, they are system-recovery conditions, which permit the program to note the cause of CPU delay and to keep a log of such incidents.

## Error Checking and Correction

When sufficient redundancy is included in circuitry or in a checking block, failures can be corrected. For example, circuitry can be triplicated, with a voting circuit to determine the correct value by selecting two matching results out of three, thus correcting a single failure. An arrangement for correction of failures of one order and for detection of failures of a higher order is called error checking and correction (ECC). Commonly, ECC allows correction of single-bit failures and detection of double-bit failures.

Depending on the model and the portion of the machine in which ECC is applied, correction may be reported as system recovery, or no report may be given.

Uncorrected errors in storage and in the storage key may be reported, along with a failing-storage address, to indicate where the error occurred. Depending on the situation, these errors may be reported along with system recovery or with the damage or backup condition resulting from the error.

## CPU Retry

In some models, information about some portion of the state of the machine is saved periodically. The point in the processing at which this information is saved is called a checkpoint. The information saved is referred to as the checkpoint information. The action of saving the information is referred to as establishing a checkpoint. The action of discarding previously saved information is called invalidation of the checkpoint information.



LzLabs GmbH. Ex. 1008-598

The length of the interval between establishing checkpoints is model-dependent. Checkpoints may be established at the beginning of each instruction or several times within a single instruction, or checkpoints may be established less frequently.

Subsequently, this saved information may be used to restore the machine to the state that existed at the time when the checkpoint was established. After restoring the appropriate portion of the machine state, processing continues from the checkpoint. The process of restoring to a checkpoint and then continuing is called CPU retry.

CPU retry may be used for machine-check recovery, to effect nullification and suppression of instruction execution when certain program interruptions occur, and in other model-dependent situations.

## Effects of CPU Retry

CPU retry is, in general, performed so that there is no effect on the program. However, change bits which have been changed from zeros to ones are not necessarily set back to zeros. As a result, change bits may appear to be set to ones for blocks which would have been accessed if restoring to the checkpoint had not occurred. If the path taken by the program is dependent on information that may be changed by another CPU or by a channel program or if an interruption occurs, then the final path taken by the program may be different from the earlier path; therefore, change bits may be ones because of stores along a path apparently never taken.

## Checkpoint Synchronization

Checkpoint synchronization consists in the following steps.

1. The CPU operation is delayed until all conceptually previous accesses by this CPU to storage have been completed, both for purposes of machine-check detection and as observed by other CPUs and by channel programs.
2. All previous checkpoints, if any, are canceled.
3. Optionally, a new checkpoint is established.

The CPU operation is delayed until all of these actions appear to be completed, as observed by other CPUs and by channel programs.

## Handling of Machine Checks during Checkpoint Synchronization

When, in the process of completing all previous stores as part of the checkpoint-synchronization action, the machine is unable to complete all stores successfully but can successfully restore the machine to a previous checkpoint, processing backup is reported.

When, in the process of completing all stores as part of the checkpoint-synchronization action, the machine is unable to complete all stores successfully and cannot successfully restore the machine to a previous checkpoint, the type of machine-check-interruption condition reported depends on the origin of the store. Failure to successfully complete stores associated with instruction execution may be reported as instruction-processing damage, or some less critical machine-check-interruption condition may be reported with the storage-logical-validity bit set to zero. A failure to successfully complete stores associated with the execution of an interruption, other than program or supervisor call, is reported as system damage.

When the machine check occurs as part of a checkpoint-synchronization action before the execution of an instruction, the execution of the instruction is nullified. When it occurs before the execution of an interruption, the interruption condition, if the interruption is external, I/O, or restart, is held pending. If the checkpoint-synchronization operation was a machine-check interruption, then along with the originating condition, either the storage-logical-validity bit is set to zero or instruction-processing damage is also reported. Program interruptions, if any, are lost.

## Checkpoint-Synchronization Operations

All interruptions and the execution of certain instructions cause a checkpoint-synchronization action to be performed. The operations which cause a checkpoint-synchronization action are called checkpoint-synchronization operations and include:

- CPU reset
- All interruptions: external, I/O, machine check, program, restart, and supervisor call
- The BRANCH ON CONDITION (BCR) instruction with the $M_1$ and $R_2$ fields containing all ones and all zeros, respectively


LzLabs GmbH. Ex. 1008-599

- The instructions LOAD PSW, SET STORAGE KEY EXTENDED, and SUPERVISOR CALL
- All I/O instructions
- The instructions MOVE TO PRIMARY, MOVE TO SECONDARY, PROGRAM CALL, PROGRAM CALL FAST, PROGRAM TRANSFER, SET ADDRESS SPACE CONTROL, and SET SECONDARY ASN, and PROGRAM RETURN when the state entry to be unstacked is a program-call state entry
- The three trace functions: branch tracing, ASN tracing, and explicit tracing
- PAGE IN and PAGE OUT

**Programming Note:** The instructions which are defined to cause the checkpoint-synchronization action invalidate checkpoint information but do not necessarily establish a new checkpoint. Additionally, the CPU may establish a checkpoint between any two instructions or units of operation, or within a single unit of operation. Thus, the point of interruption for the machine check is not necessarily at an instruction defined to cause a checkpoint-synchronization action.

### Checkpoint-Synchronization Action

For all interruptions except I/O interruptions, a checkpoint-synchronization action is performed at the completion of the interruption. For I/O interruptions, a checkpoint-synchronization action may or may not be performed at the completion of the interruption. For all interruptions except program, supervisor-call, and exigent machine-check interruptions, a checkpoint-synchronization action is also performed before the interruption. The fetch access to the new PSW may be performed either before or after the first checkpoint-synchronization action. The store accesses and the changing of the current PSW associated with the interruption are performed after the first checkpoint-synchronization action and before the second.

For all checkpoint-synchronization instructions except BRANCH ON CONDITION (BCR), I/O instructions, and SUPERVISOR CALL, checkpoint-synchronization actions are performed before and after the execution of the instruction. For BCR, only one checkpoint-synchronization action is necessarily performed, and it may be performed either before or after the instruction address is updated. For SUPERVISOR CALL, a checkpoint-synchronization action is performed before the instruction is executed, including the updating of the instruction address in the PSW. The checkpoint-synchronization action taken after the supervisor-call interruption is considered to be part of the interruption action and not part of the instruction execution. For I/O instructions, a checkpoint-synchronization action is always performed before the instruction is executed and may or may not be performed after the instruction is executed.

The three trace functions — branch tracing, ASN tracing, and explicit tracing — cause checkpoint-synchronization actions to be performed before the trace action and after completion of the trace action.

## Channel-Subsystem Recovery

When errors are detected in the channel subsystem, the channel subsystem attempts to analyze and recover the internal state associated with the various channel-subsystem functions and the state of the channel subsystem and various subchannels. This process, which is called channel-subsystem recovery, may result in a complete recovery or may result in the termination of one or more I/O operations and the clearing of the affected subchannels. Special channel-report-pending machine-check-interruption conditions may be generated to indicate to the program the status of the channel-subsystem recovery.

Malfunctions associated with the I/O operations, depending on the severity of the malfunction, may be reported by means of the I/O-interruption mechanism or by means of the channel-report-pending and channel-subsystem-damage machine-check-interruption conditions.

## Unit Deletion

In some models, malfunctions in certain units of the system can be circumvented by discontinuing the use of the unit. Examples of cases where unit deletion may occur include the disabling of all or a portion of a cache or of a translation-lookaside buffer (TLB). Unit deletion may be reported as a degradation machine-check-interruption condition.



LzLabs GmbH. Ex. 1008-600

## Handling of Machine Checks

A machine check is caused by a machine malfunction and not by data or instructions. This is ensured during the power-on sequence by initializing the machine controls to a valid state and by placing valid CBC in the CPU registers, in the storage keys, and in main storage.

Designation of an unavailable component, such as a storage location, subchannel, or I/O device, does not cause a machine-check indication. Instead, such a condition is indicated by the appropriate program or I/O interruption or condition-code setting. In particular, an attempt to access a storage location which is not in the configuration, or which has power off at the storage unit, results in an addressing exception when detected by the CPU and does not generate a machine-check condition, even though the storage location or its associated storage key has invalid CBC. Similarly, if the channel subsystem attempts to access such a location, an I/O-interruption condition indicating program check is generated rather than a machine-check condition.

A machine check is indicated whenever the result of an operation could be affected by information with invalid CBC or when any other malfunction makes it impossible to establish reliably that an operation can be, or has been, performed correctly. When information with invalid CBC is fetched but not used, the condition may or may not be indicated, and the invalid CBC is preserved.

When a machine malfunction is detected, the action taken depends on the model, the nature of the malfunction, and the situation in which the malfunction occurs. Malfunctions affecting operator-facility actions may result in machine checks or may be indicated to the operator. Malfunctions affecting certain other operations such as SIGNAL PROCESSOR may be indicated by means of a condition code or may result in a machine-check-interruption condition.

A malfunction detected as part of an I/O operation may cause a machine-check-interruption condition, an I/O-error condition, or both. I/O-error conditions are indicated by an I/O interruption or by the appropriate condition-code setting during the execution of an I/O instruction. When the machine reports a failing-storage location detected during an I/O operation, both I/O-error and machine-check conditions may be indicated. The I/O-error condition is the primary indication to the program. The machine-check condition is a secondary indication, which is presented as system recovery together with a failing-storage address.

Certain malfunctions detected as part of I/O instructions and I/O operations are reported by means of special machine-check conditions called I/O machine-check conditions. Thus, malfunctions detected as part of an operation which is I/O related may be reported, depending on the error, in any of three ways: I/O-error condition, I/O machine-check condition, or non-I/O machine-check condition. In some cases, the definition requires the error to be reported by only one of these mechanisms; in other cases, any one, or in some cases, more than one, may be indicated.

**Programming Note:** Although the definition for machine-check conditions is that they are caused by machine malfunctions and not by data and instructions, there are certain unusual situations in which machine-check conditions are caused by events which are not machine malfunctions. Two examples follow:

1. In some cases, the channel-report-pending machine-check-interruption condition indicates a non-error situation. For example, this condition is generated at the completion of the function specified by RESET CHANNEL PATH.
2. Improper use of DIAGNOSE may result in machine-check conditions.

## Validation

Machine errors can be generally classified as solid or intermittent, according to the persistence of the malfunction. A persistent machine error is said to be solid, and one that is not persistent is said to be intermittent. In the case of a register or storage location, a third type of error must be considered, called externally generated. An externally generated error is one where no failure exists in the register or storage location but invalid CBC has been introduced into the location by actions external to the location. For example, the value could be affected by a power transient, or an incorrect value may have been introduced when the information was placed at the location.


LzLabs GmbH. Ex. 1008-601

Invalid CBC is preserved as invalid when information with invalid CBC is fetched or when an attempt is made to update only a portion of the checking block. When an attempt is made to replace the contents of the entire checking block and the block contains invalid CBC, it depends on the operation and the model whether the block remains with invalid CBC or is replaced. An operation which replaces the contents of a checking block with valid CBC, while ignoring the current contents, is called a validation operation. Validation is used to place a valid CBC in a register or at a location which has an intermittent or externally generated error.

Validating a checking block does not ensure that a valid CBC will be observed the next time the checking block is accessed. If the failure is solid, validation is effective only if the information placed in the checking block is such that the failing bits are set to the value to which they fail. If an attempt is made to set the bits to the state opposite to that in which they fail, then the validation will not be effective. Thus, for a solid failure, validation is only useful to eliminate the error condition, even though the underlying failure remains, thereby reducing the exposure to additional reports. The locations, however, cannot be used, since invalid CBC will result from attempts to store other values at the location. For an intermittent failure, however, validation is useful to restore a valid CBC such that a subsequent partial store into the checking block will be permitted. (A partial store is a store into a checking block without replacing the entire checking block.)

When a checking block consists of multiple bytes in storage, or multiple bits in CPU registers, the invalid CBC can be made valid only when all of the bytes or bits are replaced simultaneously.

For each type of field in the system, certain instructions are defined to validate the field. Depending on the model, additional instructions may also perform validation; or, in some models, a register is automatically validated as part of the machine-check-interruption sequence after the original contents of the register are placed in the appropriate save area.

When an error occurs in a checking block, the original information contained in the checking block should be considered lost even after validation. Automatic register validation leaves the contents unpredictable. Programmed and manual validation of checking blocks causes the contents to be changed explicitly.

**Programming Note:** The machine-check-interruption handler must assume that the registers require validation. Thus, each register should be loaded, using an instruction defined to validate, before the register is used or stored.

## Invalid CBC in Storage

The size of the checking block in storage depends on the model but is never more than 4K bytes.

When invalid CBC is detected in storage, a machine-check condition may occur; depending on the circumstances, the machine-check condition may be system damage, instruction-processing damage, or system recovery. If the invalid CBC is detected as part of the execution of a channel program, the error is reported as an I/O-error condition. When a CCW, indirect-data-address word, or data is prefetched from storage, is found to have invalid CBC, but is not used in the channel program, the condition is normally not reported as an I/O-error condition. The condition may or may not be reported as a machine-check-interruption condition. Invalid CBC detected during accesses to storage for other than CPU-related accesses may be reported as system recovery with storage error uncorrected indicated, since the primary error indication is reported by some other means.

When the storage checking block consists of multiple bytes and contains invalid CBC, special storage-validation procedures are generally necessary to restore or place new information in the checking block. Validation of storage is provided with the manual load-clear and system-reset-clear operations and is also provided as a program function. Programmed storage validation is done a block at a time, by executing the privileged instruction TEST BLOCK. Manual storage validation by clear reset validates all blocks which are available in the configuration.

A checking block with invalid CBC is never validated unless the entire contents of the checking block are replaced. An attempt to store into a checking block having invalid CBC, without replacing the entire checking block, leaves the data in the checking block (including the check bits) unchanged. Even when an instruction or a



LzLabs GmbH. Ex. 1008-602

channel-program-input operation specifies that the entire contents of a checking block are to be replaced, validation may or may not occur, depending on the operation and the model.

**Programming Note:** Machine-check conditions may be reported for prefetched and unused data. Depending on the model, such situations may or may not be successfully retried. For example, a BRANCH AND LINK (BALR) instruction which specifies an $R_2$ field of zero will never branch, but on some models a prefetch of the location designated by register 0 may occur. Access exceptions associated with this prefetch will not be reported. However, if an invalid checking-block code is detected, CPU retry may be attempted. Depending on the model, the prefetch may recur as part of the retry, and thus the retry will not be successful. Even when the CPU retry is successful, the performance degradation of such a retry is significant, and system recovery may be presented, normally with a failing-storage address. To avoid continued degradation, the program should initiate proceedings to eliminate use of the location and to validate the location.

### Programmed Validation of Storage

Provided that an invalid CBC does not exist in the storage key associated with a 4K-byte block, the instruction TEST BLOCK causes the entire 4K-byte block to be set to zeros with a valid CBC, regardless of the current contents of the storage. TEST BLOCK thus removes an invalid CBC from a location in storage which has an intermittent, or one-time, failure. However, if a permanent failure exists in a portion of the storage, a subsequent fetch may find an invalid CBC.

## Invalid CBC in Storage Keys

Depending on the model, each storage key may be contained in a single checking block, or the access-control and fetch-protection bits and the reference and change bits may be in separate checking blocks.

Figure 11-1 on page 11-8 describes the action taken when the storage key has invalid CBC. The figure indicates the action taken for the case when the access-control and fetch-protection bits are in one checking block and the reference and change bits are in a separate checking block. In machines where both fields are included in a single checking block, the action taken is the combination of the actions for each field in error, except that completion is permitted only if an error in all affected fields permits completion. References to main storage to which key-controlled protection does not apply are treated as if an access key of zero is used for the reference. This includes such references as channel-program references during initial program loading and implicit references, such as interruption action and DAT-table accesses.


LzLabs GmbH. Ex. 1008-603

| Type of Reference | Action Taken on Invalid CBC | |
|---|---|---|
| | For Access-Control and Fetch-Protection Bits | For Reference and Change Bits |
| SET STORAGE KEY EXTENDED | Complete; validate. | Complete; validate. |
| INSERT STORAGE KEY EXTENDED | PD; preserve. | PD; preserve. |
| RESET REFERENCE BIT EXTENDED | PD or complete; preserve. | PD; preserve. |
| INSERT VIRTUAL STORAGE KEY or TEST PROTECTION | PD; preserve. | CPF; preserve. |
| CPU prefetch (information not used) | CPF; preserve. | CPF; preserve. |
| Channel-program prefetch (information not used) | IPF; preserve. | IPF; preserve. |
| Fetch, nonzero access key | MC; preserve. | MC or complete; preserve. |
| Store[1], nonzero access key | MC[2]; preserve. | MC and preserve; or complete[3] and correct. |
| Fetch, zero access key[4] | MC or complete; preserve. | MC or complete; preserve. |
| Store[1], zero access key[2] | MC or complete; preserve. | MC and preserve; or complete[3] and correct. |

**Explanation:**

[1] CPU virtual- and logical-address store accesses are subject to page protection. When the page-protection bit is one, the location will not be changed; however, the machine may indicate a machine-check condition if the storage key or the data itself has invalid CBC.

[2] The contents of the main-storage location are not changed.

[3] The contents of the reference and change bits are set to ones if the "complete" action is taken.

[4] The action shown for an access key of zero is also applicable to references to which key-controlled protection does not apply.

*Figure 11-1 (Part 1 of 2). Invalid CBC in Storage Keys*



LzLabs GmbH. Ex. 1008-604

**Explanation** (Continued):

Complete The condition does not cause termination of the execution of the instruction, and, unless an unrelated condition prohibits it, the execution of the instruction is completed, ignoring the error condition. No machine-check-damage conditions are reported, but system recovery may be reported.

Correct The reference and change bits are set to ones with valid CBC.

Preserve The contents of the entire checking block having invalid CBC are left unchanged.

Validate The entire key is set to the new value with valid CBC.

CPF Invalid CBC in the storage key for a CPU prefetch which is unused, or for instructions which do not examine the reference and change bits, may result in any of the following situations:

- The operation is completed; no machine-check condition is reported.
- The operation is completed; system recovery, with storage-key error uncorrected, is reported.
- Instruction-processing damage, with or without backup and with storage-key error uncorrected, is reported.

IPF Invalid CBC in the storage key for a channel-program prefetch which is unused may result in any of the following:

- The I/O operation is completed; no machine-check condition is reported.
- The I/O operation is completed; system recovery, with storage-key error uncorrected, is reported.

MC Same as PD for CPU references, but a channel-subsystem reference may result in the following combinations of I/O-error conditions and machine-check conditions:

- An I/O-error condition is reported; no machine-check condition is reported.
- An I/O-error condition is reported; system recovery, with or without storage-key error uncorrected, is reported.

PD Instruction-processing damage, with or without backup and with or without storage-key error uncorrected, is reported.

**Note:** When storage-key error uncorrected is reported, a failing-storage address may or may not also be reported.

*Figure 11-1 (Part 2 of 2). Invalid CBC in Storage Keys*


LzLabs GmbH. Ex. 1008-605

## Invalid CBC in Registers

When invalid CBC is detected in a CPU register, a machine-check condition may be recognized. CPU registers include the general, floating-point, floating-point-control, access, and control registers, the current PSW, the prefix register, the TOD clock, the CPU timer, and the clock comparator.

When a machine-check interruption occurs, whether or not it is due to invalid CBC in a CPU register, the following actions affecting the CPU registers, other than the prefix register and the TOD-clock, are taken as part of the interruption.

1. The contents of the registers are saved in assigned storage locations. Any register which is in error is identified by a corresponding validity bit of zero in the machine-check-interruption code. Malfunctions detected during register saving do not result in additional machine-check-interruption conditions; instead, the correctness of all the information stored is indicated by the appropriate setting of the validity bits.
2. On some models, registers with invalid CBC are then validated, their actual contents being unpredictable. On other models, programmed validation is required.

The prefix register and the TOD clock are not stored during a machine-check interruption, have no corresponding validity bit, and are not validated.

On those models in which registers are not automatically validated as part of the machine-check interruption, a register with invalid CBC will not cause a machine-check-interruption condition unless the contents of the register are actually used. In these models, each register may consist of one or more checking blocks, but multiple registers are not included in a single checking block. When only a portion of a register is accessed, invalid CBC in the unused portion of the same register may cause a machine-check-interruption condition. For example, invalid CBC in the right half of a floating-point register may cause a machine-check-interruption condition if a LOAD (LE) operation attempts to replace the left half, or short form, of the register.

Invalid CBC associated with the prefix register cannot safely be reported by the machine-check interruption, since the interruption itself requires that the prefix value be applied to convert real addresses to the corresponding absolute addresses. Invalid CBC in the prefix register causes the CPU to enter the check-stop state immediately.

On those models which do not validate registers during a machine-check interruption, the following instructions will cause validation of a register, provided the information in the register is not used before the register is validated. Other instructions, although they replace the entire contents of a register, do not necessarily cause validation.

General registers are validated by BRANCH AND LINK (BAL, BALR), BRANCH AND SAVE (BAS, BASR), LOAD (LR), and LOAD ADDRESS. LOAD (L) and LOAD MULTIPLE validate if the operand is on a word boundary, and LOAD HALFWORD validates if the operand is on a halfword boundary.

Floating-point registers are validated by LOAD (LDR) and, if the operand is on a doubleword boundary, by LOAD (LD).

The floating-point-control register is validated by LOAD FLOATING POINT CONTROL REGISTER.

Access registers are validated by LOAD ACCESS MULTIPLE. Only the even-odd access-register pairs that are included in the set of access registers specified for LOAD ACCESS MULTIPLE are validated. Thus, when a single access register is specified, or when a pair of access registers starting with an odd-numbered register is specified, no register is validated.

Control registers may be validated either singly or in groups by using the instruction LOAD CONTROL.

The TOD programmable register, CPU timer, clock comparator, and prefix register are validated by SET CLOCK PROGRAMMABLE REGISTER, SET CPU TIMER, SET CLOCK COMPARATOR, and SET PREFIX, respectively.

The TOD clock is validated by a SET CLOCK instruction that sets the clock.

**Programming Note:** Depending on the register and the model, the contents of a register may be validated by the machine-check interruption, or the



LzLabs GmbH. Ex. 1008-606

model may require that a program execute a validating instruction after the machine-check interruption has occurred. In the case of the CPU timer, depending on the model, both the machine-check interruption and validating instructions may be required to restore the CPU timer to full working order.

## Check-Stop State

In certain situations, it is impossible or undesirable to continue operation when a machine error occurs. In these cases, the CPU may enter the check-stop state, which is indicated by the check-stop indicator.

In general, the CPU may enter the check-stop state whenever an uncorrectable error or other malfunction occurs and the machine is unable to recognize a specific machine-check-interruption condition.

The CPU always enters the check-stop state if any of the following conditions exists:

- PSW bit 13 is zero, and an exigent machine-check condition is generated.
- During the execution of an interruption due to one exigent machine-check condition, another exigent machine-check condition is detected.
- During a machine-check interruption, the machine-check-interruption code cannot be stored successfully, or the new PSW cannot be fetched successfully.
- Invalid CBC is detected in the prefix register.
- A malfunction in the receiving CPU, which is detected after accepting the order, prevents the successful completion of a SIGNAL PROCESSOR order and the order was a reset, or the receiving CPU cannot determine what the order was. The receiving CPU enters the check-stop state.

There may be many other conditions for particular models when an error may cause check stop.

When the CPU is in the check-stop state, instructions and interruptions are not executed. The TOD clock is normally not affected by the check-stop state. The CPU timer may or may not run in the check-stop state, depending on the error and the model. The start key and stop key are not effective in this state.

The CPU may be removed from the check-stop state by CPU reset.

In a multiprocessing configuration, a CPU entering the check-stop state generates a request for a malfunction-alert external interruption to all CPUs in the configuration. Except for the reception of a malfunction alert, other CPUs and the I/O system are normally unaffected by the check-stop state in a CPU. However, depending on the nature of the condition causing the check stop, other CPUs may also be delayed or stopped, and channel subsystem and I/O activity may be affected.

### System Check Stop

In a multiprocessing configuration, some errors, malfunctions, and damage conditions are of such severity that the condition causes all CPUs in the configuration to enter the check-stop state. This condition is called a system check stop. The state of the channel subsystem and I/O activity is unpredictable.

## Machine-Check Interruption

A request for a machine-check interruption, which is made pending as the result of a machine check, is called a machine-check-interruption condition. There are two types of machine-check-interruption conditions: exigent conditions and repressible conditions.

## Exigent Conditions

Exigent machine-check-interruption conditions are those in which damage has or would have occurred such that execution of the current instruction or interruption sequence cannot safely continue. Exigent conditions include two subclasses: instruction-processing damage and system damage. In addition to indicating specific exigent conditions, system damage is used to report any malfunction or error which cannot be isolated to a less severe report.

Exigent conditions for instruction sequences can be either nullifying exigent conditions or terminating exigent conditions, according to whether the instructions affected are nullified or terminated. Exigent conditions for interruption sequences are terminating exigent conditions. The terms “nullification” and “termination” have the same


LzLabs GmbH. Ex. 1008-607

meanings as that used in Chapter 5, "Program Execution," except that more than one instruction may be involved. Thus, a nullifying exigent condition indicates that the CPU has returned to the beginning of a unit of operation prior to the error. A terminating exigent condition means that the results of one or more instructions may have unpredictable values.

## Repressible Conditions

Repressible machine-check-interruption conditions are those in which the results of the instruction-processing sequence have not been affected. Repressible conditions can be delayed, until the completion of the current instruction or even longer, without affecting the integrity of CPU operation. Repressible conditions are of three groups: recovery, alert, and repressible damage. Each group includes one or more subclasses.

A malfunction in the CPU, storage, or operator facilities which has been successfully corrected or circumvented internally without logical damage is called a recovery condition. Depending on the model and the type of malfunction, some or all recovery conditions may be discarded and not reported. Recovery conditions that are reported are grouped in one subclass, system recovery.

A machine-check-interruption condition not directly related to a machine malfunction is called an alert condition. The alert conditions are grouped in two subclasses: degradation and warning.

A malfunction resulting in an incorrect state of a portion of the system not directly affecting sequential CPU operation is called a repressible-damage condition. Repressible-damage conditions are grouped in six subclasses, according to the function affected: timing-facility damage, external damage, channel report pending, channel-subsystem damage, service-processor damage, and vector-facility failure.

**Programming Notes:**

1. Even though repressible conditions are usually reported only at normal points of interruption, they may also be reported with exigent machine-check conditions. Thus, if an exigent machine-check condition causes an instruction to be abnormally terminated and a machine-check interruption occurs to report the exigent condition, any pending repressible conditions may also be reported. The meaningfulness of the validity bits depends on what exigent condition is reported.
2. Classification of damage as either exigent or repressible does not imply the severity of the damage. The distinction is whether action must be taken as soon as the damage is detected (exigent) or whether the CPU can continue processing (repressible). For a repressible condition, the current instruction can be completed before taking the machine-check interruption if the CPU is enabled for machine checks; if the CPU is disabled for machine checks, the condition can safely be kept pending until the CPU is again enabled for machine checks.

   For example, the CPU may be disabled for machine-check interruptions because it is handling an earlier instruction-processing-damage interruption. If, during that time, an I/O operation encounters a storage error, that condition can be kept pending because it is not expected to interfere with the current machine-check processing. If, however, the CPU also makes a reference to the area of storage containing the error before re-enabling machine-check interruptions, another instruction-processing-damage condition is created, which is treated as an exigent condition and causes the CPU to enter the check-stop state.
3. A repressible condition may be a floating condition. A floating repressible condition is eligible to cause an interruption on any CPU in the configuration. At the point when a CPU performs an interruption for a floating repressible condition, the condition is no longer eligible to cause an interruption on the remaining CPUs in the configuration.

## Interruption Action

A machine-check interruption causes the following actions to be taken.

If z/Architecture is installed, an architectural-mode identification with an all zeros value is stored at real location 163. The following occurs regardless of whether z/Architecture is installed.

The PSW reflecting the point of interruption is stored as the machine-check old PSW at real



LzLabs GmbH. Ex. 1008-608

location 48. The contents of other registers are stored in register save areas at real locations 216-231 and 288-511, and in a machine-check extended save area designated by the contents of real locations 212-215. After the contents of the registers are stored in register save areas and the extended save area, depending on the model, the registers may be validated with the contents being unpredictable. A failing-storage address may be stored at real locations 248-251, and an external-damage code may be stored at real locations 244-247. A machine-check-interruption code (MCIC) of eight bytes is stored at real locations 232-239. The new PSW is fetched from real locations 112-119. In addition, a machine-check logout may have occurred.

The machine-generated addresses used to access the old and new PSW, the MCIC, extended interruption information, the locations containing the extended-save-area address, and the fixed-logout area are all real addresses. The extended-save-area address is an absolute address.

The fields in assigned storage locations that are accessed during the machine-check interruption are summarized in Figure 11-2.

| Information Stored (Fetched) | Starting Location* | Length in Bytes |
|---|---|---|
| Old PSW | 48 | 8 |
| Architectural-mode identification¹ | 163 | 1 |
| New PSW (fetched) | 112 | 8 |
| Extended-save-area address (fetched) | 212 | 4 |
| Machine-check-interruption code | 232 | 8 |
| Register save areas | | |
| CPU timer | 216 | 8 |
| Clock comparator | 224 | 8 |
| Access registers 0-15 | 288 | 64 |
| Floating-point registers 0, 2, 4, 6 | 352 | 32 |
| General registers 0-15 | 384 | 64 |
| Control registers 0-15 | 448 | 64 |
| Extended interruption information | | |
| External-damage code | 244 | 4 |
| Failing-storage address | 248 | 4 |
| Fixed-logout area | 256 | 16 |

**Explanation:**

* All locations are in real storage.
¹ Stored only if z/Architecture is installed.

*Figure 11-2. Machine-Check-Interruption Locations*

When the basic-floating-point-extensions facility is installed, the extended-save-area control, bit 2 of control register 14, is one, and bits 1-19 of the word at real locations 212-215 are not all zeros, then other fields are stored in the machine-check extended save area. Figure 11-3 lists the fields that are stored, their offsets within the area, and their lengths. Bytes 144-4095 of the extended save area remain unchanged.

| Field | Byte Offset | Length in Bytes |
|---|---|---|
| Fl-pt registers 0-15 | 0 | 128 |
| Fl-pt-control register | 128 | 4 |
| Reserved (zeros stored) | 132 | 12 |

*Figure 11-3. Machine-Check Extended-Save-Area Locations*

The address of the machine-check extended save area is formed by appending 12 zeros to the right of bits 1-19 of the word at real locations 212-215. This address is treated as a 31-bit absolute address. If the 4096-byte block of storage at the address is not available in the configuration, storing into the extended save area is not performed.

If the machine-check-interruption code cannot be stored successfully or the new PSW cannot be fetched successfully, the CPU enters the check-stop state.

A repressible machine-check condition can initiate a machine-check interruption only if both PSW bit 13 is one and the associated subclass mask bit, if any, in control register 14 is also one. When it occurs, the interruption does not terminate the execution of the current instruction; the interruption is taken at a normal point of interruption, and no program or supervisor-call interruptions are eliminated. If the machine check occurs during the execution of a machine function, such as a CPU-timer update, the machine-check interruption takes place after the machine function has been completed.

When the CPU is disabled for a particular repressible machine-check condition, the condition remains pending. Depending on the model and the condition, multiple repressible conditions may be held pending for a particular subclass, or only one condition may be held pending for a particular subclass, regardless of the number of conditions that may have been detected for that subclass.

When a repressible machine-check interruption occurs because the interruption condition is in a subclass for which the CPU is enabled, pending conditions in other subclasses may also be indicated by the same interruption code, even though


LzLabs GmbH. Ex. 1008-609

the CPU is disabled for those subclasses. All indicated conditions are then cleared.

If a machine check which is to be reported as a system-recovery condition is detected during the execution of the interruption procedure due to a previous machine-check condition, the system-recovery condition may be combined with the other conditions, discarded, or held pending.

An exigent machine-check condition can cause a machine-check interruption only when PSW bit 13 is one. When a nullifying exigent condition causes a machine-check interruption, the interruption is taken at a normal point of interruption. When a terminating exigent condition causes a machine-check interruption, the interruption terminates the execution of the current instruction and may eliminate the program and supervisor-call interruptions, if any, that would have occurred if execution had continued. Proper execution of the interruption sequence, including the storing of the old PSW and other information, depends on the nature of the malfunction. When an exigent machine-check condition occurs during the execution of a machine function, such as a CPU-timer update, the sequence is not necessarily completed.

If, during the execution of an interruption due to one exigent machine-check condition, another exigent machine check is detected, the CPU enters the check-stop state. If an exigent machine check is detected during an interruption due to a repressible machine-check condition, system damage is reported.

When PSW bit 13 is zero, an exigent machine-check condition causes the CPU to enter the check-stop state.

Machine-check-interruption conditions are handled in the same manner regardless of whether the wait-state bit in the PSW is one or zero: a machine-check condition causes an interruption if the CPU is enabled for that condition.

Machine checks which occur while the rate control is set to the instruction-step position are handled in the same manner as when the control is set to the process position; that is, recovery mechanisms are active, and machine-check interruptions occur when allowed. Machine checks occurring during a manual operation may be indicated to the operator, may generate a system-recovery condition, may result in system damage, or may cause a check stop, depending on the model.

Every reasonable attempt is made to limit the side effects of any machine check and the associated interruption. Normally, interruptions, as well as the progress of I/O operations, remain unaffected. The malfunction, however, may affect these activities, and, if the currently active PSW has bit 13 set to one, the machine-check interruption will indicate the total extent of the damage caused, and not just the damage which originated the condition.

## Point of Interruption

The point in the processing which is indicated by the interruption and used as a reference point by the machine to determine and indicate the validity of the status stored is referred to as the point of interruption.

Because of the checkpoint capability in models with CPU retry, the interruption resulting from an exigent machine-check-interruption condition may indicate a point in the CPU processing sequence which is logically prior to the error. Additionally, the model may have some choice as to which point in the CPU processing sequence the interruption is indicated, and, in some cases, the status which can be indicated as valid depends on the point chosen.

Only certain points in the processing may be used as a point of interruption. For repressible machine-check interruptions, the point of interruption must be after one unit of operation is completed and any associated program or supervisor-call interruption is taken, and before the next unit of operation is begun.

Exigent machine-check conditions for instruction sequences are those in which damage has or would have occurred to the instruction stream. Thus, the damage can normally be associated with a point part way though an instruction, and this point is called the point of damage. In some cases, there may be one or more instructions separating the point of damage and the point of interruption, and the processing associated with one or more instructions may be damaged. When the point of interruption is a point prior to the point of damage due to a nullifiable exigent machine-check condition, the point of interruption can be only at



LzLabs GmbH. Ex. 1008-610

the same points as for repressible machine-check conditions.

In addition to the point of interruption permitted for repressible machine-check conditions, the point of interruption for a terminating exigent machine-check condition may also be after the unit of operation is completed but before any associated program or supervisor-call interruption occurs. In this case, a valid PSW instruction address is defined as that which would have been stored in the old PSW for the program or supervisor-call interruption. Since the operation has been terminated, the values in the result fields, other than the instruction address, are unpredictable. Thus, the validity bits associated with fields which are due to be changed by the instruction stream are meaningless when a terminating exigent machine-check condition is reported.

When the point of interruption and the point of damage due to an exigent machine-check condition are separated by a checkpoint-synchronization function, the damage has not been isolated to a particular program, and system damage is indicated.

When an exigent machine-check-interruption condition occurs, the point of interruption which is chosen affects the amount of damage which must be indicated. An attempt is made, when possible, to choose a point of interruption which permits the minimum indication of damage. In general, the preference is the interruption point immediately preceding the error.

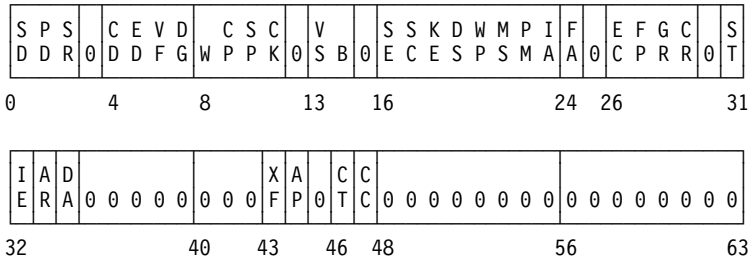
When all the status information stored as a result of an exigent machine-check-interruption condition does not reflect the same point, an attempt is made, when possible, to choose the point of interruption so that the instruction address which is stored in the machine-check old PSW is valid.

## Machine-Check-Interruption Code

On all machine-check interruptions, a machine-check-interruption code (MCIC) is stored in the doubleword starting at real location 232. The code has the format shown in Figure 11-4 on page 11-16.

Bits in the MCIC which are not assigned or not implemented by a particular model, are stored as zeros.


LzLabs GmbH. Ex. 1008-611




| Bits | Name |
|---|---|
| 0 | System damage (SD) |
| 1 | Instruction-processing damage (PD) |
| 2 | System recovery (SR) |
| 4 | Timing-facility damage (CD) |
| 5 | External damage (ED) |
| 6 | Vector-facility failure (VF) |
| 7 | Degradation (DG) |
| 8 | Warning (W) |
| 9 | Channel report pending (CP) |
| 10 | Service-processor damage (SP) |
| 11 | Channel-subsystem damage (CK) |
| 13 | Vector-facility source (VS) |
| 14 | Backed up (B) |
| 16 | Storage error uncorrected (SE) |
| 17 | Storage error corrected (SC) |
| 18 | Storage-key error uncorrected (KE) |
| 19 | Storage degradation (DS) |
| 20 | PSW-MWP validity (WP) |
| 21 | PSW mask and key validity (MS) |
| 22 | PSW program-mask and condition-code validity (PM) |
| 23 | PSW-instruction-address validity (IA) |
| 24 | Failing-storage-address validity (FA) |
| 26 | External-damage-code validity (EC) |
| 27 | Floating-point-register validity (FP) |
| 28 | General-register validity (GR) |
| 29 | Control-register validity (CR) |
| 31 | Storage logical validity (ST) |
| 32 | Indirect storage error (IE) |
| 33 | Access-register validity (AR) |
| 34 | Delayed-access exception (DA) |
| 43 | Extended-floating-point-register validity (XF) |
| 44 | Ancillary report (AP) |
| 46 | CPU-timer validity (CT) |
| 47 | Clock-comparator validity (CC) |

Note: All other bits of the MCIC are unassigned and stored as zeros.

*Figure 11-4. Machine-Check Interruption-Code Format*

## Subclass

Bits 0-2 and 4-11 are the subclass bits which identify the type of machine-check condition causing the interruption. At least one of the subclass bits is stored as a one. When multiple errors have occurred, several subclass bits may be set to ones.

### System Damage

Bit 0 (SD), when one, indicates that damage has occurred which cannot be isolated to one or more of the less severe machine-check subclasses. When system damage is indicated, the ancillary-report bit, bit 44, is meaningful, the remaining bits in the machine-check-interruption code are not meaningful, and information stored in the register save areas and machine-check extended-interruption fields is not meaningful.

System damage is a terminating exigent condition and has no subclass-mask bit.



LzLabs GmbH. Ex. 1008-612

## Instruction-Processing Damage

Bit 1 (PD), when one, indicates that damage has occurred to the instruction processing of the CPU.

The exact meaning of bit 1 depends on the setting of the backed-up bit, bit 14. When the backed-up bit is one, the condition is called processing backup. When the backed-up bit is zero, the condition is called processing damage. These two conditions are described in "Synchronous Machine-Check-Interruption Conditions" on page 11-20.

Instruction-processing damage can be a nullifying or a terminating exigent condition and has no subclass-mask bit.

## System Recovery

Bit 2 (SR), when one, indicates that malfunctions were detected but did not result in damage or have been successfully corrected. Some malfunctions detected as part of an I/O operation may result in a system-recovery condition in addition to an I/O-error condition. The presence and extent of the system-recovery capability depend on the model.

System recovery is a repressible condition. It is masked by the recovery subclass-mask bit, which is in bit position 4 of control register 14.

**Programming Notes:**

1. System recovery may be used to report a failing-storage address detected by a CPU prefetch or by an I/O operation.
2. Unless the corresponding validity bits are ones, the indication of system recovery does not imply storage logical validity or that the fields stored as a result of the machine-check interruption are valid.

## Timing-Facility Damage

Bit 4 (CD), when one, indicates that damage has occurred to the TOD clock, CPU timer, clock comparator, or TOD programmable register, or to the CPU-timer or clock-comparator external-interruption conditions. The timing-facility-damage machine-check condition is set whenever any of the following occurs:

1. The TOD clock accessed by this CPU enters the error or not-operational state.
2. The CPU timer is damaged, and the CPU is enabled for CPU-timer external interruptions. On some models, this condition may be recognized even when the CPU is not enabled for CPU-timer interruptions. Depending on the model, the machine-check condition may be generated only as the CPU timer enters an error state. Or, the machine-check condition may be continuously generated whenever the CPU is enabled for CPU-timer interruptions, until the CPU timer is validated.
3. The clock comparator is damaged, and the CPU is enabled for clock-comparator external interruptions. On some models, this condition may be recognized even when the CPU is not enabled for clock-comparator interruptions.

Timing-facility damage may also be set along with instruction-processing damage when an instruction which accesses the TOD clock, CPU timer, or clock comparator produces incorrect results. Depending on the model, the TOD programmable register, CPU timer, or clock comparator may be validated by the interruption which reports the TOD clock, CPU timer, or clock comparator as invalid.

Timing-facility damage is a repressible condition. It is masked by the external-damage subclass-mask bit, which is in bit position 6 of control register 14.

Timing-facility-damage conditions for the CPU timer and the clock comparator are not recognized on most models when these facilities are not in use. The facilities are considered not in use when the CPU is disabled for the corresponding external interruptions (PSW bit 7, or the subclass-mask bits, bits 20 and 21 of control register 0, are zeros), and when the corresponding set and store instructions are not executed. Timing-facility-damage conditions that are already pending remain pending, however, when the CPU is disabled for the corresponding external interruption.

Timing-facility-damage conditions due to damage to the TOD clock are always recognized.


LzLabs GmbH. Ex. 1008-613

### External Damage

Bit 5 (ED), when one, indicates that damage has occurred during operations not directly associated with processing the current instruction.

When bit 5, external damage, is one and bit 26, external-damage-code validity, is also one, the external-damage code has been stored to indicate, in more detail, the cause of the external-damage machine-check interruption. When the external damage cannot be isolated to one or more of the conditions as defined in the external-damage code, or when the detailed indication for the condition is not implemented by the model, external damage is indicated with bit 26 set to zero. The presence and extent of reporting external damage depend on the model.

External damage is a repressible condition. It is masked by the external-damage subclass-mask bit, which is in bit position 6 of control register 14.

### Vector-Facility Failure

Bit 6 (VF) of the machine-check-interruption code, when one, indicates that the vector facility has failed to such an extent that the service processor has made the facility not available.

This bit may be set to one regardless of whether the vector-control bit, bit 14 of control register 0, is one or zero.

Vector-facility failure is a repressible condition and has no subclass-mask bit.

### Degradation

Bit 7 (DG), when one, indicates that continuous degradation of system performance, more serious than that indicated by system recovery, has occurred. Degradation may be reported when system-recovery conditions exceed a machine-preestablished threshold or when unit deletion has occurred. The presence and extent of the degradation-report capability depend on the model.

Degradation is a repressible condition. It is masked by the degradation subclass-mask bit, which is in bit position 5 of control register 14.

### Warning

Bit 8 (W), when one, indicates that damage is imminent in some part of the system (for example, that power is about to fail, or that a loss of cooling is occurring). Whether warning conditions are recognized depends on the model.

If the condition responsible for the imminent damage is removed before the interruption request is honored (for example, if power is restored), the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from the same condition.

Warning is a repressible condition. It is masked by the warning subclass-mask bit, which is in bit position 7 of control register 14.

### Channel Report Pending

Bit 9 (CP), when one, indicates that a channel report, consisting of one or more channel-report words, has been made pending, and the contents of the channel-report words describe, in further detail, the effect of the malfunction and the results of analysis or the action performed. A channel report becomes pending when one of the following conditions has occurred:

1. Channel-subsystem recovery has been completed. The channel-subsystem recovery may have been initiated with no prior notice to the program or may have been a result of a condition previously reported to the program.
2. The function specified by RESET CHANNEL PATH has been completed.

A channel report may also become pending under other conditions.

The channel-report words which make up the channel report may be cleared, one at a time, by execution of the instruction STORE CHANNEL REPORT WORD, which is described in Chapter 14, “I/O Instructions.”

Bit 9 is meaningless when channel-subsystem damage is reported.

Channel report pending is a floating repressible condition. It is masked by the channel-report-pending subclass-mask bit, which is in bit position 3 of control register 14.



LzLabs GmbH. Ex. 1008-614

### Service-Processor Damage

Bit 10 (SP), when one, indicates that damage has occurred to the service processor. Service-processor damage may be made pending at all CPUs in the configuration, or it may be detected independently by each CPU. The presence and extent of reporting service-processor damage depend on the model.

Service-processor damage is a repressible condition and has no subclass-mask bit.

### Channel-Subsystem Damage

Bit 11 (CK), when one, indicates that an error or malfunction has occurred in the channel subsystem, or that the channel subsystem is in the check-stop state. The channel subsystem enters the check-stop state when a malfunction occurs which is so severe that the channel subsystem cannot continue, or if power is lost in the channel subsystem.

Channel-subsystem damage is a floating repressible condition and has no subclass-mask bit.

## Subclass Modifiers

Bits 13 (VS), 14 (B), 34 (DA), and 44 (AP) of the machine-check-interruption code act as modifiers to the subclass bits.

### Vector-Facility Source

Bit 13 (VS) of the machine-check-interruption code, when one, indicates that the vector facility is the source of the reported machine-check condition. Vector-facility source is reported together with instruction-processing damage. When this bit is one, the contents of vector-facility registers may have been damaged.

This bit may be set to one regardless of whether the vector-control bit, bit 14 of control register 0, is one or zero.

Bit 13 is not meaningful when vector-facility failure is reported.

### Backed Up

Bit 14 (B), when one, indicates that the point of interruption is at a checkpoint before the point of error. This bit is meaningful only when the instruction-processing-damage bit, bit 1, is also set to one. The presence and extent of the capability to indicate a backed-up condition depend on the model.

### Delayed Access Exception

Bit 34 (DA), when one, indicates that an access exception was detected during a storage access using DAT when no such exception was detected by an earlier test for access exceptions.

Bit 34 is a modifier to instruction-processing damage (bit 1) and is meaningful only when bit 1 of the machine-check-interruption code is one. When bit 1 is zero, bit 34 has no meaning. The presence and extent of reporting delayed access exception depend on the model.

**Programming Note:** The occurrence of a delayed access exception normally indicates that the program is using an improper procedure to update the DAT tables.

### Ancillary Report

Bit 44 (AP), when one, indicates that a malfunction of a system component has occurred which has been recognized previously or which has affected the activities of multiple system elements such as CPUs and subchannels. When the malfunction affects the activities of multiple elements, an ancillary-report condition is recognized for all of the affected elements except one. This bit, when zero, indicates that this malfunction of a system component has not been recognized previously. This bit is meaningful for all conditions indicated by either the machine-check-interruption code or the external-damage code.

Depending on the model, recognition of an ancillary-report condition may not be provided, or it may not be provided for all system malfunctions. When ancillary-report recognition is not provided, bit 44 is set to zero.


LzLabs GmbH. Ex. 1008-615

## Synchronous Machine-Check-Interruption Conditions

The instruction-processing damage and backed-up bits, bits 1 and 14 of the machine-check-interruption code, identify, in combination, two conditions.

| Bit 1 | Bit 14 | Name of Condition |
|---|---|---|
| 1 | 0 | Processing damage |
| 1 | 1 | Processing backup |

### Processing Backup

The processing-backup condition indicates that the point of interruption is prior to the point, or points, of error. This is a nullifying exigent condition. When all of the other CPU-related-damage subclasses and modifiers of the machine-check-interruption code are zero, and certain validity bits associated with CPU status are indicated as valid, then the machine has successfully returned to a checkpoint prior to the malfunction, and no damage has yet occurred to the CPU.

The subclass bits which must be zero for this to be the case are as follows:

**MCIC**

| Bit | Name |
|---|---|
| 0 | System damage |
| 4 | Timing-facility damage |
| 6 | Vector-facility failure |

The subclass-modifier bits which must be zero for this to be the case are as follows:

**MCIC**

| Bit | Name |
|---|---|
| 13 | Vector-facility source |
| 34 | Delayed-access exception |

The validity bits in the machine-check-interruption code which must be one for this to be the case are as follows:

**MCIC**

| Bit | Name |
|---|---|
| 20 | PSW MWP bits |
| 21 | PSW mask and key |
| 22 | PSW program mask and condition code |
| 23 | PSW instruction address |
| 27 | Floating-point registers 0, 2, 4, and 6 |
| 28 | General registers |
| 29 | Control registers |
| 31 | Storage logical validity (result fields within current checkpoint interval) |
| 33 | Access registers |
| 43 | Floating-point registers 0-15 and the floating-point-control register |
| 46 | CPU timer |
| 47 | Clock comparator |

**Programming Note:** The processing-backup condition is reported rather than system recovery to indicate that a malfunction or failure stands in the way of continued operation of the CPU. The malfunction has not been circumvented, and damage would have occurred if instruction processing had continued.

### Processing Damage

The processing-damage condition indicates that damage has occurred to the instruction processing of the CPU. The point of interruption is a point beyond some or all of the points of damage. Processing damage is a terminating exigent condition; therefore, the contents of result fields may be unpredictable and still indicated as valid.

Processing damage may include malfunctions in program-event recording, monitor call, tracing, access-register translation, and dynamic address translation. Processing damage causes any supervisor-call-interruption condition and program-interruption condition to be discarded. However, the contents of the old PSW and interruption-code locations for these interruptions may be set to unpredictable values.

## Storage Errors

Bits 16-18 of the machine-check-interruption code are used to indicate an invalid CBC or a near-valid CBC detected in main storage or an invalid CBC in a storage key. Bit 19, storage degradation, may be indicated concurrently with bit 17. The failing-storage-address field, when indicated as valid, identifies a location within the storage checking block containing the error, or, for storage-key error uncorrected, within the block associated with the storage key. Bit 32, indirect storage error, may be set to one to indicate that the location designated by the failing-storage address is not the original source of the error.

The storage-error-uncorrected and storage-key-error-uncorrected bits do not in themselves indicate the occurrence of damage because the error



LzLabs GmbH. Ex. 1008-616

detected may not have affected a result. The portion of the configuration affected by an invalid CBC is indicated in the subclass field of the machine-check-interruption code.

Storage errors detected for a channel program, when indicated as I/O-error conditions, may also be reported as system recovery. CBC errors that occur in storage or in the storage key and that are detected on prefetched or unused data for a CPU program may or may not be reported, depending on the model.

### Storage Error Uncorrected

Bit 16 (SE), when one, indicates that a checking block in main storage contained invalid CBC and that the information could not be corrected. The contents of the checking block in main storage have not been changed. The location reported may have been accessed or prefetched for this CPU or another CPU or a channel program, or it may have been accessed as the result of a model-dependent storage access.

### Storage Error Corrected

Bit 17 (SC), when one, indicates that a checking block in main storage contained near-valid CBC and that the information has been corrected before being used. Depending on the model, the contents of the checking block in main storage may or may not have been restored to valid CBC. The location reported may have been accessed or prefetched for this CPU or for another CPU or for a channel program, or it may have been accessed as the result of a model-dependent storage access. The presence and extent of the storage-error-correction capability depend on the model. This indication may or may not be accompanied by an indication of storage degradation, bit 19 (DS).

### Storage-Key Error Uncorrected

Bit 18 (KE), when one, indicates that a storage key contained invalid CBC and that the information could not be corrected. The contents of the checking block in the storage key have not been changed. The storage key may have been accessed or prefetched for this CPU or for another CPU or for a channel program, or it may have been accessed as the result of a model-dependent storage access.

### Storage Degradation

Bit 19 (DS), when one, indicates that degradation of the recovery characteristics has occurred for the 4K-byte block reported by the failing-storage address.

Storage degradation indicates that although the associated storage error has been corrected, there are solid failures associated with the storage block (or with its associated key) that cause the correction process to take a substantial amount of time, and that if an additional error occurs in the block, the error may not be correctable or may go undetected. Thus, this bit indicates that use of the indicated block of storage should be avoided, if possible.

The indication of storage degradation has meaning only when failing-storage-address validity, MCIC bit 24, is also one. The presence and extent of reporting storage degradation depend on the model.

**Programming Note:** Because storage degradation is normally reported with system recovery, the recovery subclass mask, bit 4 of control register 14, should be set to one in order for storage degradation to be indicated.

### Indirect Storage Error

Bit 32 (IE), when one, indicates that the physical main-storage location identified by the failing-storage address is not the original source of the error. Instead, the error originated in another level of the storage hierarchy and has been propagated to the current physical-storage portion of the storage hierarchy. Bit 32 is meaningful only when bit 16 or 18 (storage error uncorrected or storage-key error uncorrected) of the machine-check-interruption code is one. When bits 16 and 18 are both zeros, bit 32 has no meaning.

For errors originating outside the storage hierarchy, the attempt to store is rejected, and the appropriate error indication is presented. When an error is detected during implicit movement of information inside the storage hierarchy, the action is not rejected and reported in this manner because the movement may be asynchronous and may be initiated as the result of an attempt to access completely unrelated information. Instead, errors in the contents of the source during implicit moving of information from one portion of the


LzLabs GmbH. Ex. 1008-617

storage hierarchy to another may be preserved in the target area by placing a special invalid CBC in the checking block associated with the target location. These propagated errors, when detected later, are reported as indirect storage errors. The original source of such an error may have been in a cache associated with an I/O processor or a CPU, or the error may have been the result of a data-path failure in transmitting data from one portion of the storage hierarchy to another. Additionally, a propagated error may be generated during the movement of data from one physical portion of storage to another as the result of a storage-reconfiguration action.

The presence and extent of reporting indirect storage error depend on the model.

**Programming Note:** See the programming notes under TEST BLOCK in Chapter 10, "Control Instructions" for the action which should be taken after storage errors are reported.

## Machine-Check Interruption-Code Validity Bits

Bits 20-24, 26-29, 31, 33, 42, 43, 46, and 47 of the machine-check-interruption code are validity bits. Each bit indicates the validity of a particular field in storage. With the exception of the storage-logical-validity bit (bit 31), each bit is associated with a field stored during the machine-check interruption. When a validity bit is one, it indicates that the saved value placed in the corresponding storage field is valid with respect to the indicated point of interruption and that no error was detected when the data was stored.

When a validity bit is zero, one or more of the following conditions may have occurred: the original information was incorrect, the original information had invalid CBC, additional malfunctions were detected while storing the information, or none or only part of the information was stored. Even though the information is unpredictable, the machine attempts, when possible, to place valid CBC in the storage field and thus reduce the possibility of additional machine checks being caused.

The validity bits for the floating-point registers, general registers, control registers, access registers, extended floating-point registers (and also the floating-point control register), CPU timer, and clock comparator indicate the validity of the saved value placed in the corresponding save area. The information in these registers after the machine-check interruption is not necessarily correct even when the correct value has been placed in the save area and the validity bit set to one. The use of the registers and the operation of the facility associated with the control registers, floating-point control register, TOD programmable register, CPU timer, and clock comparator are unpredictable until these registers are validated. (See "Invalid CBC in Registers" on page 11-10.)

### PSW-MWP Validity

Bit 20 (WP), when one, indicates that bits 12-15 of the machine-check old PSW are correct.

### PSW Mask and Key Validity

Bit 21 (MS), when one, indicates that the system mask, PSW key, and miscellaneous bits of the machine-check old PSW are correct. Specifically, this bit covers bits 0-11, 16, 17, and 24-31 of the PSW.

### PSW Program-Mask and Condition-Code Validity

Bit 22 (PM), when one, indicates that the program mask and condition code of the machine-check old PSW are correct.

### PSW-Instruction-Address Validity

Bit 23 (IA), when one, indicates that the addressing mode and instruction address (bits 32-63) of the machine-check old PSW are correct.

### Failing-Storage-Address Validity

Bit 24 (FA), when one, indicates that a correct failing-storage address has been placed at real location 248 after a storage-error-uncorrected, storage-key-error-uncorrected, or storage-error-corrected condition has occurred. The presence and extent of the capability to identify the failing-storage location depend on the model. When no such errors are reported, that is, bits 16-18 of the machine-check-interruption code are zeros, the failing-storage address is meaningless, even though it may be indicated as valid.

### External-Damage-Code Validity

Bit 26 (EC), when one, and provided that bit 5, external damage, is also one, indicates that a valid external-damage code has been stored in the word at real location 244. When bit 5 is zero, bit 26 has no meaning.



LzLabs GmbH. Ex. 1008-618

### Floating-Point-Register Validity

Bit 27 (FP), when one, indicates that the contents of the floating-point-register save area at real locations 352-383 reflect the correct state of floating-point registers 0, 2, 4, and 8 at the point of interruption.

### General-Register Validity

Bit 28 (GR), when one, indicates that the contents of the general-register save area at real locations 384-447 reflect the correct state of the general registers at the point of interruption.

### Control-Register Validity

Bit 29 (CR), when one, indicates that the contents of the control-register save area at real locations 448-511 reflect the correct state of the control registers at the point of interruption.

### Storage Logical Validity

Bit 31 (ST), when one, indicates that the storage locations, the contents of which are modified by the instructions being executed, contain the correct information relative to the point of interruption. That is, all stores before the point of interruption are completed, and all stores, if any, after the point of interruption are suppressed. When a store before the point of interruption is suppressed because of an invalid CBC, the storage-logical-validity bit may be indicated as one, provided that the invalid CBC has been preserved as invalid.

When instruction-processing damage is indicated but processing backup is not indicated, the storage-logical-validity bit has no meaning.

Storage logical validity reflects only the instruction-processing activity and does not reflect errors in the state of storage as the result of either I/O operations or the storing of the old PSW and other interruption information.

### Access-Register Validity

Bit 33 (AR), when one, indicates that the contents of the access-register save area at real locations 288-351 reflect the correct state of the access registers at the point of interruption.

### Extended-Floating-Point-Register Validity

Bit 43 (XF), when one, indicates that the contents of locations 0-143 of the machine-check extended save area reflect the correct state of floating-point registers 0-15 and the floating-point-control register at the point of interruption.

Bit 43 is zero when the basic-floating-point-extensions facility is not installed, the extended-save-area control, bit 2 of control register 14, is zero, or the machine-check extended-save-area address formed from the contents of real locations 212-215 is either invalid or all zeros.

### CPU-Timer Validity

Bit 46 (CT), when one, indicates that the CPU timer is not in error and that the contents of the CPU-timer save area at real location 216 reflect the correct state of the CPU timer at the time the interruption occurred.

### Clock-Comparator Validity

Bit 47 (CC), when one, indicates that the clock comparator is not in error and that the contents of the clock-comparator save area at real location 224 reflect the correct state of the clock comparator at the time the interruption occurred.

**Programming Note:** The validity bits must be used in conjunction with the subclass bits and the backed-up bit in order to determine the extent of the damage caused by a machine-check condition. No damage has occurred to the system when all of the following are true:

- The four PSW-validity bits, the four register-validity bits, the extended-floating-point-register validity bit, the two timing-facility-validity bits, and the storage-logical-validity bit are all ones.
- Subclass bits 0, 4, 5, 6, 10, and 11 are zeros.
- The instruction-processing-damage bit is zero or, if one, the backed-up bit is also one.
- The vector-facility-source bit and the delayed-access-exception bit are zeros.


LzLabs GmbH. Ex. 1008-619

## Machine-Check Extended Interruption Information

As part of the machine-check interruption, in some cases, extended interruption information is placed in fixed areas assigned in storage. The contents of registers associated with the CPU are placed in register save areas. For external damage, additional information is provided for some models by storing an external-damage code. When storage error uncorrected, storage error corrected, or storage-key error uncorrected is indicated, the failing-storage address is saved.

Each of these fields has associated with it a validity bit in the machine-check-interruption code. If, for any reason, the machine cannot store the proper information in the field, the associated validity bit is set to zero.

## Register Save Areas

As part of the machine-check interruption, the current contents of the CPU registers, except for the prefix register and the TOD clock, are stored in six register save areas assigned in storage. Each of these areas has associated with it a validity bit in the machine-check-interruption code. If, for any reason, the machine cannot store the proper information in the field, the associated validity bit is set to zero.

The following are the six sets of registers and the real locations in storage where their contents are saved during a machine-check interruption.

| Locations | Registers |
|---|---|
| 216-223 | CPU timer |
| 224-231 | Clock comparator |
| 288-351 | Access registers 0-15 |
| 352-383 | Floating-point registers 0, 2, 4, 6 |
| 384-447 | General registers 0-15 |
| 448-511 | Control registers 0-15 |

## Machine-Check Extended Save Area

When the basic-floating-point-extensions facility is installed, the extended-save-area control, bit 2 of control register 14, is one, and bits 1-19 of real locations 212-215 are not all zeros, then, as part of a machine-check interruption, the current contents of floating-point registers 0-15 and the floating-point-control register are stored in a machine-check extended save area. The absolute address of the extended save area is obtained by appending 12 bits to the right of bits 1-19 of real locations 212-215. Storing does not occur if the address is invalid.

The extended save area has associated with it a validity bit in the machine-check-interruption code. If, for any reason, the machine cannot store the proper information in the area, the associated validity bit is set to zero.

Figure 11-5 lists the fields that are stored, their offsets within the area, and their lengths. Bytes 144-4095 of the extended save area remain unchanged.

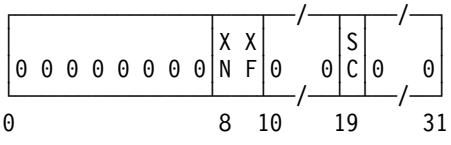
| Field | Byte Offset | Length in Bytes |
|---|---|---|
| Fl-pt registers 0-15 | 0 | 128 |
| Fl-pt-control register | 128 | 4 |
| Reserved (zeros stored) | 132 | 12 |

*Figure 11-5. Machine-Check Extended-Save-Area Locations*

## External-Damage Code

The word at real location 244 is the external-damage code. This field, when implemented and indicated as valid, describes the cause of external damage. The field is valid only when the external-damage bit and the external-damage-code-validity bit (bits 5 and 26 in the machine-check-interruption code) are both ones. The presence and extent of reporting an external-damage code depend on the model.

The external-damage code has the following format:




***Expanded Storage Not Operational (XN):*** Bit 8, when one, indicates that the controller associated with some or all of the expanded storage in the configuration has become not operational.

Expanded-storage-not-operational conditions are reported to all CPUs in the configuration.



LzLabs GmbH. Ex. 1008-620

***Expanded-Storage Control Failure (XF):*** Bit 9, when one, indicates that a malfunction has been detected in a controller associated with some or all of the expanded storage in the configuration. When expanded-storage control failure is indicated, the blocks of the expanded storage contain either the proper contents or a preserved error. Expanded-storage-control-failure conditions are reported to all CPUs in the configuration.

***ETR Sync Check (SC):*** Bit 19, when one, indicates that bits 32 through the rightmost incremented bit of a running clock are not in synchronism with the same bits of the ETR.

If the condition happens more than once before the interruption occurs, the condition is generated only once. The condition is generated for all CPUs in the configuration, and the condition for a CPU is cleared by the interruption taken by the CPU.

***Reserved:*** Bits 0-7, 10-18, and 20-31 are reserved for future expansion and are always set to zeros.

## Failing-Storage Address

When storage error uncorrected, storage error corrected, or storage-key error uncorrected is indicated in the machine-check-interruption code, the associated address, called the failing-storage address, is stored at real locations 248-251. The field is valid only if the failing-storage-address validity bit, bit 24 of the machine-check-interruption code, is one.

In the case of storage errors, the failing-storage address may designate any byte within the checking block. For storage-key error uncorrected, the failing-storage address may designate any address within the block of storage associated with the storage key that is in error. When an error is detected in more than one location before the interruption, the failing-storage address may designate any of the failing locations. The address stored is an absolute address; that is, the value stored is the address that is used to reference storage after dynamic address translation and prefixing have been applied.

# Handling of Machine-Check Conditions

## Floating Interruption Conditions

An interruption condition which is made available to any CPU in a multiprocessing configuration is called a floating interruption condition. The first CPU that accepts the interruption clears the interruption condition, and it is no longer available to any other CPU in the configuration.

Floating interruption conditions include service-signal external-interruption and I/O-interruption conditions. Two machine-check-interruption conditions, channel report pending and channel-subsystem damage, are floating interruption conditions. Depending on the model, some machine-check-interruption conditions associated with system recovery and warning may also be floating interruption conditions.

A floating interruption is presented to the first CPU in the configuration which is enabled for the interruption condition and can accept the interruption. A CPU cannot accept the interruption when the CPU is in the check-stop state, has an invalid prefix, is performing an unending string of interruptions due to a PSW-format error of the type that is recognized early, or is in the stopped state. However, a CPU with the rate control set to instruction step can accept the interruption when the start key is activated.

**Programming Note:** When a CPU enters the check-stop state in a multiprocessing configuration, the program on another CPU can determine whether a floating interruption may have been reported to the failing CPU and then lost. This can be accomplished if the interruption program places zeros in the real storage locations containing old PSWs and interruption codes after the interruption has been handled (or has been moved into another area for later processing). After a CPU enters the check-stop state, the program in another CPU can inspect the old-PSW and interruption-code locations of the failing CPU. A nonzero value in an old PSW or interruption code indicates that the CPU has been interrupted but the program did not complete the handling of the interruption.


LzLabs GmbH. Ex. 1008-621

### Floating Machine-Check-Interruption Conditions

Floating machine-check-interruption conditions are reset only by the manually initiated resets through the operator facilities. When a machine check occurs which prohibits completion of a floating machine-check interruption, the interruption condition is no longer considered a floating interruption condition, and system damage is indicated.

### Floating I/O Interruptions

The detection of a machine malfunction by the channel subsystem, while in the process of presenting an I/O-interruption request for a floating I/O interruption, may be reported as channel report pending or as channel-subsystem damage. Detection of a machine malfunction by a CPU, while in the process of accepting a floating I/O interruption, is reported as system damage.

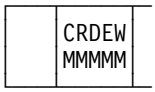
## Machine-Check Masking

All machine-check interruptions are under control of the machine-check mask, PSW bit 13. In addition, some machine-check conditions are controlled by subclass masks in control register 14.

The exigent machine-check conditions (system damage and instruction-processing damage) are controlled only by the machine-check mask, PSW bit 13. When PSW bit 13 is one, an exigent condition causes a machine-check interruption. When PSW bit 13 is zero, the occurrence of an exigent machine-check condition causes the CPU to enter the check-stop state.

The repressible machine-check conditions, except vector-facility failure, channel-subsystem damage, and service-processor damage, are controlled both by the machine-check mask, PSW bit 13, and by five subclass-mask bits in control register 14. If PSW bit 13 is one and one of the subclass-mask bits is one, the associated condition initiates a machine-check interruption. If a subclass-mask bit is zero, the associated condition does not initiate an interruption but is held pending. However, when a machine-check interruption is initiated because of a condition for which the CPU is enabled, those conditions for which the CPU is not enabled may be presented along with the condition which initiates the interruption. All conditions presented are then cleared.

Control register 14 contains mask bits that specify whether certain conditions can cause machine-check interruptions. It has the following format:




Bits 3-7 of control register 14 are the subclass masks for repressible machine-check conditions. In addition, bit 0 of control register 14 is initialized to one but is otherwise ignored by the machine.

**Programming Note:** The program should avoid, whenever possible, operating with PSW bit 13, the machine-check mask, set to zero, since any exigent machine-check condition which is recognized during this situation will cause the CPU to enter the check-stop state. In particular, the program should avoid executing I/O instructions or allowing I/O interruptions with PSW bit 13 zero.

### Channel-Report-Pending Subclass Mask

Bit 3 (CM) of control register 14 controls channel-report-pending interruption conditions. This bit is initialized to zero.

### Recovery Subclass Mask

Bit 4 (RM) of control register 14 controls system-recovery interruption conditions. This bit is initialized to zero.

### Degradation Subclass Mask

Bit 5 (DM) of control register 14 controls degradation interruption conditions. This bit is initialized to zero.

### External-Damage Subclass Mask

Bit 6 (EM) of control register 14 controls timing-facility-damage and external-damage interruption conditions. This bit is initialized to one.

### Warning Subclass Mask

Bit 7 (WM) of control register 14 controls warning interruption conditions. This bit is initialized to zero.



LzLabs GmbH. Ex. 1008-622

## Machine-Check Logout

As part of the machine-check interruption, some models may place model-dependent information in the fixed-logout area. This area is 16 bytes in length and starts at real location 256.

## Summary of Machine-Check Masking

A summary of machine-check masking is given in Figure 11-6 and Figure 11-7.

| Machine-Check Condition | | Sub-Class Mask | Action when CPU Disabled for Subclass |
|---|---|---|---|
| MCIC Bit | Subclass | | |
| 0 | System damage | - | Check stop |
| 1 | Instruction-processing damage | - | Check stop |
| 2 | System recovery | RM | Y |
| 4 | Timing-facility damage | EM | P |
| 5 | External damage | EM | P |
| 6 | Vector-facility failure | - | P |
| 7 | Degradation | DM | P |
| 8 | Warning | WM | P |
| 9 | Channel report pending | CM | P |
| 10 | Service-processor damage | - | P |
| 11 | Channel-subsystem damage | - | P |

**Explanation:**

- `-` The condition does not have a subclass mask.
- `P` Indication is held pending.
- `Y` Indication may be held pending or may be discarded.
- `CM` Channel-report-pending subclass mask (bit 3 of CR14).
- `DM` Degradation subclass mask (bit 5 of CR14).
- `EM` External-damage subclass mask (bit 6 of CR14).
- `RM` Recovery subclass mask (bit 4 of CR14).
- `WM` Warning subclass mask (bit 7 of CR14).

*Figure 11-6. Machine-Check-Condition Masking*

| Bit Description | Control Register 14 Bit Position | State of Bit on Initial CPU Reset |
|---|---|---|
| Channel-report-pending subclass mask | 3 | 0 |
| Recovery subclass mask | 4 | 0 |
| Degradation subclass mask | 5 | 0 |
| External-damage subclass mask | 6 | 1 |
| Warning subclass mask | 7 | 0 |

*Figure 11-7. Machine-Check Control-Register Bits*


LzLabs GmbH. Ex. 1008-623

LzLabs GmbH. Ex. 1008-624

# Chapter 12. Operator Facilities


Manual Operation . . . . . 12-1
Basic Operator Facilities . . . . . 12-1
Address-Compare Controls . . . . . 12-1
Alter-and-Display Controls . . . . . 12-2
Architectural-Mode Indicator . . . . . 12-2
Architectural-Mode-Selection Controls . . . 12-2
Check-Stop Indicator . . . . . 12-2
IML Controls . . . . . 12-3
Interrupt Key . . . . . 12-3
Load Indicator . . . . . 12-3
Load-Clear Key . . . . . 12-3
Load-Normal Key . . . . . 12-3
Load-Unit-Address Controls . . . . . 12-3
Manual Indicator . . . . . 12-3
Power Controls . . . . . 12-3
Rate Control . . . . . 12-4
Restart Key . . . . . 12-4
Start Key . . . . . 12-4
Stop Key . . . . . 12-4
Store-Status Key . . . . . 12-4
System-Reset-Clear Key . . . . . 12-5
System-Reset-Normal Key . . . . . 12-5
Test Indicator . . . . . 12-5
TOD-Clock Control . . . . . 12-5
Wait Indicator . . . . . 12-5
Multiprocessing Configurations . . . . . 12-6


## Manual Operation

The operator facilities provide functions for the manual operation and control of the machine. The functions include operator-to-machine communication, indication of machine status, control over the setting of the TOD clock, initial program loading, resets, and other manual controls for operator intervention in normal machine operation.

A model may provide additional operator facilities which are not described in this chapter. Examples are the means to indicate specific error conditions in the equipment, to change equipment configurations, and to facilitate maintenance. Furthermore, controls covered in this chapter may have additional settings which are not described here. Such additional facilities and settings may be described in the appropriate System Library publication.

Most models provide, in association with the operator facilities, a console device which may be used as an I/O device for operator communication with the program; this console device may also be used to implement some or all of the facilities described in this chapter.

The operator facilities may be implemented on different models in various technologies and configurations. On some models, more than one set of physical representations of some keys, controls, and indicators may be provided, such as on multiple local or remote operating stations, which may be effective concurrently.

A machine malfunction that prevents a manual operation from being performed correctly, as defined for that operation, may cause the CPU to enter the check-stop state or give some other indication to the operator that the operation has failed. Alternatively, a machine malfunction may cause a machine-check-interruption condition to be recognized.

## Basic Operator Facilities

### Address-Compare Controls

The address-compare controls provide a way to stop the CPU when a preset address matches the address used in a specified type of main-storage reference.

One of the address-compare controls is used to set up the address to be compared with the storage address.

Another control provides at least two positions to specify the action, if any, to be taken when the address match occurs:

1. The normal position disables the address-compare operation.
2. The stop position causes the CPU to enter the stopped state on an address match. When the control is in this setting, the test indicator is on. Depending on the model and the type of reference, pending I/O, external, and

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-625

machine-check interruptions may or may not be taken before entering the stopped state.

A third control may specify the type of storage reference for which the address comparison is to be made. A model may provide one or more of the following positions, as well as others:

1. The any position causes the address comparison to be performed on all storage references.
2. The data-store position causes address comparison to be performed when storage is addressed to store data.
3. The I/O position causes address comparison to be performed when storage is addressed by the channel subsystem to transfer data or to fetch a channel-command or indirect-data-address word. Whether references to the measurement block, interruption-response block, channel-path-status word, channel-report word, subchannel-status word, subchannel-information block, and operation-request block cause a match to be indicated depends on the model.
4. The instruction-address position causes address comparison to be performed when storage is addressed to fetch an instruction. The rightmost bit of the address setting may or may not be ignored. The match is indicated only when the first byte of the instruction is fetched from the selected location. It depends on the model whether a match is indicated when fetching the target instruction of EXECUTE.

Depending on the model and the type of reference, address comparison may be performed on virtual, real, or absolute addresses, and it may be possible to specify the type of address.

In a multiprocessing configuration, it depends on the model whether the address setting applies to one or all CPUs in the configuration and whether an address match causes one or all CPUs in the configuration to stop.

## Alter-and-Display Controls

The operator facilities provide controls and procedures to permit the operator to alter and display the contents of locations in storage, the storage keys, the general, floating-point, floating-point-control, access, and control registers, the prefix, and the PSW. The TOD programmable register is not included.

Before alter-and-display operations may be performed, the CPU must first be placed in the stopped state. During alter-and-display operations, the manual indicator may be turned off temporarily, and the start and restart keys may be inoperative.

Addresses used to select storage locations for alter-and-display operations are real addresses. The capability of specifying logical, virtual, or absolute addresses may also be provided.

## Architectural-Mode Indicator

The architectural-mode indicator shows the architectural mode of operation (the ESA/390 mode or some other mode) selected by the last architectural-mode-selection operation.

## Architectural-Mode-Selection Controls

The architectural-mode-selection controls provide for the selection of either the ESA/390 architectural mode of operation or, possibly, some other architectural mode of operation. Depending on the model, the architectural-mode selection may be provided as part of the IML operation or may be a separate operation.

As part of the architectural-mode-selection process, all CPUs and the associated channel-subsystem components in a particular configuration are placed in the same architectural mode.

## Check-Stop Indicator

The check-stop indicator is on when the CPU is in the check-stop state. Reset operations normally cause the CPU to leave the check-stop state and thus turn off the indicator. The manual indicator may also be on in the check-stop state.



LzLabs GmbH. Ex. 1008-626

## IML Controls

The IML controls provided with some models perform initial machine loading (IML), which is the loading of licensed internal code into the machine. The IML operation, when provided, may be used to select the ESA/390 mode or, possibly, some other mode of operation.

When the IML operation is completed, the state of the affected CPUs, PLO locks, channel subsystem, main storage, and operator facilities is the same as if a power-on reset had been performed, except that the value and state of the TOD clock are not changed. The contents of expanded storage may have been cleared to zeros with valid checking-block code or may have remained unchanged, depending on the model.

The IML controls are effective while the power is on.

## Interrupt Key

When the interrupt key is activated, an external-interruption condition indicating the interrupt key is generated. (See "Interrupt Key" on page 6-12.)

The interrupt key is effective when the CPU is in the operating or stopped state. It depends on the model whether the interrupt key is effective when the CPU is in the load state.

## Load Indicator

The load indicator is on during initial program loading, indicating that the CPU is in the load state. The indicator goes on for a particular CPU when the load-clear or load-normal key is activated for that CPU and the corresponding operation is started. It goes off after the new PSW is loaded successfully. For details, see "Initial Program Loading" on page 4-43.)

## Load-Clear Key

Activating the load-clear key causes a reset operation to be performed and initial program loading to be started by using the I/O device designated by the load-unit-address controls. Clear reset is performed on the configuration. For details, see "Resets" on page 4-37 and "Initial Program Loading" on page 4-43.

The load-clear key is effective when the CPU is in the operating, stopped, load, or check-stop state.

## Load-Normal Key

Activating the load-normal key causes a reset operation to be performed and initial program loading to be started by using the I/O device designated by the load-unit-address controls. Initial CPU reset is performed on the CPU for which the load-normal key was activated, CPU reset is propagated to all other CPUs in the configuration, and a subsystem reset is performed on the remainder of the configuration. For details, see "Resets" on page 4-37 and "Initial Program Loading" on page 4-43.

The load-normal key is effective when the CPU is in the operating, stopped, load, or check-stop state.

## Load-Unit-Address Controls

The load-unit-address controls specify four hexadecimal digits that provide the device number used for initial program loading. For details, see "Initial Program Loading" on page 4-43.

## Manual Indicator

The manual indicator is on when the CPU is in the stopped state. Some functions and several manual controls are effective only when the CPU is in the stopped state.

## Power Controls

The power controls are used to turn the power on and off.

The CPUs, storage, channel subsystem, operator facilities, and I/O devices may all have their power turned on and off by common controls, or they may have separate power controls. When a particular unit has its power turned on, that unit is reset. The sequence is performed so that no instructions or I/O operations are performed until explicitly specified. The controls may also permit power to be turned on in stages, but the machine does not become operational until power on is complete.


LzLabs GmbH. Ex. 1008-627

When the power is completely turned on, an IML operation is performed on models which have an IML function. A power-on reset is then initiated (see "Resets" on page 4-37). It depends on the model whether the architectural mode of operation can be selected when the power is turned on, or whether the mode-selection controls have to be used to change the mode after the power is on.

## Rate Control

The setting of the rate control determines the effect of the start function and the manner in which instructions are executed.

The rate control has at least two positions. The normal position is the process position. Another position is the instruction-step position. When the rate control is set to the process position and the start function is performed, the CPU starts operating at normal speed. When the rate control is set to the instruction-step position and the wait-state bit is zero, one instruction or, for interruptible instructions, one unit of operation is executed, and all pending allowed interruptions are taken before the CPU returns to the stopped state. When the rate control is set to the instruction-step position and the wait-state bit is one, no instruction is executed, but all pending allowed interruptions are taken before the CPU returns to the stopped state. For details, see "Stopped, Operating, Load, and Check-Stop States" on page 4-1.

The test indicator is on while the rate control is not set to the process position.

If the setting of the rate control is changed while the CPU is in the operating or load state, the results are unpredictable.

## Restart Key

Activating the restart key initiates a restart interruption. (See "Restart Interruption" on page 6-46.)

The restart key is effective when the CPU is in the operating or stopped state. The key is not effective when the CPU is in the check-stop state. It depends on the model whether the restart key is effective when any CPU in the configuration is in the load state.

The effect is unpredictable when the restart key is activated while any CPU in the configuration is in the load state. In particular, if the CPU performs a restart interruption and enters the operating state while another CPU is in the load state, operations such as I/O instructions, the SIGNAL PROCESSOR instruction, and the INVALIDATE PAGE TABLE ENTRY instruction may not operate according to the definitions given in this publication.

## Start Key

Activating the start key causes the CPU to perform the start function. (See "Stopped, Operating, Load, and Check-Stop States" on page 4-1.)

The start key is effective only when the CPU is in the stopped state. The effect is unpredictable when the stopped state has been entered by a reset.

## Stop Key

Activating the stop key causes the CPU to perform the stop function. (See "Stopped, Operating, Load, and Check-Stop States" on page 4-1.)

The stop key is effective only when the CPU is in the operating state.

**Operation Note:** Activating the stop key has no effect when:

- An unending string of certain program or external interruptions occurs.
- The prefix register contains an invalid address.
- The CPU is in the load or check-stop state.

## Store-Status Key

Activating the store-status key initiates a store-status operation. (See "Store Status" on page 4-43.)

The store-status key is effective only when the CPU is in the stopped state.

**Operation Note:** The store-status operation may be used in conjunction with a standalone dump program for the analysis of major program malfunctions. For such an operation, the following sequence would be called for:



LzLabs GmbH. Ex. 1008-628

1. Activation of the stop or system-reset-normal key
2. Activation of the store-status key
3. Activation of the load-normal key to enter a standalone dump program

The system-reset-normal key must be activated in step 1 when (1) the stop key is not effective because a continuous string of interruptions is occurring, (2) the prefix register contains an invalid address, or (3) the CPU is in the check-stop state.

## System-Reset-Clear Key

Activating the system-reset-clear key causes a clear-reset operation to be performed on the configuration. For details, see "Resets" on page 4-37.

The system-reset-clear key is effective when the CPU is in the operating, stopped, load, or check-stop state.

## System-Reset-Normal Key

Activating the system-reset-normal key causes a CPU-reset operation and a subsystem-reset operation to be performed. In a multiprocessing configuration, a CPU reset is propagated to all CPUs in the configuration. For details, see the section "Resets" in Chapter 4, "Control."

The system-reset-normal key is effective when the CPU is in the operating, stopped, load, or check-stop state.

## Test Indicator

The test indicator is on when a manual control for operation or maintenance is in an abnormal position that can affect the normal operation of a program.

Setting the address-compare controls to the stop position or setting the rate control to the instruction-step position turns on the test indicator.

The test indicator may be on when one or more diagnostic functions under the control of DIAGNOSE are activated, or when other abnormal conditions occur.

The abnormal setting of a manual control causes the test indicator of the affected CPU to be turned on; however, in a multiprocessing configuration, the operation of other CPUs may be affected even though their test indicators are not turned on.

**Operation Note:** If a manual control is left in a setting intended for maintenance purposes, such an abnormal setting may, among other things, result in false machine-check indications or cause actual machine malfunctions to be ignored. It may also alter other aspects of machine operation, including instruction execution, channel-subsystem operation, and the functioning of operator controls and indicators, to the extent that operation of the machine does not comply with that described in this publication.

## TOD-Clock Control

When the TOD-clock control is not activated, that is, the control is set to the secure position, the state and value of the TOD clock are protected against unauthorized or inadvertent change by not permitting the instructions SET CLOCK or DIAGNOSE to change the state or value.

When the TOD-clock control is activated, that is, the control is set to the enable-set position, alteration of the clock state or value by means of SET CLOCK or DIAGNOSE is permitted. This setting is momentary, and the control automatically returns to the secure position.

In a multiprocessing configuration, activating the TOD-clock control enables all TOD clocks in the configuration to be set. If there is more than one physical representation of the TOD-clock control, no TOD clock is secure unless all TOD-clock controls in the configuration are set to the secure position.

## Wait Indicator

The wait indicator is on when the wait-state bit in the current PSW is one. Instead of a wait indicator, a model may have a means of indicating a time-averaged value of the wait-state bit.


LzLabs GmbH. Ex. 1008-629

## Multiprocessing Configurations

In a multiprocessing configuration, one of each of the following keys and controls is provided for each CPU: alter and display, interrupt, rate, restart, start, stop, and store status. The load-clear key, load-normal key, and load-unit-address controls are provided for each CPU capable of performing I/O operations. Alternatively, a single set of initial-program-loading keys and controls may be used together with a control to select the desired CPU.

There need not be more than one of each of the following keys and controls in a multiprocessing configuration: address compare, IML, power, system reset clear, system reset normal, and TOD clock.

One check-stop, manual, test, and wait indicator is provided for each CPU. A load indicator is provided only on a CPU capable of performing I/O operations. Alternatively, a single set of indicators may be switched to more than one CPU.

There need not be more than one architectural-mode indicator in a multiprocessing configuration.

In a system capable of reconfiguration, there must be a separate set of keys, controls, and indicators in each configuration.



LzLabs GmbH. Ex. 1008-630

# Chapter 13. I/O Overview


Input/Output (I/O) . . . . . 13-1
The Channel Subsystem . . . . . 13-1
  Subchannels . . . . . 13-2
Attachment of Input/Output Devices . . . . . 13-2
  Channel Paths . . . . . 13-2
  Control Units . . . . . 13-4
  I/O Devices . . . . . 13-4
I/O Addressing . . . . . 13-5
  Channel-Path Identifier . . . . . 13-5
  Subchannel Number . . . . . 13-5
  Device Number . . . . . 13-5
  Device Identifier . . . . . 13-5
Execution of I/O Operations . . . . . 13-6
  Start-Function Initiation . . . . . 13-6
  Path Management . . . . . 13-6
  Channel-Program Execution . . . . . 13-7
  Conclusion of I/O Operations . . . . . 13-8
  I/O Interruptions . . . . . 13-9


## Input/Output (I/O)

The terms “input” and “output” are used to describe the transfer of data between I/O devices and main storage. An operation involving this kind of transfer is referred to as an I/O operation. The facilities used to control I/O operations are collectively called the channel subsystem. (I/O devices and their control units attach to the channel subsystem.) This chapter provides a brief description of the basic components and operation of the channel subsystem.

## The Channel Subsystem

The channel subsystem directs the flow of information between I/O devices and main storage. It relieves CPUs of the task of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O processing. The channel subsystem uses one or more channel paths as the communication link in managing the flow of information to or from I/O devices. As part of I/O processing, the channel subsystem also performs a path-management operation by testing for channel-path availability, chooses an available channel path, and initiates the performance of the I/O operation by the device.

Within the channel subsystem are subchannels. One subchannel is provided for and dedicated to each I/O device accessible to the program through the channel subsystem. Each subchannel provides information concerning the associated I/O device and its attachment to the channel subsystem. The subchannel also provides information concerning I/O operations and other functions involving the associated I/O device. The subchannel is the means by which the channel subsystem provides information about associated I/O devices to CPUs, which obtain this information by executing I/O instructions. The actual number of subchannels provided depends on the model and the configuration; the maximum addressability is 0-65,535.

I/O devices are attached through control units to the channel subsystem by means of channel paths. Control units may be attached to the channel subsystem by more than one channel path, and an I/O device may be attached to more than one control unit. In all, an individual I/O device may be accessible to the channel subsystem by as many as eight different channel paths via a subchannel, depending on the model and the configuration. The total number of channel paths provided by a channel subsystem depends on the model and the configuration; the maximum addressability is 0-255.

The performance of a channel subsystem depends on its use and on the system model in which it is implemented. Channel paths are provided with different data-transfer capabilities, and an I/O device designed to transfer data only at a specific rate (a magnetic-tape unit or a disk storage, for example) can operate only on a channel path that can accommodate at least this data rate.

The channel subsystem contains common facilities for the control of I/O operations. When these facilities are provided in the form of separate, autonomous equipment designed specifically to control I/O devices, I/O operations are completely overlapped with the activity in CPUs. The only main-storage cycles required by the channel sub-

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-631

system during I/O operations are those needed to transfer data and control information to or from the final locations in main storage, along with those cycles that may be required for the channel subsystem to access the subchannels when they are implemented as part of nonaddressable main storage. These cycles do not delay CPU programs, except when both the CPU and the channel subsystem concurrently attempt to reference the same main-storage area.

## Subchannels

A subchannel provides the logical appearance of a device to the program and contains the information required for sustaining a single I/O operation. The subchannel consists of internal storage that contains information in the form of a CCW address, channel-path identifier, device number, count, status indications, and I/O-interruption-subclass code, as well as information on path availability and functions pending or being performed. I/O operations are initiated with a device by the execution of I/O instructions that designate the subchannel associated with the device.

Each device is accessible by means of one subchannel in each channel subsystem to which it is assigned during configuration at installation time. The device may be a physically identifiable unit or may be housed internal to a control unit. For example, in certain disk-storage devices, each actuator used in retrieving data is considered to be a device. In all cases, a device, from the point of view of the channel subsystem, is an entity that is uniquely associated with one subchannel and that responds to selection by the channel subsystem by using the communication protocols defined for the type of channel path by which it is accessible.

On some models, subchannels are provided in blocks. On these models, more subchannels may be provided than there are attached devices. Subchannels that are provided but do not have devices assigned to them are not used by the channel subsystem to perform any function and are indicated by storing the associated device-number-valid bit as zero in the subchannel-information block of the subchannel.

The number of subchannels provided by the channel subsystem is independent of the number of channel paths to the associated devices. For example, a device accessible through alternate channel paths still is represented by a single subchannel. Each subchannel is addressed by using a 16-bit binary subchannel number.

After I/O processing at the subchannel has been requested by the execution of START SUBCHANNEL, the CPU is released for other work, and the channel subsystem assembles or disassembles data and synchronizes the transfer of data bytes between the I/O device and main storage. To accomplish this, the channel subsystem maintains and updates an address and a count that describe the destination or source of data in main storage. Similarly, when an I/O device provides signals that should be brought to the attention of the program, the channel subsystem transforms the signals into status information and stores the information in the subchannel, where it can be retrieved by the program.

# Attachment of Input/Output Devices

## Channel Paths

The channel subsystem communicates with I/O devices by means of channel paths between the channel subsystem and control units. A control unit may be accessible by the channel subsystem by more than one channel path. Similarly, an I/O device may be accessible by the channel subsystem through more than one control unit, each having one or more channel paths to the channel subsystem.

Devices that are attached to the channel subsystem by multiple channel paths configured to a subchannel, may be accessed by the channel subsystem using any of the available channel paths. Similarly, a device having the dynamic-reconnection feature and operating in the multipath mode can be initialized to operate such that the device may choose any of the available channel paths configured to the subchannel, when logically reconnecting to the channel subsystem to continue a chain of I/O operations.

The channel subsystem may contain more than one type of channel path. Examples of channel-path types used by the channel subsystem are the



LzLabs GmbH. Ex. 1008-632

ESCON I/O interface, FICON I/O interface, FICON-converted I/O interface, and IBM System/360 and System/370 I/O interface. The term "serial-I/O interface" is used to refer the ESCON I/O interface, the FICON I/O interface, and the FICON-converted I/O interface. The term "parallel-I/O interface" is used to refer to the IBM System/360 and System/370 I/O interface.

The ESCON I/O interface is described in the System Library publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202. The FICON I/O interface is described in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*. The IBM System/360 and System/370 I/O interface is described in the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974.

Depending on the type of channel path, the facilities provided by the channel path, and the I/O device, an I/O operation may occur in one of three modes, frame-multiplex mode, burst mode, or byte-multiplex mode.

In the frame-multiplex mode, the I/O device may stay logically connected to the channel path for the duration of the execution of a channel program. The facilities of a channel path capable of operating in the frame-multiplex mode may be shared by a number of concurrently operating I/O devices. In this mode the information required to complete an I/O operation is divided into frames that may be interleaved with frames from I/O operations for other I/O devices. During this period, multiple I/O devices are considered to be logically connected to the channel path.

In the burst mode, the I/O device monopolizes a channel path and stays logically connected to the channel path for the transfer of a burst of information. No other device can communicate over the channel path during the time a burst is transferred. The burst can consist of a few bytes, a whole block of data, a sequence of blocks with associated control and status information (the block lengths may be zero), or status information that monopolizes the channel path. The facilities of the channel path capable of operating in the burst mode may be shared by a number of concurrently operating I/O devices.

Some channel paths can tolerate an absence of data transfer for about a half minute during a burst-mode operation, such as occurs when a long gap on magnetic tape is read. An equipment malfunction may be indicated when an absence of data transfer exceeds the prescribed limit.

In the byte-multiplex mode, the I/O device stays logically connected to the channel path only for a short interval of time. The facilities of a channel path capable of operating in the byte-multiplex mode may be shared by a number of concurrently operating I/O devices. In this mode, all I/O operations are split into short intervals of time during which only a segment of information is transferred over the channel path. During such an interval, only one device and its associated subchannel are logically connected to the channel path. The intervals associated with the concurrent operation of multiple I/O devices are sequenced in response to demands from the devices. The channel-subsystem facility associated with a subchannel exercises its controls for any one operation only for the time required to transfer a segment of information. The segment can consist of a single byte of data, a few bytes of data, a status report from the device, or a control sequence used for the initiation of a new operation.

Ordinarily, devices with high data-transfer-rate requirements operate with the channel path in the frame-multiplex mode, slower devices operate in the burst mode, and the slowest devices operate in the byte-multiplex mode. Some control units have a manual switch for setting the desired mode of operation.

An I/O operation that occurs on a parallel-I/O-interface type of channel path may occur in either the burst mode or the byte-multiplex mode depending on the facilities provided by the channel path and the I/O device. For improved performance, some channel paths and control units are provided with facilities for high-speed transfer and data streaming. See the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974, for a description of those two facilities.

An I/O operation that occurs on a serial-I/O-interface type of channel path may occur in either the frame-multiplex mode or the burst mode. For improved performance, some control


LzLabs GmbH. Ex. 1008-633

units attaching to the serial-I/O interface provide the capability to provide sense data to the program concurrent with the presentation of unit-check status, if permitted to do so by the program. (See “Concurrent Sense” on page 17-21.)

Depending on the control unit or channel subsystem, access to a device through a subchannel may be restricted to a single channel-path type.

The modes and features described above affect only the protocol used to transfer information over the channel path and the speed of transmission. No effects are observable by CPU or channel programs with respect to the way these programs are executed.

## Control Units

A control unit provides the logical capabilities necessary to operate and control an I/O device and adapts the characteristics of each device so that it can respond to the standard form of control provided by the channel subsystem.

Communication between the control unit and the channel subsystem takes place over a channel path. The control unit accepts control signals from the channel subsystem, controls the timing of data transfer over the channel path, and provides indications concerning the status of the device.

The I/O device attached to the control unit may be designed to perform only certain limited operations, or it may perform many different operations. A typical operation is moving a recording medium and recording data. To accomplish its operations, the device needs detailed signal sequences peculiar to its type of device. The control unit decodes the commands received from the channel subsystem, interprets them for the particular type of device, and provides the signal sequence required for the performance of the operation.

A control unit may be housed separately, or it may be physically and logically integrated with the I/O device, the channel subsystem, or a CPU. In the case of most electromechanical devices, a well-defined interface exists between the device and the control unit because of the difference in the type of equipment the control unit and the device require. These electromechanical devices often are of a type where only one device of a group attached to a control unit is required to transfer data at a time (magnetic-tape units or disk-access mechanisms, for example), and the control unit is shared among a number of I/O devices. On the other hand, in some electronic I/O devices, such as the channel-to-channel adapter, the control unit does not have an identity of its own.

From the programmer's point of view, most functions performed by the control unit can be merged with those performed by the I/O device. Therefore, this publication normally makes no specific mention of the control-unit function; the performance of I/O operations is described as if the I/O devices communicated directly with the channel subsystem. Reference is made to the control unit only when emphasizing a function performed by it or when describing how the sharing of the control unit among a number of devices affects the performance of I/O operations.

## I/O Devices

An input/output (I/O) device provides external storage, a means of communication between data-processing systems, or a means of communication between a system and its environment. I/O devices include such equipment as magnetic-tape units, direct-access-storage devices (for example, disks), display units, typewriter-keyboard devices, printers, teleprocessing devices, and sensor-based equipment. An I/O device may be physically distinct equipment, or it may share equipment with other I/O devices.

Most types of I/O devices, such as printers, or tape devices, use external media, and these devices are physically distinguishable and identifiable. Other types are solely electronic and do not directly handle physical recording media. The channel-to-channel adapter, for example, provides for data transfer between two channel paths, and the data never reaches a physical recording medium outside main storage. Similarly, communication controllers may handle the transmission of information between the data-processing system and a remote station, and its input and output are signals on a transmission line.

In the simplest case, an I/O device is attached to one control unit and is accessible from one channel path. Switching equipment is available to make some devices accessible from two or more channel paths by switching devices among control



LzLabs GmbH. Ex. 1008-634

units and by switching control units among channel paths. Such switching equipment provides multiple paths by which an I/O device may be accessed. Multiple channel paths to an I/O device are provided to improve performance or I/O availability, or both, within the system. The management of multiple channel paths to devices is under the control of the channel subsystem and the device, but the channel paths may indirectly be controlled by the program.

## I/O Addressing

Four different types of I/O addressing are provided by the channel subsystem for the necessary addressing of the various components: channel-path identifiers, subchannel numbers, device numbers, and, though not visible to programs, addresses dependent on the channel-path type.

## Channel-Path Identifier

The channel-path identifier (CHPID) is a system-unique eight-bit value assigned to each installed channel path of the system. A CHPID is used to address a channel path. A CHPID is specified by the second-operand address of RESET CHANNEL PATH and used to designate the channel path that is to be reset. The channel paths by which a device is accessible are identified in the subchannel-information block (SCHIB), each by its associated CHPID, when STORE SUBCHANNEL is executed. The CHPID can also be used in operator messages when it is necessary to identify a particular channel path. A system model may provide as many as 256 channel paths. The maximum number of channel paths and the assignment of CHPIDs to channel paths depends on the system model.

## Subchannel Number

A subchannel number is a system-unique 16-bit value used to address a subchannel. This value is unique within a channel subsystem. The subchannel is addressed by eight I/O instructions: CANCEL SUBCHANNEL, CLEAR SUBCHANNEL, HALT SUBCHANNEL, MODIFY SUBCHANNEL, RESUME SUBCHANNEL, START SUBCHANNEL, STORE SUBCHANNEL, and TEST SUBCHANNEL. All I/O functions relative to a specific I/O device are specified by the program by designating a subchannel assigned to the I/O device. Subchannels are always assigned subchannel numbers within a single range of contiguous numbers. The lowest-numbered subchannel is subchannel 0. The highest-numbered subchannel of the channel subsystem has a subchannel number equal to one less than the number of subchannels provided. A maximum of 65,536 subchannels can be provided. Normally, subchannel numbers are only used in communication between the CPU program and the channel subsystem.

## Device Number

Each subchannel that has an I/O device assigned to it also contains a parameter called the device number. The device number is a 16-bit value that is assigned as one of the parameters of the subchannel at the time the device is assigned to the subchannel. The device number uniquely identifies a device to the program.

The device number provides a means to identify a device, independent of any limitations imposed by the system model, the configuration, or channel-path protocols. The device number is used in communications concerning the device that take place between the system and the system operator. For example, the device number is entered by the system operator to designate the input device to be used for initial program loading.

**Programming Note:** The device number is assigned at device-installation time and may have any value. Device numbers may be assigned installation-unique values in an installation with multiple system installations in order to avoid ambiguity, particularly where a device can be switched between two or more systems.

Additionally, the user must observe any restrictions on device-number assignment that may be required by the control program, support programs, or the particular control unit or I/O device.

## Device Identifier

A device identifier is an address, not apparent to the program, that is used by the channel subsystem to communicate with I/O devices. The type of device identifier used depends on the specific channel-path type and the protocols provided. Each subchannel contains one or more device identifiers.


LzLabs GmbH. Ex. 1008-635

For a channel path of the parallel-I/O-interface type the device identifier is called a device address and consists of an eight-bit value. For the ESCON I/O interface, the device identifier consists of a four-bit control-unit address and an eight-bit device address. For the FICON I/O interface, the device identifier consists of an eight-bit control-unit-image ID and an eight-bit device address. For the FICON-converted I/O interface, the device identifier consists of a four-bit control-unit address and an eight-bit device address.

The device address identifies the particular I/O device (and, on the parallel-I/O interface, the control unit) associated with a subchannel. The device address may identify, for example, a particular magnetic-tape drive, disk-access mechanism, or transmission line. Any number in the range 0-255 can be assigned as a device address.

For further information about the device identifier used with a particular channel-path type, see the appropriate publication for the channel-path type.

## Execution of I/O Operations

I/O operations are initiated and controlled by information with three types of formats: the instruction START SUBCHANNEL, channel-command words (CCWs), and orders. The START SUBCHANNEL instruction is executed by a CPU and is part of the CPU program that supervises the flow of requests for I/O operations from other programs that manage or process the I/O data.

When START SUBCHANNEL is executed, parameters are passed to the target subchannel requesting that the channel subsystem perform a start function with the I/O device associated with the subchannel. The channel subsystem performs the start function by using information at the subchannel, including the information passed during the execution of the START SUBCHANNEL instruction, to find an accessible channel path to the device. Once the device has been selected, the execution of an I/O operation is accomplished by the decoding and execution of a CCW by the channel subsystem and the I/O device. One or more CCWs arranged for sequential execution form a channel program and are executed as one or more I/O operations, respectively. Both instructions and CCWs are fetched from main storage, and their formats are common for all types of I/O devices, although the modifier bits in the command code of a CCW may specify device-dependent conditions for the execution of an operation at the device.

Operations peculiar to a device, such as rewinding tape or positioning the access mechanism on a disk drive, are specified by orders that are decoded and executed by I/O devices. Orders may be transferred to the device as modifier bits in the command code of a control command, may be transferred to the device as data during a control or write operation, or may be made available to the device by other means.

### Start-Function Initiation

CPU programs initiate I/O operations with the instruction START SUBCHANNEL. This instruction passes the contents of an operation-request block (ORB) to the subchannel. The contents of the ORB include the subchannel key, the address of the first CCW to be executed, and a specification of the format of the CCWs. The CCW specifies the command to be executed and the storage area, if any, to be used.

When the ORB contents have been passed to the subchannel, the execution of START SUBCHANNEL is complete. The results of the execution of the instruction are indicated by the condition code set in the program-status word.

When facilities become available, the channel subsystem fetches the first CCW and decodes it according to the format bit specified in the ORB. If the format bit is zero, format-0 CCWs are specified. If the format bit is one, format-1 CCWs are specified. Format-0 and format-1 CCWs contain the same information, but the fields are arranged differently in the format-1 CCW so that 31-bit addresses can be specified directly in the CCW.

### Path Management

If the first CCW passes certain validity tests and does not have the suspend flag specified as one, the channel subsystem attempts device selection by choosing a channel path from the group of channel paths that are available for selection. A control unit that recognizes the device identifier connects itself logically to the channel path and responds to its selection. The channel subsystem sends the command-code part of the CCW over



LzLabs GmbH. Ex. 1008-636

the channel path, and the device responds with a status byte indicating whether the command can be executed. The control unit may logically disconnect from the channel path at this time, or it may remain connected to initiate data transfer.

If the attempted selection does not occur as a result of either a busy indication or a path-not-operational condition, the channel subsystem attempts to select the device by an alternate channel path if one is available. When selection has been attempted on all paths available for selection and the busy condition persists, the operation remains pending until a path becomes free. If a path-not-operational condition is detected on one or more of the channel paths on which device selection was attempted, the program is alerted by a subsequent I/O interruption. The I/O interruption occurs either upon execution of the channel program (assuming the device was selected on an alternate channel path) or as a result of the execution being abandoned because path-not-operational conditions were detected on all of the channel paths on which device selection was attempted.

## Channel-Program Execution

If the command is initiated at the device and command execution does not require any data to be transferred to or from the device, the device may signal the end of the operation immediately on receipt of the command code. In operations that involve the transfer of data, the subchannel is set up so that the channel subsystem will respond to service requests from the device and assume further control of the operation.

An I/O operation may involve the transfer of data to or from one storage area, designated by a single CCW, or to or from a number of noncontiguous storage areas. In the latter case, generally a list of CCWs is used for the execution of the I/O operation, with each CCW designating a contiguous storage area and the CCWs are coupled by data chaining. Data chaining is specified by a flag in the CCW and causes the channel subsystem to fetch another CCW upon the exhaustion or filling of the storage area designated by the current CCW. The storage area designated by a CCW fetched on data chaining pertains to the I/O operation already in progress at the I/O device, and the I/O device is not notified when a new CCW is fetched.

Provision is made in the CCW format for the programmer to specify that, when the CCW is decoded, the channel subsystem request an I/O interruption as soon as possible, thereby notifying a CPU program that chaining has progressed at least as far as that CCW in the channel program.

To complement dynamic address translation in CPUs, CCW indirect data addressing is provided. A flag in the CCW specifies that an indirect-data-address list is to be used to designate the storage areas for that CCW. Each time the boundary of a block of storage is reached, the list is referenced to determine the next block of storage to be used. The ORB specifies whether the size of each block of storage is 2K bytes or 4K bytes. CCW indirect data addressing permits essentially the same CCW sequences to be used for a program running with dynamic address translation active in the CPU as would be used if the CPU were operating with equivalent contiguous real storage. CCW indirect data addressing permits the program to designate data blocks having absolute storage addresses up to $2^{31}$-1, independent of whether format-0 or format-1 CCWs have been specified in the ORB.

In general, the execution of an I/O operation or chain of operations involves as many as three levels of participation:

1. Except for effects due to the integration of CPU and channel-subsystem equipment, a CPU is busy for the duration of the execution of START SUBCHANNEL, which lasts until the addressed subchannel has been passed the ORB contents.
2. The subchannel is busy for a new START SUBCHANNEL from the receipt of the ORB contents until the primary interruption condition is cleared at the subchannel.
3. The I/O device is busy from the initiation of the first operation at the device until either the subchannel becomes suspended or the secondary interruption condition is placed at the subchannel. In the case of a suspended subchannel, the device again becomes busy when the execution of the suspended channel program is resumed.


LzLabs GmbH. Ex. 1008-637

## Conclusion of I/O Operations

The conclusion of an I/O operation normally is indicated by two status conditions: channel end and device end. The channel-end condition indicates that the I/O device has received or provided all data associated with the operation and no longer needs channel-subsystem facilities. This condition is called the primary interruption condition, and the channel end in this case is the primary status. Generally, the primary interruption condition is any interruption condition that relates to an I/O operation and that signals the conclusion at the subchannel of the I/O operation or chain of I/O operations.

The device-end signal indicates that the I/O device has concluded execution and is ready to perform another operation. This condition is called the secondary interruption condition, and the device end in this case is the secondary status. Generally, the secondary interruption condition is any interruption condition that relates to an I/O operation and that signals the conclusion at the device of the I/O operation or chain of operations. The secondary interruption condition can occur concurrently with, or later than, the primary interruption condition.

Concurrent with the primary or secondary interruption conditions, both the channel subsystem and the I/O device can provide indications of unusual situations.

The conditions signaling the conclusion of an I/O operation can be brought to the attention of the program by I/O interruptions or, when the CPUs are disabled for I/O interruptions, by programmed interrogation of the channel subsystem. In the former case, these conditions cause storing of the I/O-interruption code, which contains information concerning the interrupting source. In the latter case, the interruption code is stored as a result of the execution of TEST PENDING INTERRUPTION.

When the primary interruption condition is recognized, the channel subsystem attempts to notify the program, by means of an interruption request, that a subchannel contains information describing the conclusion of an I/O operation at the subchannel. The information identifies the last CCW used and may provide its residual byte count, thus describing the extent of main storage used. Both the channel subsystem and the I/O device may provide additional indications of unusual conditions as part of either the primary or the secondary interruption condition. The information contained at the subchannel may be stored by the execution of TEST SUBCHANNEL or the execution of STORE SUBCHANNEL. This information, when stored, is called a subchannel-status word (SCSW).

Facilities are provided for the program to initiate the execution of a chain of I/O operations with a single START SUBCHANNEL instruction. When the current CCW specifies command chaining and no unusual conditions have been detected during the operation, the receipt of the device-end signal causes the channel subsystem to fetch a new CCW. If the CCW passes certain validity tests and the suspend flag is not specified as a one in the new CCW, execution of a new command is initiated at the device. If the CCW fails to pass the validity tests, the new command is not initiated, command chaining is suppressed, and the status associated with the new CCW causes an interruption condition to be generated. If the suspend flag is specified as a one and this value is valid because of a one value in the suspend control, bit 4 of word 1 of the associated ORB, execution of the new command is not initiated, and command chaining is concluded.

Execution of the new command is initiated by the channel subsystem in the same way as in the previous operation. The ending signals occurring at the conclusion of an operation caused by a CCW specifying command chaining are not made available to the program. When another I/O operation is initiated by command chaining, the channel subsystem continues execution of the channel program. If, however, an unusual condition has been detected, command chaining is suppressed, the channel program is terminated, an interruption condition is generated, and the ending signals causing the termination are made available to the program.

The suspend-and-resume function provides the program with control over the execution of a channel program. The initiation of the suspend function is controlled by the setting of the suspend-control bit in the ORB. The suspend function is signaled to the channel subsystem during channel-program execution when the suspend-control bit in the ORB is one and the



LzLabs GmbH. Ex. 1008-638

suspend flag in the first CCW or in a CCW fetched during command chaining is one.

Suspension occurs when the channel subsystem fetches a CCW with the suspend flag validly (because of a one value of the suspend-control bit in the ORB) specified as one. The command in this CCW is not sent to the I/O device, and the device is signaled that the chain of commands is concluded. A subsequent RESUME SUBCHANNEL instruction informs the channel subsystem that the CCW that caused suspension may have been modified and that the channel subsystem must refetch the CCW and examine the current setting of the suspend flag. If the suspend flag is found to be zero in the CCW, the channel subsystem resumes execution of the chain of commands with the I/O device.

Channel-program execution may be terminated prematurely by CANCEL SUBCHANNEL, HALT SUBCHANNEL or CLEAR SUBCHANNEL. The execution of CANCEL SUBCHANNEL causes the channel subsystem to terminate the start function at the subchannel if the channel program has not been initiated at the device. When the start function is terminated by the execution of CANCEL SUBCHANNEL, the channel subsystem sets condition code 0 in response to the CANCEL SUBCHANNEL instruction. The execution of HALT SUBCHANNEL causes the channel subsystem to issue the halt signal to the I/O device and terminate channel-program execution at the subchannel. When channel-program execution is terminated by the execution of HALT SUBCHANNEL, the program is notified of the termination by means of an I/O-interruption request. The interruption request is generated when the device presents status for the terminated operation. If, however, the halt signal was issued to the device during command chaining after the receipt of device end but before the next command was transferred to the device, the interruption request is generated after the device has been signaled. In the latter case, the device-status field of the SCSW will contain zeros. The execution of CLEAR SUBCHANNEL clears the subchannel of indications of the channel program in execution, causes the channel subsystem to issue the clear signal to the I/O device, and causes the channel subsystem to generate an I/O-interruption request to notify the program of the completion of the clear function.

## I/O Interruptions

Conditions causing I/O-interruption requests are asynchronous to activity in CPUs, and more than one condition can occur at the same time. The conditions are preserved at the subchannels until cleared by TEST SUBCHANNEL or CLEAR SUBCHANNEL, or reset by an I/O-system reset.

When an I/O-interruption condition has been recognized by the channel subsystem and indicated at the subchannel, an I/O-interruption request is made pending for the I/O-interruption subclass specified at the subchannel. The I/O-interruption subclass for which the interruption is made pending is under programmed control through the use of MODIFY SUBCHANNEL. A pending I/O interruption may be accepted by any CPU that is enabled for interruptions from its I/O-interruption subclass. Each CPU has eight mask bits, in control register 6, that control the enablement of that CPU for each of the eight I/O-interruption subclasses, with the I/O mask, bit 6 in the PSW, being the master I/O-interruption mask for the CPU.

When an I/O interruption occurs at a CPU, the I/O-interruption code is stored in the I/O-communication area of that CPU, and the I/O-interruption request is cleared. The I/O-interruption code identifies the subchannel for which the interruption was pending. The conditions causing the generation of the interruption request may then be retrieved from the subchannel explicitly by TEST SUBCHANNEL or by STORE SUBCHANNEL.

A pending I/O-interruption request may also be cleared by TEST PENDING INTERRUPTION when the corresponding I/O-interruption subclass is enabled but the PSW has I/O interruptions disabled or by TEST SUBCHANNEL when the CPU is disabled for I/O interruptions from the corresponding I/O-interruption subclass. A pending I/O-interruption request may also be cleared by CLEAR SUBCHANNEL. Both CLEAR SUBCHANNEL and TEST SUBCHANNEL clear the preserved interruption condition at the subchannel as well.

Normally, unless the interruption request is cleared by CLEAR SUBCHANNEL, the program issues TEST SUBCHANNEL to obtain information concerning the execution of the operation.


LzLabs GmbH. Ex. 1008-639

LzLabs GmbH. Ex. 1008-640

# Chapter 14. I/O Instructions

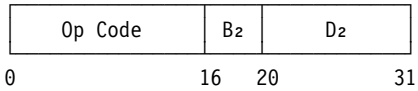

I/O-Instruction Formats . . . . . 14-1
I/O-Instruction Execution . . . . . 14-1
Serialization . . . . . 14-1
Operand Access . . . . . 14-1
Condition Code . . . . . 14-2
Program Exceptions . . . . . 14-2
Instructions . . . . . 14-2
CANCEL SUBCHANNEL . . . . . 14-4
CLEAR SUBCHANNEL . . . . . 14-5
HALT SUBCHANNEL . . . . . 14-6
MODIFY SUBCHANNEL . . . . . 14-7
RESET CHANNEL PATH . . . . . 14-9
RESUME SUBCHANNEL . . . . . 14-10
SET ADDRESS LIMIT . . . . . 14-11
SET CHANNEL MONITOR . . . . . 14-12
START SUBCHANNEL . . . . . 14-14
STORE CHANNEL PATH STATUS . . . 14-16
STORE CHANNEL REPORT WORD . . 14-17
STORE SUBCHANNEL . . . . . 14-17
TEST PENDING INTERRUPTION . . . . 14-18
TEST SUBCHANNEL . . . . . 14-20


| All the I/O instructions described here are provided for the control of channel-subsystem operations. The I/O instructions are listed in Figure 14-1 on page 14-3. All of the I/O instructions are privileged instructions.

Several I/O instructions result in the channel subsystem being signaled to perform functions asynchronous to the execution of the instructions. The description of each instruction of this type contains a section, “Associated Functions,” that summarizes the asynchronous functions.

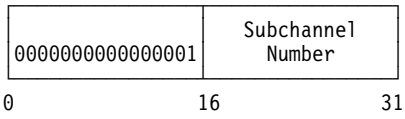
## I/O-Instruction Formats

| All I/O instructions use the S format:




The use of the second-operand address and general registers 1 and 2 (as implied operands) depends on the I/O instruction. Figure 14-1 on page 14-3 defines which operands are used to execute each I/O instruction. In addition, detailed information regarding operand usage appears in the description of each I/O instruction.

All I/O instructions that reference a subchannel use the contents of general register 1 as an implied operand. For these I/O instructions, general register 1 contains the subsystem-
| identification word. The format of the subsystem-identification word is as follows:




Bit positions 16-31 contain an unsigned binary integer designating the subchannel to be used for the function specified by the instruction. Bit positions 0-15 specify the binary number one.

## I/O-Instruction Execution

### Serialization

The execution of any I/O instruction causes serialization and checkpoint synchronization to occur. For a definition of the serialization of CPU operations, see “CPU Serialization” on page 5-91.

### Operand Access

During the execution of an I/O instruction, the order in which fields of the operand and fields of the subchannel, if applicable, are accessed is unpredictable. It is also unpredictable whether fetch accesses are made to fields of an operand or the subchannel, as applicable, when those fields are not needed to complete the execution of the I/O instruction. (See “Relation between Operand Accesses” on page 5-90.)

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-641

## Condition Code

During the execution of some I/O instructions, the results of certain tests are used to set one of four condition codes in the PSW. The I/O instructions for which execution can result in the setting of the condition code are listed in Figure 14-1 on page 14-3. The condition code indicates the result of the execution of the I/O instruction. The general meaning of the condition code for I/O instructions is given below; the meaning of the condition code for a specific instruction appears in the description of that instruction.

***Condition Code 0:*** Instruction execution produced the expected or most probable result. (See "Deferred Condition Code (CC)" on page 16-8 for a description of conditions that can be encountered subsequent to the presentation of condition code 0 that result in a nonzero deferred condition code.)

***Condition Code 1:*** Instruction execution produced the alternate or second-most-probable result, or status conditions were present that may or may not have prevented the expected result.

***Condition Code 2:*** Instruction execution was ineffective because the designated subchannel or channel-subsystem facility was busy with a previously initiated function.

***Condition Code 3:*** Instruction execution was ineffective because the designated element was not operational or because some condition precluded initiation of the normal function.

In situations where conditions exist that could cause more than one nonzero condition code to be set, the priority of the condition codes is as follows:

Condition code 3 has precedence over condition codes 1 and 2.

Condition code 1 has precedence over condition code 2.

## Program Exceptions

The program exceptions that the I/O instructions can encounter are access, operand, operation, privileged-operation, and specification exceptions. Figure 14-1 on page 14-3 shows the exceptions that are applicable to each of the I/O instructions. The execution of the instruction is suppressed for privileged-operation, operation, operand, and specification exceptions. Except as indicated otherwise in the section "Special Conditions" for each instruction, the instruction ending for access exceptions is as described in "Recognition of Access Exceptions" on page 6-35.

# Instructions

The mnemonics, format, and operation codes of the I/O instructions are given in Figure 14-1 on page 14-3. The figure also indicates the conditions that can cause a program interruption and whether the condition code is set.

In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For START SUBCHANNEL, for example, SSCH is the mnemonic and $D_2(B_2)$ the operand designation.



LzLabs GmbH. Ex. 1008-642

| Name | Mne-monic | Characteristics | | | | | Op Code |
|---|---|---|---|---|---|---|---|
| CANCEL SUBCHANNEL | XSCH | S C CI | P | OP ¢ GS | | | B276 |
| CLEAR SUBCHANNEL | CSCH | S C | P | OP ¢ GS | | | B230 |
| HALT SUBCHANNEL | HSCH | S C | P | OP ¢ GS | | | B231 |
| MODIFY SUBCHANNEL | MSCH | S C | P A SP | OP ¢ GS | | $B_2$ | B232 |
| RESET CHANNEL PATH | RCHP | S C | P | OP ¢ G1 | | | B23B |
| RESUME SUBCHANNEL | RSCH | S C | P | OP ¢ GS | | | B238 |
| SET ADDRESS LIMIT | SAL | S | P | OP ¢ G1 | | | B237 |
| SET CHANNEL MONITOR | SCHM | S | P | OP ¢ GM | | | B23C |
| START SUBCHANNEL | SSCH | S C | P A SP | OP ¢ GS | | $B_2$ | B233 |
| STORE CHANNEL PATH STATUS | STCPS | S | P A SP | ¢ | ST | $B_2$ | B23A |
| STORE CHANNEL REPORT WORD | STCRW | S C | P A SP | ¢ | ST | $B_2$ | B239 |
| STORE SUBCHANNEL | STSCH | S C | P A SP | OP ¢ GS | ST | $B_2$ | B234 |
| TEST PENDING INTERRUPTION | TPI | S C | P $A^1$ SP | ¢ | ST | $B_2$ | B236 |
| TEST SUBCHANNEL | TSCH | S C | P A SP | OP ¢ GS | ST | $B_2$ | B235 |

**Explanation:**

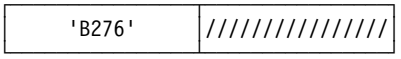
¢ Causes serialization and checkpoint synchronization.
A Access exceptions for logical addresses.
$A^1$ When the effective address is zero, it is not used to access storage, and no access exceptions can occur, except that access exceptions may occur during access-register translation.
$B_2$ $B_2$ field designates an access register in the access-register mode.
C Condition code is set.
CI Cancel-I/O facility.
G1 Instruction execution includes the implied use of general register 1 as a parameter.
GM Instruction execution includes the implied use of multiple general registers.
GS Instruction execution includes the implied use of general register 1 as the subsystem-identification word.
P Privileged-operation exception.
S S instruction format.
SP Specification exception.
ST PER storage-alteration event.

*Figure 14-1. Summary of I/O Instructions*


LzLabs GmbH. Ex. 1008-643

## CANCEL SUBCHANNEL

XSCH [S]




The current start function, if any, is terminated at the designated subchannel if CANCEL SUBCHANNEL is applicable.

General register 1 contains a subsystem-identification word that designates the subchannel for which the current START FUNCTION, if any, is to be terminated.

If the subchannel (1) is not subchannel active, (2) is start pending, resume pending, or suspended, and (3) is performing only the start function, then the start function at the subchannel is terminated, and the subchannel is made no longer start pending, resume pending, or suspended, as appropriate. In addition, internal indications of busy are reset for the subchannel.

Condition code 0 is set to indicate that the actions described above have been taken.

If an invalid ORB field or a no-path-available condition is present for a previously initiated start function and the condition was not reported during the execution of START SUBCHANNEL, condition code 0 may be indicated for CANCEL SUBCHANNEL provided that the subchannel is not yet status pending to report the error condition; if condition code 0 is presented, no subsequent status is generated to indicate the error condition.

**Special Conditions**

***Condition code 1*** is set, and no other action is taken, when the subchannel is status pending with any status.

***Condition code 2*** is set, and no other action is taken, when CANCEL SUBCHANNEL is not applicable and the subchannel is not status pending. CANCEL SUBCHANNEL is not applicable when the subchannel (1) has no function specified, (2) has a function other than the start function alone specified, (3) is not resume pending, is not start pending, and is not suspended, or (4) is subchannel active.

***Condition code 3*** is set, and no other action is taken, when the subchannel is not operational for CANCEL SUBCHANNEL. A subchannel is not operational for CANCEL SUBCHANNEL when the subchannel is not provided by the channel subsystem, has no valid device number assigned to it, or is not enabled.

CANCEL SUBCHANNEL can encounter the program exceptions described or listed below.

Bits 0-15 of the SID must contain 0001 hex; otherwise, an operand exception is recognized.

***Resulting Condition Code:***

0 Start function canceled
1 Status pending
2 CANCEL SUBCHANNEL not applicable
3 Not operational

***Program Exceptions:***

- Operand
- Operation (if the cancel-I/O facility is not installed)
- Privileged operation

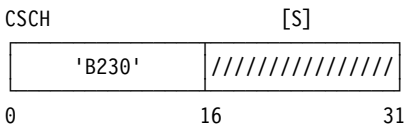
**Programming Notes:**

1. The actions taken by CANCEL SUBCHANNEL are completed during the execution of the instruction. If condition code 0 is presented, there is no subsequent I/O interruption resulting from the terminated I/O operation. However, the device may have signaled a busy condition while the canceled operation was start pending. In this case, the device owes a no-longer-busy signal to the channel subsystem. This may result in unsolicited device-end status before the next operation is initiated at the device.
2. Upon the completion of CANCEL SUBCHANNEL with condition code 0, the subchannel is ready to accept a new start function initiated by START SUBCHANNEL.



LzLabs GmbH. Ex. 1008-644

## CLEAR SUBCHANNEL




The designated subchannel is cleared, the current start or halt function, if any, is terminated at the designated subchannel, and the channel subsystem is signaled to asynchronously perform the clear function at the designated subchannel and at the associated device.

General register 1 contains a subsystem-identification word (SID) that designates the subchannel to be cleared.

If a start or halt function is in progress, it is terminated at the subchannel.

The subchannel is made no longer status pending. All activity, as indicated in the activity-control field of the SCSW, is cleared at the subchannel, except that the subchannel is made clear pending. Any functions in progress, as indicated in the function-control field of the SCSW, are cleared at the subchannel, except for the clear function that is to be performed because of the execution of this instruction.

The channel subsystem is signaled to asynchronously perform the clear function. The clear function is summarized below in the section "Associated Functions" and is described in detail in "Clear Function" on page 15-14.

Condition code 0 is set to indicate that the actions described above have been taken.

### Associated Functions

Subsequent to the execution of CLEAR SUBCHANNEL, the channel subsystem asynchronously performs the clear function. If conditions allow, the channel subsystem chooses a channel path and attempts to issue the clear signal to the device to terminate the I/O operation, if any. The subchannel then becomes status pending. Conditions encountered by the channel subsystem that preclude issuing the clear signal to the device do not prevent the subchannel from becoming status pending (see "Clear Function" on page 15-14).

When the subchannel becomes status pending as a result of performing the clear function, data transfer, if any, with the associated device has been terminated. The SCSW stored when the resulting status is cleared by TEST SUBCHANNEL has the clear-function bit stored as one. If the channel subsystem can determine that the clear signal was issued to the device, the clear-pending bit is stored as zero in the SCSW. Otherwise, the clear-pending bit is stored as one, and other indications are provided that describe in greater detail the condition that was encountered. (See "Interruption-Response Block" on page 16-6.)

Measurement data is not accumulated, and device-connect time is not stored in the extended-status word for the subchannel, for a start function that is terminated by CLEAR SUBCHANNEL.

### Special Conditions

***Condition code 3*** is set, and no other action is taken, when the subchannel is not operational for CLEAR SUBCHANNEL. A subchannel is not operational for CLEAR SUBCHANNEL when the subchannel is not provided in the channel subsystem, has no valid device number assigned to it, or is not enabled.

CLEAR SUBCHANNEL can encounter the program exceptions described or listed below.

Bits 0-15 of the SID must contain 0001 hex; otherwise, an operand exception is recognized.

***Resulting Condition Code:***

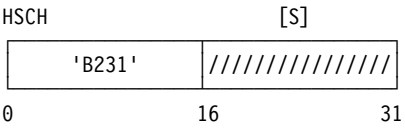
0 Function initiated
1 --
2 --
3 Not operational

***Program Exceptions:***

- Operand
- Privileged operation


LzLabs GmbH. Ex. 1008-645

## HALT SUBCHANNEL




The current start function, if any, is terminated at the designated subchannel, and the channel subsystem is signaled to asynchronously perform the halt function at the designated subchannel and at the associated device.

General register 1 contains a subsystem-identification word that designates the subchannel to be halted.

If a start function is in progress, it is terminated at the subchannel.

The subchannel is made halt pending, and the halt function is indicated at the subchannel.

When HALT SUBCHANNEL is executed and the designated subchannel is subchannel-and-device active and status pending with intermediate status, the status-pending indication is eliminated (see the discussion of bits 24, 25, and 28 in "Activity Control (AC)" on page 16-13). The status-pending condition is reestablished as part of the halt function (see the section "Associated Functions" below).

The channel subsystem is signaled to asynchronously perform the halt function. The halt function is summarized below in the section "Associated Functions" and is described in detail in "Halt Function" on page 15-15.

Condition code 0 is set to indicate that the actions described above have been taken.

**Associated Functions**

Subsequent to the execution of HALT SUBCHANNEL, the channel subsystem asynchronously performs the halt function. If conditions allow, the channel subsystem chooses a channel path and attempts to issue the halt signal to the device to terminate the I/O operation, if any. The subchannel then becomes status pending.

When the subchannel becomes status pending as a result of performing the halt function, data transfer, if any, with the associated device has been terminated. The SCSW stored when the resulting status is cleared by TEST SUBCHANNEL has the halt-function bit stored as one. If the halt signal was issued to the device, the halt-pending bit is stored as zero. Otherwise, the halt-pending bit is stored as one, and other indications are provided that describe in greater detail the condition that was encountered. (See "Interruption-Response Block" on page 16-6 and "Halt Function" on page 15-15.)

On some models, path availability is tested as part of the halt function instead of as part of the execution of the instruction. In these models, when no channel path is available for selection, the halt signal is not issued, and the subchannel is made status pending. When the status-pending condition is subsequently cleared by TEST SUBCHANNEL, the halt-pending bit is stored as one in the SCSW.

If a status-pending condition is eliminated during the execution of HALT SUBCHANNEL, then this condition is reestablished along with the other status conditions when the completion of the halt function is indicated to the program.

The halt-pending condition may not be recognized by the channel subsystem if a status-pending condition has been generated. This situation could occur, for example, when alert status is presented or generated while the subchannel is already start pending or resume pending, or when primary status is presented during the attempt to initiate the I/O operation for the first command as specified by the start function or implied by the resume function. If recognition of the status-pending condition by the channel subsystem has occurred logically prior to recognition of the halt-pending condition, the SCSW, when cleared by TEST SUBCHANNEL, has the halt-pending bit stored as one.

If measurement data is being accumulated when a start function is terminated by HALT SUBCHANNEL, the measurement data continues to be accumulated for the subchannel and reflects the extent of subchannel and device usage required, if any, while performing the currently terminated start function. The measurement data, if any, is accumulated in the measurement block for the subchannel or placed in the extended-status word, as appropriate, when the subchannel becomes status-pending with primary or secondary status.



LzLabs GmbH. Ex. 1008-646

(See "Channel-Subsystem Monitoring" on page 17-1.)

**Special Conditions**

***Condition code 1*** is set, and no other action is taken, when the subchannel is status pending alone or is status pending with any combination of alert, primary, or secondary status.

***Condition code 2*** is set, and no other action is taken, when the subchannel is busy for HALT SUBCHANNEL. The subchannel is busy for HALT SUBCHANNEL when a halt function or clear function is already in progress at the subchannel.

***Condition code 3*** is set, and no other action is taken, when the subchannel is not operational for HALT SUBCHANNEL. A subchannel is not operational for HALT SUBCHANNEL when the subchannel is not provided in the channel subsystem, has no valid device number assigned to it, or is not enabled. On some models, a subchannel is also not operational for HALT SUBCHANNEL when no channel path is available for selection by the device. (See "Channel-Path Availability" on page 15-13 for a description of channel paths that are available for selection.)

HALT SUBCHANNEL can encounter the program exceptions described or listed below.

Bits 0-15 of the SID must contain 0001 hex; otherwise, an operand exception is recognized.

***Resulting Condition Code:***

0 Function initiated
1 Status pending with other than intermediate status
2 Busy
3 Not operational

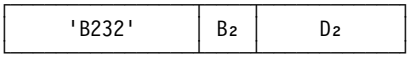
***Program Exceptions:***

- Operand
- Privileged operation

**Programming Note:** After the execution of HALT SUBCHANNEL, the status-pending condition indicating the completion of the halt function may be delayed for an extended period of time, for example, when the device is a magnetic-tape unit executing a rewind command.

# MODIFY SUBCHANNEL

MSCH $D_2(B_2)$ [S]




The information contained in the subchannel-information block (SCHIB) is placed in the program-modifiable fields at the subchannel. As a result, the program influences, for that subchannel, certain aspects of I/O processing relative to the clear, halt, resume, and start functions and certain I/O support functions.

General register 1 contains a subsystem-identification word (SID) that designates the subchannel that is to be modified as specified by certain fields of the SCHIB. The second-operand address is the logical address of the SCHIB and must be designated on a word boundary; otherwise, a specification exception is recognized.

The channel-subsystem operations that may be influenced due to placement of SCHIB information in the subchannel are:

- I/O processing (E field)
- Interruption processing ( interruption parameter and ISC field)
- Path management (D, LPM, and POM fields)
- Monitoring and address-limit checking (measurement-block index, LM, and MM fields)
- Concurrent-sense facility (S field)
- Measurement-block address (MBA)

Bits 0, 1, 6, and 7 of word 1, and bits 0-28 of word 6 of the SCHIB operand must be zeros, and bits 9 and 10 of word 1 must not both be ones. When the extended-I/O-measurement-block facility is installed, word 10, and bit 0 and bits 26-31 of word 11 must be specified as zeros. When the extended-I/O-measurement-block facility is not installed, bit 29 of word 6 must be specified as zero; otherwise, an operand exception is recognized. When the extended-I/O-measurement-word facility is not installed, or is installed but not enabled, bit 30 of word 6 must be specified as zero; otherwise, an operand exception is recognized. The remaining fields of the SCHIB are ignored and do not affect the processing of MODIFY SUBCHANNEL. (For further details, see "Subchannel-Information Block" on page 15-1.)


LzLabs GmbH. Ex. 1008-647

Condition code 0 is set to indicate that the information from the SCHIB has been placed in the program-modifiable fields at the subchannel, except that, when the device-number-valid bit (V) at the designated subchannel is zero, then condition code 0 is set, and the information from the SCHIB is not placed in the program-modifiable fields.

**Special Conditions**

***Condition code 1*** is set, and no other action is taken, when the subchannel is status pending. (See "Status Control (SC)" on page 16-16.)

***Condition code 2*** is set, and no other action is taken, when a clear, halt, or start function is in progress at the subchannel. (See "Function Control (FC)" on page 16-12.)

***Condition code 3*** is set, and no other action is taken, when the subchannel is not operational for MODIFY SUBCHANNEL. A subchannel is not operational for MODIFY SUBCHANNEL when the subchannel is not provided in the channel subsystem.

MODIFY SUBCHANNEL can encounter the program exceptions described or listed below.

In word 1 of the SCHIB, bits 0, 1, 6, and 7 must be zeros, and bits 9 and 10 must not both be ones. In word 6 of the SCHIB, bits 0-28 must be zeros. Otherwise an operand exception is recognized.

When the extended-I/O-measurement-block facility is installed, word 10, and bit 0 and bits 26-31 of word 11 must be specified as zeros; otherwise, an operand exception is recognized. When the extended-I/O-measurement-block facility is not installed, bit 29 or word 6 must be specified as zero; otherwise, an operand exception is recognized. When the extended-I/O-measurement-word facility is not installed, or is installed but not enabled, bit 30 or word 6 must be specified as zero; otherwise, an operand exception is recognized.

Bits 0-15 of the SID must contain 0001 hex; otherwise, an operand exception is recognized.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The execution of MODIFY SUBCHANNEL is suppressed on all addressing and protection exceptions.

***Resulting Condition Code:***

0 Function completed
1 Status pending
2 Busy
3 Not operational

***Program Exceptions:***

- Access (fetch, operand 2)
- Operand
- Privileged operation
- Specification

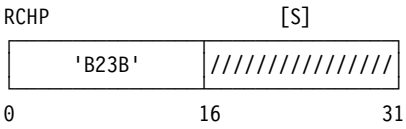
**Programming Notes:**

1. If a device signals I/O-error alert while the associated subchannel is disabled, the channel subsystem issues the clear signal to the device and discards the I/O-error-alert indication without generating an I/O-interruption condition.
2. If a device presents unsolicited status while the associated subchannel is disabled, that status is discarded by the channel subsystem without generating an I/O-interruption condition. However, if the status presented contains unit check, the channel subsystem issues the clear signal for the associated subchannel and does not generate an I/O-interruption condition. This should be taken into account when the program uses MODIFY SUBCHANNEL to enable a subchannel. For example, the medium on the associated device that was present when the subchannel became disabled may have been replaced, and, therefore, the program should verify the integrity of that medium.
3. It is recommended that the program inspect the contents of the subchannel by subsequently issuing STORE SUBCHANNEL when MODIFY SUBCHANNEL sets condition code 0. Use of STORE SUBCHANNEL is a method for determining if the designated subchannel was changed or not. Failure to inspect the subchannel following the setting of condition code 0 by MODIFY SUBCHANNEL may result in conditions that the program does not expect to occur.



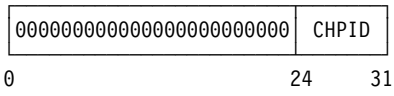
LzLabs GmbH. Ex. 1008-648

## RESET CHANNEL PATH




The channel-path-reset facility is signaled to perform the channel-path-reset function on the channel path designated by the contents of general register 1.

The format of general register 1 is as follows:




***Channel-Path Identifier (CHPID):*** Bit positions 24-31 of general register 1 contain an unsigned binary integer that designates the channel path on which the channel-path-reset function is to be performed.

Bit positions in general register 1 that are shown as zeros, are reserved and must contain zeros; otherwise, an operand exception is recognized.

If conditions allow, the channel-path-reset facility is signaled to asynchronously perform the channel-path-reset function on the designated channel path. The channel-path-reset function is summarized below in the section "Associated Functions" and is described in detail in "Channel-Path Reset" on page 17-13.

Condition code 0 is set to indicate that the channel-path-reset facility has been signaled.

**Associated Functions**

Subsequent to the execution of RESET CHANNEL PATH, the channel-path-reset facility asynchronously performs the channel-path-reset function. Certain indications are reset at all subchannels that have access to the designated channel path, and the reset signal is issued on that channel path. Any I/O functions in progress at the devices are reset, but only for the channel path on which the reset signal is received. An I/O operation or chain of I/O operations taking place in the multipath mode may be able to continue to be executed on other channel paths in the multipath group, if any. (See "Channel-Path-Reset Function" on page 15-45.)

The result of performing the channel-path-reset function on the designated channel path is communicated to the program by means of a channel report (see "Channel Report" on page 17-22).

**Special Conditions**

***Condition code 2*** is set, and no other action is taken, when, on some models, the channel-path-reset facility is busy performing the channel-path-reset function for a previous execution of the RESET CHANNEL PATH instruction.

***Condition code 3*** is set, and no other action is taken, when, on some models, the designated channel path is not operational for the execution of RESET CHANNEL PATH. On these models, the channel path is not operational for the execution of RESET CHANNEL PATH when the designated channel path is not physically available.

If the channel-path-reset facility is busy and the designated channel path is not physically available, it depends on the model whether condition code 2 or 3 is set.

RESET CHANNEL PATH can encounter the program exceptions described or listed below.

Bit positions 0-23 of general register 1 must contain zeros; otherwise, an operand exception is recognized.

***Resulting Condition Code:***

0 Function initiated
1 --
2 Busy
3 Not operational

***Program Exceptions:***

- Operand
- Privileged operation

**Programming Notes:**

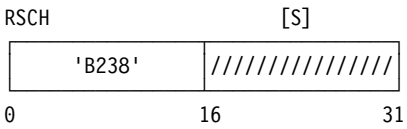
1. To eliminate the possibility of a data-integrity exposure for devices that have the capability of generating unsolicited device-end status, I/O operations in progress with such devices on the channel path for which RESET CHANNEL PATH is to be executed must be


LzLabs GmbH. Ex. 1008-649

terminated by the execution of either HALT SUBCHANNEL or CLEAR SUBCHANNEL. Otherwise, subsequent to receiving the reset signal, the device may present an unsolicited device end that may be interpreted by the channel subsystem as a solicited device end and cause command chaining to occur.

2. If the status-verification facility is being used and RESET CHANNEL PATH is executed without first stopping all ongoing operations associated with the channel path being reset, erroneous device-status-check conditions may be detected.

## RESUME SUBCHANNEL




The channel subsystem is signaled to perform the resume function at the designated subchannel.

General register 1 contains a subsystem-identification word that designates the subchannel at which the resume function is to be performed.

The subchannel is made resume pending.

Logically prior to the setting of condition code 0 and only if the subchannel is currently in the suspended state, path-not-operational conditions at the subchannel, if any, are cleared.

The channel subsystem is signaled to asynchronously perform the resume function. The resume function is summarized below in the section "Associated Functions" and is described in detail in "Start Function and Resume Function" on page 15-18.

Condition code 0 is set to indicate that the actions described above have been taken.

### Associated Functions

Subsequent to the execution of RESUME SUBCHANNEL, the channel subsystem asynchronously performs the resume function. Except when the subchannel is subchannel active, if the execution of RESUME SUBCHANNEL results in the setting of condition code 0, performance of the resume function causes execution of a currently suspended channel program to be resumed with the associated device, provided that the suspend flag for the current CCW has been set to zero by the program. If the suspend flag remains one, execution of the channel program remains suspended. But, if the subchannel is subchannel active at the time the execution of RESUME SUBCHANNEL results in the setting of condition code 0, then it is unpredictable whether execution of the current program is resumed or whether it is found by the resume function that the subchannel has become suspended in the interim. The subchannel is found to be suspended by the resume function only if the subchannel is status pending with intermediate status when the resume-pending condition is recognized by the channel subsystem. (See "Start Function and Resume Function" on page 15-18.)

### Special Conditions

***Condition code 1*** is set, and no other action is taken, when the subchannel is status pending.

***Condition code 2*** is set, and no other action is taken, when the resume function is not applicable. The resume function is not applicable when the subchannel (1) has any function other than the start function alone specified, (2) has no function specified, (3) is resume pending, or (4) does not have suspend control specified for the start function in progress.

***Condition code 3*** is set, and no other action is taken, when the subchannel is not operational for the resume function. A subchannel is not operational for the resume function if the subchannel is not provided in the channel subsystem, has no valid device number assigned to it, or is not enabled.

| RESUME SUBCHANNEL can encounter the
| program exceptions described or listed below.

| Bits 0-15 of the SID must contain 0001 hex; otherwise, an operand exception is recognized.

***Resulting Condition Code:***

0 Function initiated
1 Status pending
2 Function not applicable
3 Not operational



LzLabs GmbH. Ex. 1008-650

***Program Exceptions:***

- Operand
- Privileged operation

**Programming Notes:**

1. When channel-program execution is resumed from the suspended state, the device views the resumption as the beginning of a new chain of commands. When the suspension of channel-program execution occurs and the device requires that certain commands be first or appear only once in a chain of commands (for example, direct-access-storage devices), the program must ensure that the appropriate commands in the proper sequence are fetched by the channel subsystem after channel-program execution is resumed. One way the program can ensure proper sequencing of commands at the device is by allowing the I/O interruption to occur for an intermediate interruption condition due to suspension.

   It is not reliable to notify the program that the subchannel is suspended by using the PCI flag in the CCW that contains the S flag because the PCI I/O interruption may occur before the subchannel is suspended. The SCSW would indicate that an I/O operation is in progress at the subchannel and device in this case.

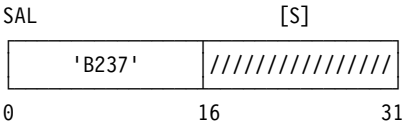
   The suspend flag of the target CCW should be set to zero before RESUME SUBCHANNEL is executed; otherwise, it is possible that the resume-pending condition may be recognized and the CCW refetched while the suspend flag is still one, in which case the resume-pending condition would be reset, and the execution of the channel program would be suspended. If the suspend flag of the target CCW is set to zero before the execution of RESUME SUBCHANNEL, the channel program is not suspended, provided that the subchannel is not subchannel active at the time the execution of RESUME SUBCHANNEL results in the setting of condition code 0. If condition code 0 is set while the subchannel is still subchannel active, it is unpredictable whether the resume-pending condition is recognized by the channel subsystem or whether it is found by the resume function that the subchannel has become suspended in the interim. The subchannel is found to be suspended by the resume function only if the subchannel is status pending with intermediate status at the time the resume-pending condition is recognized. When the subchannel is suspended, the execution of TEST SUBCHANNEL, which clears the intermediate interruption condition, also clears the indication of resume pending.

2. Some models recognize a resume-pending condition only after a CCW having an S flag validly set to one is fetched. Therefore, if a subchannel is resume pending and, during the execution of the channel program, no CCW is fetched having an S flag validly set to one, the subchannel remains resume pending until the primary interruption condition is cleared by TEST SUBCHANNEL.

3. Path availability is not tested during the execution of RESUME SUBCHANNEL. Instead, path availability is tested when the channel subsystem begins performance of the resume function.

4. The contents of the CCW fetched during performance of the resume function may be different from the contents of the same CCW when it was previously fetched and contained an S flag validly set to one.

## SET ADDRESS LIMIT

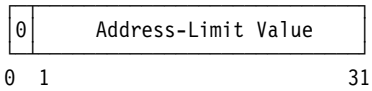



The address-limit-checking facility is signaled to use the specified address as the address-limit value, and the specified address is passed to the facility. Depending on the model, this instruction may not be provided. When this instruction is not provided, it is checked for operand exception and privileged-operation exception, and then is suppressed.

General register 1 contains the absolute address to be used as the address-limit value. The specified address must be on a 64K-byte boundary and may designate a maximum absolute storage address of 2,147,418,112 (7FFF0000 hex) regardless of whether the CPU is operating in the 24-bit or 31-bit addressing mode.


LzLabs GmbH. Ex. 1008-651

General register 1 has the following format:




**Associated Functions**

The value that is used by the address-limit-checking facility when determining whether to permit or prohibit a data access is called the address-limit value. The initial address-limit value is zero. The initial address-limit value is used by the address-limit-checking facility until the facility recognizes a signal, caused by the execution of SET ADDRESS LIMIT, to use a specified address. The recognition of this specified address as the new address-limit value occurs asynchronously with respect to the execution of SET ADDRESS LIMIT.

If address-limit checking is specified for a subchannel, then whether the specified address is used by the address-limit-checking facility, when determining whether to permit or prohibit a data access, depends on whether SET ADDRESS LIMIT was executed before, during, or after the execution of START SUBCHANNEL for that subchannel. If SET ADDRESS LIMIT is executed before START SUBCHANNEL, the specified address is used by the address-limit-checking facility. If SET ADDRESS LIMIT is executed during or after the execution of START SUBCHANNEL, it is unpredictable whether the specified address is used by the address-limit-checking facility for that particular start function. For a description of the manner in which address-limit checking is performed, see "Address-Limit Checking" on page 17-20.

**Special Conditions**

SET ADDRESS LIMIT can encounter the program exceptions described or listed below.

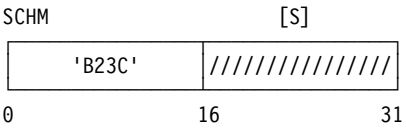
The address in general register 1 must be designated on a 64K byte boundary, and the leftmost bit of general register 1 must be zero; otherwise, an operand exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

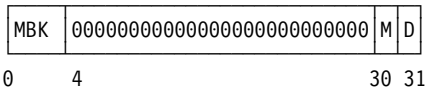
- Operand
- Privileged operation

# SET CHANNEL MONITOR




Each of the measurement-block-update mode and device-connect-time-measurement mode of the channel subsystem is made either active or inactive, depending on the values of the measurement-mode-control bits in general register 1. If the measurement-mode-control bit for measurement-block update is one, the measurement-block origin and measurement-block key are passed to the channel subsystem.

General register 1 has the following format:




***Measurement-Block Key (MBK):*** Bit positions 0-3 of general register 1 contain the measurement-block key. When bit 30 is one, MBK specifies the access key that is to be used by the channel subsystem when it accesses the measurement-block area and, when the extended-I/O-measurement-block facility is installed, to access format-1 measurement blocks. Otherwise, MBK is ignored.

***Measurement-Block-Update Control (M):*** Bit 30 of general register 1 is the measurement-mode-control bit that controls the measurement-block-update mode. When bit 30 of general register 1 is one and conditions allow, the measurement-block-update facility is signaled to asynchronously make the measurement-block-update mode active. In addition, the measurement-block-origin (MBO) address in general register 2 and the measurement-block key (MBK) in general register 1 are passed to the measurement-block-update facility. Furthermore, when bit 30 is one, bit 0 of general register 2 must be zero. The measurement-block origin is used to determine the location of format-0 measurement blocks; the address of format-1 measurement



LzLabs GmbH. Ex. 1008-652

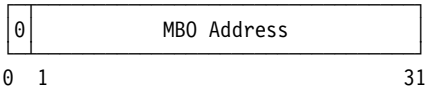
blocks is stored at the subchannel using MODIFY SUBCHANNEL. The measurement-block key is used to access both format-0 and format-1 measurement blocks. The asynchronous functions that are performed by the measurement-block-update facility are summarized below in the section "Associated Functions" and are described in detail in "Channel-Subsystem Monitoring" on page 17-1.

When bit 30 of general register 1 is zero and conditions allow, the measurement-block-update mode is made inactive if it is active or remains inactive if it is inactive. The contents of bit positions 0-3 (MBK) of general register 1 and the contents of general register 2 are ignored.

***Device-Connect-Time-Measurement Control (D):*** Bit 31 of general register 1 is the measurement-mode-control bit (D). When bit 31 is one and conditions allow, the device-connect-time-measurement mode is made active if it is inactive or remains active if it is active. When bit 31 is zero and conditions allow, the device-connect-time-measurement mode is made inactive if it is active or remains inactive if it is inactive.

The remaining bit positions of general register 1 are reserved and must contain zeros; otherwise, an operand exception is recognized.

General register 2 has the following format:




***Measurement-Block-Origin (MBO) Address:*** When bit 30 (M) of general register 1 is one, bit positions 1-31 of general register 2 contain the absolute address of the measurement-block origin (MBO), which is the beginning of the measurement-block area. The MBO address is used by the channel subsystem to locate format-0 measurement blocks. The origin of the measurement-block area must be designated on a 32-byte boundary, and, when bit 30 of general register 1 is one, bit 0 of general register 2 must be zero; otherwise, an operand exception is recognized. When bit 30 of general register 1 is zero, the contents of general register 2 are ignored.

If the channel-subsystem timer that is used by the channel-subsystem-monitoring facilities is in the error state, the state is reset. This happens independent of the setting of the two measurement-mode-control bits. (See "Channel-Subsystem Timing" on page 17-2 for a description of the timing facilities.)

**Associated Functions**

When the measurement-block-update facility is signaled (by means of SET CHANNEL MONITOR) to make the measurement-block-update mode active, the functions that are performed by the facility depend on whether or not the mode is already active when the signal is generated.

If the measurement-block-update mode is inactive when the signal is generated, the mode remains inactive until the measurement-block-update facility recognizes the signal. When the measurement-block-update facility recognizes the signal, the measurement-block-update mode is made active, and the MBK that was passed when the signal was generated is used to access all measurement blocks, and the MBO that was passed when the signal was generated is used to determine the address of format-0 measurement blocks.

If the measurement-block-update mode is active when the signal is generated, the mode remains active, and the MBK and MBO associated with the execution of a previous SET CHANNEL MONITOR instruction continue to be used to control the storing of measurement data until the measurement-block-update facility recognizes the signal. When the measurement-block-update facility recognizes the signal, the MBK and MBO associated with that signal are used instead of the MBK and MBO associated with the execution of a previous SET CHANNEL MONITOR instruction. The SET CHANNEL MONITOR instruction does not affect the measurement-block address used for format-1 measurement blocks, but the MBK associated with the signal becomes the key used to access the measurement block.

In all above cases, the measurement-block-update facility recognizes the signal during, or subsequent to, the execution of the SET CHANNEL MONITOR instruction that caused the signal to be generated and logically prior to the performance of any start function that is initiated by the subsequent execution of START SUBCHANNEL for a subchannel that is enabled for measurement by this facility. If


LzLabs GmbH. Ex. 1008-653

a subchannel that is enabled for measurement by this facility already has a start function in progress when the signal is generated, it is unpredictable when measurement data for that subchannel is stored by using the MBK and MBO associated with that signal.

While the measurement-block-update mode is active, performance measurements are accumulated for subchannels that are enabled for measurement-block update. Measurements for a subchannel are either accumulated in a single 32-byte format-0 measurement block within the measurement-block area, or a 64-byte format-1 measurement block pointed to by the measurement-block address at the subchannel. A subchannel is enabled for the measurement-block-update mode by setting the measurement-block-update-enable bit to one in the SCHIB and then executing the MODIFY SUBCHANNEL instruction for that subchannel. The measurement-block-format-control bit (F) at the subchannel specifies whether a format-0 or format-1 measurement block is stored for a subchannel when the measurement-block-update mode is active and the subchannel is enabled for measurement-block updates. When the F bit is zero, the MBO and MBI are used to determine the address of the measurement block for the subchannel, and a format-0 measurement block is stored. When the F bit is one, the measurement-block-address field at the subchannel contains the address of the measurement block for the subchannel, and a format-1 measurement block is stored. The F bit and measurement-block-address field are modified using the MODIFY SUBCHANNEL instruction.

When the device-connect-time-measurement mode is active, measurements of the length of time that the device is actively communicating with the channel subsystem during the execution of a channel program are accumulated for subchannels that are enabled for device-connect-time measurement. Measurements for a subchannel are provided in the extended-status word ESW of the IRB. A subchannel is enabled for device-connect-time-measurement mode by setting the device-connect-time-measurement-enable bit to one in the SCHIB and then executing MODIFY SUBCHANNEL for that subchannel.

For a more detailed description of the measurement-block-update mode, the format and contents of the measurement block, and the device-connect-time-measurement mode, see "Channel-Subsystem Monitoring" on page 17-1.

**Special Conditions**

SET CHANNEL MONITOR can encounter the program exceptions described or listed below.

Bits 4-29 of general register 1 must be zeros. When bit 30 (M) of general register 1 is one, bit 0 of general register 2 must be zero, and the MBO address in general register 2 must be designated on a 32-byte boundary. Otherwise, an operand exception is recognized.

***Condition Code:*** The code remains unchanged.

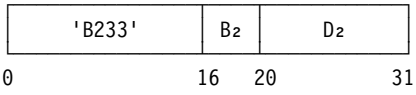
***Program Exceptions:***

- Operand
- Privileged operation

**Programming Note:** When the channel subsystem is initialized, the measurement-block-update and device-connect-time-measurement modes are made inactive.

## START SUBCHANNEL

SSCH $D_2(B_2)$ [S]




The channel subsystem is signaled to asynchronously perform the start function for the associated device, and the execution parameters that are contained in the designated ORB are placed at the designated subchannel. (See "Operation-Request Block" on page 15-22.)

General register 1 contains a subsystem-identification word that designates the subchannel to be started. The second-operand address is the logical address of the ORB and must be designated on a word boundary; otherwise, a specification exception is recognized.

The execution parameters contained in the ORB are placed at the subchannel.

When START SUBCHANNEL is executed, the subchannel is status pending with only secondary



LzLabs GmbH. Ex. 1008-654

status, and the extended-status-word-format bit (L) is zero, the status-pending condition is discarded at the subchannel.

The subchannel is made start pending, and the start function is indicated at the subchannel.

Logically prior to the setting of condition code 0, path-not-operational conditions at the subchannel, if any, are cleared.

The channel subsystem is signaled to asynchronously perform the start function. The start function is summarized below in the section "Associated Functions" and is described in detail in "Start Function and Resume Function" on page 15-18.

Condition code 0 is set to indicate that the actions described above have been taken.

**Associated Functions**

Subsequent to the execution of START SUBCHANNEL, the channel subsystem asynchronously performs the start function.

The contents of the ORB, other than the fields that must contain all zeros, are checked for validity. On some models, the fields of the ORB that must contain zeros are checked asynchronously, instead of during the execution of the instruction. When invalid fields are detected asynchronously, the subchannel becomes status pending with primary, secondary, and alert status and with deferred condition code 1 and program check indicated. (See "Program Check" on page 16-24.) In this situation, the I/O operation or chain of I/O operations is not initiated at the device, and the condition is indicated by the start-pending bit being stored as one when the SCSW is cleared by the execution of TEST SUBCHANNEL. (See "Subchannel-Status Word" on page 16-6).

On some models, path availability is tested asynchronously, instead of during the execution of the instruction. When no channel path is available for selection, the subchannel becomes status pending with primary and secondary status and with deferred condition code 3 indicated. The I/O operation or chain of I/O operations is not initiated at the device, and this condition is indicated by the start-pending bit being stored as one when the SCSW is cleared by the execution of TEST SUBCHANNEL.

If conditions allow, a channel path is chosen, and execution of the channel program that is designated in the ORB is initiated. (See "Start Function and Resume Function" on page 15-18.)

**Special Conditions**

***Condition code 1*** is set, and no other action is taken, when the subchannel is status pending when START SUBCHANNEL is executed. On some models, condition code 1 is not set when the subchannel is status pending with only secondary status; instead, the status-pending condition is discarded.

***Condition code 2*** is set, and no other action is taken, when a start, halt, or clear function is currently in progress at the subchannel (see "Function Control (FC)" on page 16-12).

***Condition code 3*** is set, and no other action is taken, when the subchannel is not operational for START SUBCHANNEL. A subchannel is not operational for START SUBCHANNEL if the subchannel is not provided in the channel subsystem, has no valid device number associated with it, or is not enabled.

A subchannel is also not operational for START SUBCHANNEL, on some models, when no channel path is available for selection. On these models, the lack of an available channel path is detected as part of the START SUBCHANNEL execution. On other models, channel-path availability is only tested as part of the asynchronous start function.

START SUBCHANNEL can encounter the program exceptions described or listed below.

In word 1 of the ORB, bits 13 and 25-30 must be zeros, and, in word 2 of the ORB, bit 0 must be zero. On models without the FICON-channel facility installed, bits 5-7 of word 1 must be zeros. On models without the z/Architecture architectural mode installed, bits 14 and 15 of word 1 must be zeros. On models without the ORB-extension facility installed, bit 31 of word 1 must be zero. Otherwise, on some models, an operand exception is recognized. On other models, an I/O-interruption condition is generated, indicating program check, as part of the asynchronous start function.


LzLabs GmbH. Ex. 1008-655

START SUBCHANNEL can also encounter the program exceptions listed below.

Bits 0-15 of the SID must contain 0001 hex; otherwise, an operand exception is recognized.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The execution of START SUBCHANNEL is suppressed on all addressing and protection exceptions.

***Resulting Condition Code:***

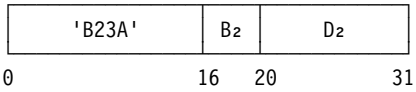
0 Function initiated
1 Status pending
2 Busy
3 Not operational

***Program Exceptions:***

- Access (fetch, operand 2)
- Operand
- Privileged operation
- Specification

## STORE CHANNEL PATH STATUS

STCPS $D_2(B_2)$ [S]




Depending on the model, this instruction may not be provided. When this instruction is not provided, it is checked for privileged operation exception and the instruction is suppressed by the machine.

A channel-path-status word of up to 256 bits is stored at the designated location.

The second-operand address is the logical address of the location where the channel-path-status word is to be stored and must be designated on a 32-byte boundary; otherwise, a specification exception is recognized.

The channel-path-status word indicates which channel paths are actively communicating with a device at the time STORE CHANNEL PATH STATUS is executed. Bit positions 0-255 correspond, respectively, to the channel paths having the channel-path identifiers 0-255. Each of the 256 bits at the designated location is set to one, set to zero, or left unchanged, as follows:

- For all channel paths in the configuration that are actively communicating with devices at the time STORE CHANNEL PATH STATUS is executed, the corresponding bits are stored as ones.
- For all channel paths that are (1) provided in the system (PIM bit in the PMCW is one) and (2) in the configuration but not currently being used by the channel subsystem in actively communicating with devices, the corresponding bits are stored as zeros.
- For all channel paths that are not provided in the system (PIM bit in the PMCW is zero), the corresponding bits either are not stored or are stored as zeros.
- For all channel paths in the configuration that are in the channel-path-terminal state or are not physically available (the corresponding PAM bit in the PMCW is zero), the corresponding bits are stored as zeros.

**Special Conditions**

STORE CHANNEL PATH STATUS can encounter the program exceptions described or listed below.

The second operand must be designated on a 32-byte boundary; otherwise, a specification exception is recognized.

The execution of STORE CHANNEL PATH STATUS is suppressed on all addressing and protection exceptions.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (store, operand 2)
- Privileged operation
- Specification

**Programming Notes:**

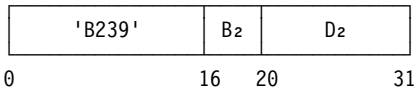
1. To ensure a consistent interpretation of channel-path-status-word bits, the program should, prior to the initial use of the area, store zeros at the location where the channel-path-status word is to be stored.



LzLabs GmbH. Ex. 1008-656

## STORE CHANNEL REPORT WORD

STCRW $D_2(B_2)$ [S]




A CRW containing information affecting the channel subsystem is stored at the designated location.

The second-operand address is the logical address of the location where the CRW is to be stored and must be designated on a word boundary; otherwise, a specification exception is recognized.

When a malfunction or other condition affecting channel-subsystem operation is recognized, a channel report (consisting of one or more CRWs) describing the condition is made pending for retrieval and analysis by the program. The channel report contains information concerning the identity and state of a facility following the detection of the malfunction or other condition. For a description of the channel report, the CRW, and program-recovery actions related to the channel subsystem, see "Channel-Subsystem Recovery" on page 17-21.

When one or more channel reports are pending, the instruction causes a CRW to be stored at the designated location and condition code 0 to be set. A pending CRW can only be stored by the execution of STORE CHANNEL REPORT WORD and, once stored, is no longer pending. Thus, each pending CRW is presented only once to the program.

When no channel reports are pending in the channel subsystem execution of STORE CHANNEL REPORT WORD causes zeros to be stored at the designated location and condition code 1 to be set.

**Special Conditions**

STORE CHANNEL REPORT WORD can encounter the program exceptions described or listed below.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The execution of STORE CHANNEL REPORT WORD is suppressed on all addressing and protection exceptions.

***Resulting Condition Code:***

0 CRW stored
1 Zeros stored
2 --
3 --

***Program Exceptions:***

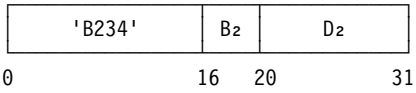
- Access (store, operand 2)
- Privileged operation
- Specification

**Programming Notes:**

1. CRW overflow conditions may occur if STORE CHANNEL REPORT WORD is not executed to clear pending channel reports. If the overflow condition is encountered, one or more channel-report words have been lost. (See "Channel-Subsystem Recovery" on page 17-21 for details.)
2. A pending CRW can be cleared by any CPU in the configuration executing STORE CHANNEL REPORT WORD, regardless of whether a machine-check interruption has occurred in any CPU.

## STORE SUBCHANNEL

STSCH $D_2(B_2)$ [S]




Control and status information for the designated subchannel is stored in the designated SCHIB.

General register 1 contains a subsystem-identification word that designates the subchannel for which the information is to be stored. The second-operand address is the logical address of the SCHIB and must be designated on a word boundary; otherwise, a specification exception is recognized.


LzLabs GmbH. Ex. 1008-657

When the extended-I/O-measurement-block facility is not installed, the information that is stored in the SCHIB consists of a path-management-control word, a SCSW, and three words of model-dependent information. When the extended-I/O-measurement-block facility is installed, the information that is stored in the SCHIB consists of a path-management-control word, a SCSW, the measurement-block-address field, and one word of model-dependent information. (See "Subchannel-Information Block" on page 15-1.)

The execution of STORE SUBCHANNEL does not change any information at the subchannel.

Condition code 0 is set to indicate that control and status information for the designated subchannel has been stored in the SCHIB. When the execution of STORE SUBCHANNEL results in the setting of condition code 0, the information in the SCHIB indicates a consistent state of the subchannel.

**Special Conditions**

***Condition code 3*** is set, and no other action is taken, when the designated subchannel is not operational for STORE SUBCHANNEL. A subchannel is not operational for STORE SUBCHANNEL if the subchannel is not provided in the channel subsystem.

STORE SUBCHANNEL can encounter the program exceptions described or listed below.

Bits 0-15 of the SID must contain 0001 hex; otherwise, an operand exception is recognized.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 SCHIB stored
1 --
2 --
3 Not operational

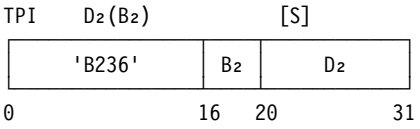
***Program Exceptions:***

- Access (store, operand 2)
- Operand
- Privileged operation
- Specification

**Programming Notes:**

1. Device status that is stored in the SCSW may include device-busy, control-unit-busy, or control-unit-end indications.
2. The information that is stored in the SCHIB is obtained from the subchannel. The STORE SUBCHANNEL instruction does not cause the channel subsystem to interrogate the addressed device.
3. STORE SUBCHANNEL may be executed at any time to sample conditions existing at the subchannel, without causing any pending status conditions to be cleared.
4. Repeated execution of STORE SUBCHANNEL without an intervening delay (for example, to determine when a subchannel changes state) should be avoided because repeated accesses of the subchannel by the CPU may delay or prohibit access of the subchannel by a channel subsystem to update the subchannel.

# TEST PENDING INTERRUPTION




The I/O-interruption code for a pending I/O interruption at a subchannel is stored at the location designated by the second-operand address, and the pending I/O-interruption request is cleared.

The second-operand address, when nonzero, is the logical address of the location where the two-word I/O-interruption code, consisting of words 0 and 1, is to be stored. The second-operand address must be designated on a word boundary; otherwise, a specification exception is recognized.

If the second-operand address is zero, the three-word I/O-interruption code, consisting of words 0-2, is stored at real locations 184-195. In this case, low-address protection and key-controlled protection do not apply.

In the access-register mode when the second-operand address is zero, it is unpredictable whether access-register translation occurs for access register $B_2$. If the translation occurs, the



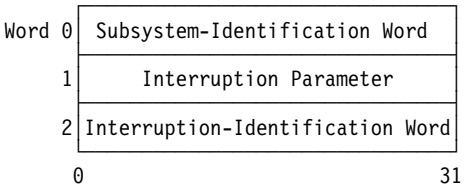
LzLabs GmbH. Ex. 1008-658

resulting segment-table designation is not used; that is, the interruption code still is stored at real locations 184-195.

Pending I/O-interruption requests are accepted only for those I/O-interruption subclasses allowed by the I/O-interruption-subclass mask in control register 6 of the CPU executing the instruction. If no I/O-interruption requests exist that are allowed by control register 6, the I/O-interruption code is not stored, the second-operand location is not modified, and condition code 0 is set.

If a pending I/O-interruption request is accepted, the I/O-interruption code is stored, the pending I/O-interruption request is cleared, and condition code 1 is set. The I/O-interruption code that is stored is the same as would be stored if an I/O interruption had occurred. However, PSWs are not swapped as when an I/O-interruption occurs.

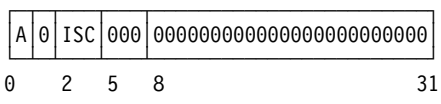
The I/O-interruption code that is stored during the execution of the instruction is defined as follows:




***Subsystem-Identification Word (SID):*** See "I/O-Instruction Formats" in Chapter 14.

***Interruption Parameter:*** Word 1 contains a four-byte parameter that was specified by the program and passed to the subchannel in word 0 of the ORB or the PMCW. When a device presents alert status and the interruption parameter was not previously passed to the subchannel by an execution of START SUBCHANNEL or MODIFY SUBCHANNEL, this field contains zeros.

***Interruption-Identification Word:*** Word 2, when stored, contains the interruption-identification word, which further identifies the source of the I/O-interruption. Word 2 is stored only when the second-operand address is zero. The interruption-identification word is defined as follows:




***A bit (A):*** Bit 0 of the interruption-identification word specifies the type of pending I/O-interruption request that was cleared. When bit 0 is zero, the I/O-interruption request was associated with a subchannel.

***I/O-Interruption Subclass (ISC):*** Bit positions 2-4 of the interruption-identification word contain an unsigned binary integer, in the range 0-7, that specifies the I/O-interruption subclass associated with the subchannel for which the pending I/O-interruption request was cleared.

The remaining bit positions are reserved and stored as zeros.

**Special Conditions**

TEST PENDING INTERRUPTION can encounter the program exceptions described or listed below.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The execution of TEST PENDING INTERRUPTION is suppressed on all addressing and protection exceptions.

***Resulting Condition Code:***

0 Interruption code not stored
1 Interruption code stored
2 --
3 --

***Program Exceptions:***

- Access (store, operand 2, second-operand address nonzero only)
- Privileged operation
- Specification

**Programming Notes:**

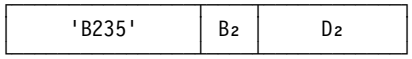
1. TEST PENDING INTERRUPTION should only be executed with a second-operand address of zero when I/O interruptions are masked off. Otherwise, an I/O-interruption code stored by the instruction may be lost if an I/O interruption occurs. The I/O-interruption code that identifies the source of an I/O interruption taken subsequent to TEST PENDING INTERRUPTION is also stored at real locations 184-195, replacing an I/O-interruption code that was stored by the instruction.


LzLabs GmbH. Ex. 1008-659

2. In the access-register mode when the second-operand address is zero, an access exception is recognized if access-register translation occurs and the access register is in error. This exception can be prevented by making the $B_2$ field zero or by placing 00000000 hex, 00000001 hex, or any other valid contents in the access register.

## TEST SUBCHANNEL

TSCH $D_2(B_2)$ [S]




Control and status information for the subchannel is stored in the designated IRB.

General register 1 contains a subsystem-identification word that designates the subchannel for which the information is to be stored. The second-operand address is the logical address of the IRB and must be designated on a word boundary; otherwise, a specification exception is recognized.

The information that is stored in the IRB consists of a SCSW, an extended-status word, and an extended-control word. (See "Interruption-Response Block" on page 16-6.)

If the subchannel is status pending, the status-pending bit of the status-control field is stored as one. Whether or not the subchannel is status pending has an effect on the functions that are performed when TEST SUBCHANNEL is executed.

When the subchannel is status pending and TEST SUBCHANNEL is executed, information, as described above, is stored in the IRB, followed by the clearing of certain conditions and indications that exist at the subchannel as described in Figure 14-2. If an I/O-interruption request is pending for the subchannel, the request is cleared. Condition code 0 is set to indicate that these actions have been taken.

When the subchannel is not status pending and TEST SUBCHANNEL is executed, information (as described above) is stored in the IRB, and no conditions or indications are cleared. Condition code 1 is set to indicate that these actions have been taken.

Figure 14-2 describes which conditions and indications are cleared by TEST SUBCHANNEL when the subchannel is status pending. All other conditions and indications at the subchannel remain unchanged.

| | Subchannel Condition* | | | | |
|---|---|---|---|---|---|
| Field | Alert Status Pdg | Int Status Pdg | Pri Status Pdg | Sec Status Pdg | Status Pdg Alone |
| Function Control | C | Nc | C | C | C |
| Activity Control | Cp | Nr | Cp | Cp | Cp |
| Status Control | Cs | Cs | Cs | Cs | Cs |
| N condition | C | Nr | C | C | C |

**Explanation:**

* Note that the rightmost column applies to status pending when it is alone. The other four status-pending conditions result in the clearing actions given. These actions apply both when a single status-pending condition occurs and when a combination of the four status-pending conditions occurs. In the combination case, all the clearing actions of the individual cases apply.
C Cleared.
Cp The resume-, start-, halt-, clear pending, and suspended conditions are cleared.
Cs The status-pending condition is cleared.
Nc Not changed unless function control indicates the halt function and activity control indicates suspended. If both the halt function and suspended are indicated, conditions are cleared as for status pending alone.
Nr Not changed unless activity control indicates suspended and function control indicates the start function with or without the halt function. If the halt function is indicated, the conditions are cleared as for status pending alone. If only the start function is indicated, the resume-pending condition and the N condition are cleared.

*Figure 14-2. Conditions and Indications Cleared at the Subchannel by TEST SUBCHANNEL*



LzLabs GmbH. Ex. 1008-660

**Special Conditions**

***Condition code 3*** is set, and no other action is taken, when the subchannel is not operational for TEST SUBCHANNEL. A subchannel is not operational for TEST SUBCHANNEL if the subchannel is not provided, has no valid device number associated with it, or is not enabled.

TEST SUBCHANNEL can encounter the program exceptions described or listed below.

Bits 0-15 of the SID must contain 0001 hex; otherwise, an operand exception is recognized.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

When the execution of TEST SUBCHANNEL is terminated on addressing and protection exceptions, the state of the subchannel is not changed.

***Resulting Condition Code:***

0 IRB stored; subchannel status pending
1 IRB stored; subchannel not status pending
2 --
3 Not operational

***Program Exceptions:***

- Access (store, operand 2)
- Operand
- Privileged operation
- Specification

**Programming Notes:**

1. Device status that is stored in the SCSW may include device-busy, control-unit-busy, or control-unit-end indications.
2. The information that is stored in the IRB is obtained from the subchannel. The TEST SUBCHANNEL instruction does not cause the channel subsystem to interrogate the addressed device.
3. When an I/O interruption occurs, it is the result of a status-pending condition at the subchannel, and typically TEST SUBCHANNEL is executed to clear the status. TEST SUBCHANNEL may also be executed at any other time to sample conditions existing at the subchannel.
4. Repeated execution of TEST SUBCHANNEL to determine when a start function has been completed should be avoided because there are conditions under which the completion of the start function may or may not be indicated. For example, if the channel subsystem is holding an interface-control-check (IFCC) condition in abeyance (for any subchannel) because another subchannel is already status pending, and if the start function being tested by TEST SUBCHANNEL has as the only path available for selection the channel path with the IFCC condition, then the start function may not be initiated until the status-pending condition in the other subchannel is cleared, allowing the IFCC condition to be indicated at the subchannel to which it applies.
5. Repeated execution of TEST SUBCHANNEL without an intervening delay, for example, to determine when a subchannel changes state, should be avoided because repeated accesses of the subchannel by the CPU may delay or prohibit accessing of the subchannel by the channel subsystem. Execution of TEST SUBCHANNEL by multiple CPUs for the same subchannel at approximately the same time may have the same effect and also should be avoided.
6. The priority of I/O-interruption handling by a CPU can be modified by the execution of TEST SUBCHANNEL. When TEST SUBCHANNEL is executed and the designated subchannel has an I/O-interruption request pending, that I/O-interruption request is cleared, and the SCSW is stored, without regard to any previously established priority. The relative priority of the remaining I/O-interruption requests is unchanged.


LzLabs GmbH. Ex. 1008-661

LzLabs GmbH. Ex. 1008-662

# Chapter 15. Basic I/O Functions


Control of Basic I/O Functions . . . . . 15-1
Subchannel-Information Block . . . . . 15-1
Path-Management-Control Word . . . . 15-2
Subchannel-Status Word . . . . . 15-8
Model-Dependent Area/Measurement Block Address . . . . . 15-8
Summary of Modifiable Fields . . . . . 15-9
Channel-Path Allegiance . . . . . 15-11
Working Allegiance . . . . . 15-12
Active Allegiance . . . . . 15-12
Dedicated Allegiance . . . . . 15-12
Channel-Path Availability . . . . . 15-13
Control-Unit Type . . . . . 15-13
Clear Function . . . . . 15-14
Clear-Function Path Management . . . . 15-14
Clear-Function Subchannel Modification . 15-14
Clear-Function Signaling and Completion . . . . . 15-15
Halt Function . . . . . 15-15
Halt-Function Path Management . . . . . 15-16
Halt-Function Signaling and Completion . 15-16
Start Function and Resume Function . . . . 15-18
Start-Function and Resume-Function Path Management . . . . . 15-19
Execution of I/O Operations . . . . . 15-21
Blocking of Data . . . . . 15-22
Operation-Request Block . . . . . 15-22
Channel-Command Word . . . . . 15-27
Command Code . . . . . 15-29
Designation of Storage Area . . . . . 15-30
Chaining . . . . . 15-31
Data Chaining . . . . . 15-33
Command Chaining . . . . . 15-34
Skipping . . . . . 15-35
Program-Controlled Interruption . . . . . 15-35
CCW Indirect Data Addressing . . . . . 15-36
Suspension of Channel-Program Execution . . . . . 15-38
Commands and Flags . . . . . 15-40
Branching in Channel Programs . . . . . 15-41
Transfer in Channel . . . . . 15-41
Command Retry . . . . . 15-42
Concluding I/O Operations before Initiation . 15-42
Concluding I/O Operations during Initiation . 15-42
Immediate Conclusion of I/O Operations . . 15-43
Concluding I/O Operations during Data Transfer . . . . . 15-43
Channel-Path-Reset Function . . . . . 15-45
Channel-Path-Reset-Function Signaling . 15-45
Channel-Path-Reset-Function-Completion Signaling . . . . . 15-45


Some I/O instructions specify to the channel subsystem that a function is to be performed. Collectively, these functions are referred to as the basic I/O functions. The basic I/O functions are the clear, halt, start, resume, and channel-path-reset functions.

## Control of Basic I/O Functions

Information that is present at the subchannel controls how the clear, halt, resume, and start functions are performed. This information is communicated to the program in the subchannel-information block during the execution of STORE SUBCHANNEL.

## Subchannel-Information Block

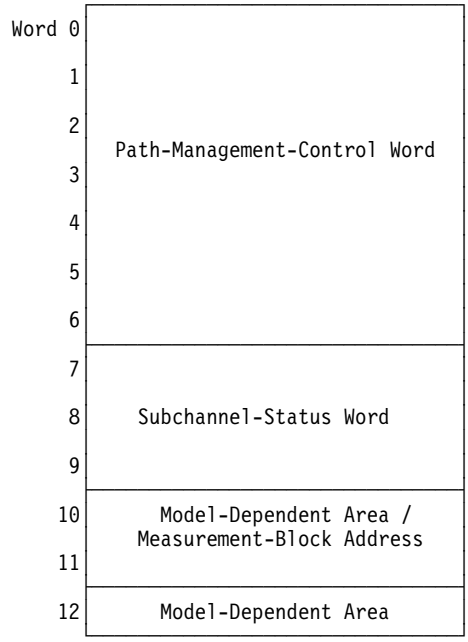
The subchannel-information block (SCHIB) is the operand of the MODIFY SUBCHANNEL and STORE SUBCHANNEL instructions. The two rightmost bits of the SCHIB address are zeros, designating the SCHIB on a word boundary. The SCHIB contains three major fields: the path-management-control word (PMCW), the subchannel-status word (SCSW), and a model-dependent area. When the extended-I/O-measurement-block facility is installed the SCHIB also contains the measurement-block-address field. (Figure 15-1 on page 15-2 shows the format of the PMCW, and Figure 16-2 on page 16-7 shows the format of the SCSW.)

STORE SUBCHANNEL is used to store the current PMCW, the SCSW, model-dependent data, and, when the extended-I/O-measurement-block facility is

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-663

installed, the measurement-block-address field, for the designated subchannel. MODIFY SUBCHANNEL alters certain PMCW fields and, when the extended-I/O-measurement-block facility is installed, the measurement-block address in the subchannel. When the program needs to change the contents of one or more of the PMCW fields, the normal procedure is to (1) issue STORE SUBCHANNEL to obtain the current contents, (2) perform the required modifications to the PMCW field or the measurement-block-address field in main storage, and (3) issue MODIFY SUBCHANNEL to pass the new information to the subchannel. The SCHIB has the following format:




## Path-Management-Control Word

Words 0-6 of the SCHIB contain the path-management-control word (PMCW). The PMCW has the format shown in Figure 15-1 when the subchannel is valid (see “Device Number Valid (V)” on page 15-4).

The format of the PMCW is as follows:

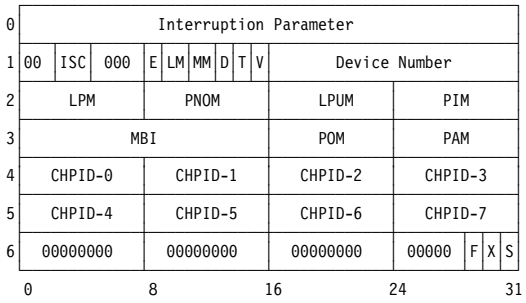



*Figure 15-1. PMCW Format*

***Interruption Parameter:*** Bit positions 0-31 of word 0 contain the interruption parameter that is stored as word 1 of the interruption code. The interruption parameter can be set to any value by START SUBCHANNEL and MODIFY SUBCHANNEL. The initial value of the interruption parameter is zero.

***I/O-Interruption-Subclass Code (ISC):*** Bits 2-4 of word 1 are an unsigned binary integer, in the range 0-7, that corresponds to the bit position of the I/O-interruption subclass-mask bit in control register 6 of each CPU in the configuration. The setting of that mask bit in control register 6 of a CPU controls the recognition of interruption requests relating to this subchannel by that CPU (see “Priority of Interruptions” on page 16-4). The ISC can be set to any value by MODIFY SUBCHANNEL. The initial value of the ISC is zero.

***Reserved:*** Bits 0, 1, and 5-7 of word 1 are reserved and stored as zeros by STORE SUBCHANNEL. Bits 0, 1, 6, and 7 must be zeros when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized. Bit 5 of word 1 is ignored when MODIFY SUBCHANNEL is executed.

***Enabled (E):*** Bit 8 of word 1, when one, indicates that the subchannel is enabled for all I/O functions. When the E bit is zero, status presented by the device is not made available to the program, and I/O instructions other than MODIFY SUBCHANNEL and STORE SUBCHANNEL that are executed for the designated subchannel cause condition code 3 to be set. The E bit can be either zero or one when MODIFY SUBCHANNEL is executed; initially, all subchannels are not enabled; IPL causes the IPL I/O device to become enabled.



LzLabs GmbH. Ex. 1008-664

***Limit Mode (LM):*** When the address-limit-checking facility is installed, bits 9 and 10 of word 1 define the limit mode (LM) of the subchannel. The limit mode is used by the channel subsystem when address-limit checking is invoked for an I/O operation. (See "Address-Limit Checking" on page 17-20.) Address-limit checking is under the control of the address-limit-checking-control bit that is passed to the subchannel in the operation-request block (ORB) during the execution of START SUBCHANNEL. (See "Address-Limit-Checking Control (A)" on page 15-25.) The definitions of the LM bits, whose values are used during data transfer, are as follows:

| Bit 9 | Bit 10 | Function |
|---|---|---|
| 0 | 0 | Initialized value. No limit checking is performed for this subchannel. |
| 0 | 1 | Data address must be equal to or greater than the current address limit. |
| 1 | 0 | Data address must be less than the current address limit. |
| 1 | 1 | Reserved. |

Bit positions 9 and 10 can contain any of the first three bit combinations shown above when MODIFY SUBCHANNEL is executed. Specification of the reserved bit combination in the operand causes an operand exception to be recognized when MODIFY SUBCHANNEL is executed.

When the address-limit-checking facility is not installed, bits 9-10 of word 1 may be set as described above; however, they are ignored and are not set at the specified subchannel. When bits 9-10 are not specified as described above and MODIFY SUBCHANNEL is executed, an operand exception is recognized.

***Measurement-Mode Enable (MM):*** Bits 11 and 12 of word 1 enable the measurement-block-update mode and the device-connect-time-measurement mode, respectively, of the subchannel. These bits can have any value when MODIFY SUBCHANNEL is executed; initially, neither measurement mode is enabled. The definition of each of these bits is as follows:

| Bit 11 | Measurement-Block-Update Enable: |
|---|---|
| 0 | Initialized value. The subchannel is not enabled for measurement-block update. Storing of measurement-block data does not occur. |
| 1 | The subchannel is enabled for measurement-block update. If the measurement-block-update mode is active, measurement data is accumulated in the measurement block at the time channel-program execution is completed or suspended at the subchannel or completed at the device, as appropriate, provided no error conditions described by subchannel logout have been detected. (See "Measurement-Block Update" on page 17-3.) If the measurement-block-update mode is inactive, no measurement-block data is stored. |

| Bit 12 | Device-Connect-Time-Measurement Enable: |
|---|---|
| 0 | Initialized value. The subchannel is not enabled for device-connect-time measurement. Storing of the device-connect-time interval (DCTI) in the extended-status word (ESW) does not occur. |
| 1 | The subchannel is enabled for device-connect-time measurement. If the device-connect-time-measurement mode is active and timing facilities are provided for the subchannel, the value of the DCTI is stored in the ESW when TEST SUBCHANNEL is executed after channel-program execution is completed or suspended at the subchannel, provided no error conditions described by subchannel logout have been detected. If the device-connect-time-measurement mode is inactive, no measurement values are stored in the ESW. |

The meaning of the measurement-mode-enable bits (MM), described above, applies when the timing-facility bit for the subchannel is one. When the timing-facility bit is zero, the effect of the MM bits is changed, as described below under "Timing Facility." (For more discussion on measurement modes, see "Measurement-Block Update" on page 17-3 and "Device-Connect-Time Measurement" on page 17-10.)

***Multipath Mode (D):*** Bit 13 of word 1, when one, indicates that the subchannel operates in the multipath mode when performing an I/O operation


LzLabs GmbH. Ex. 1008-665

or chain of I/O operations. For proper operation in the multipath mode when more than one channel path is available for selection, the associated device must have the dynamic-reconnection feature installed and must be set up for multipath-mode operation. During performance of a start function in the multipath mode, a device is allowed to request service from the channel subsystem over any of the channel paths indicated at the subchannel as being available for selection (see "Logical-Path Mask (LPM)" and "Path-Available Mask (PAM)" on page 15-7). Bit 13, when zero, indicates that the subchannel operates in single-path mode when performing an I/O operation or chain of I/O operations. In the single-path mode, the entire start function is performed by using the channel path on which the first command of the I/O operation or chain of I/O operations was accepted by the device. The D bit can be either zero or one when MODIFY SUBCHANNEL is executed; initially, the subchannel is in the single-path mode.

***Timing Facility (T):*** Bit 14 of word 1, when one, indicates that the channel-subsystem-timing facility is available for the subchannel and is under the control of the two measurement-mode-enable bits (MM) and SET CHANNEL MONITOR. Bit 14, when zero, indicates that the channel-subsystem-timing facility is not available for the subchannel. When bit 14 is zero, the START SUBCHANNEL count is the only measurement data that can be accumulated in the measurement block for the subchannel. Storing of the START SUBCHANNEL count is under the control of bit 11 and SET CHANNEL MONITOR, as described above under "Measurement Mode Enable." Similarly, if the T bit is zero, no device-connect-time-interval (DCTI) values can be measured for the subchannel. (See "Measurement-Block Update" on page 17-3 and "Device-Connect-Time Measurement" on page 17-10.)

***Device Number Valid (V):*** Bit 15 of word 1, when one, indicates that the device-number field (see below) contains a valid device number and that a device associated with this subchannel may be physically installed. Bit 15, when zero, indicates that the subchannel is not valid, there is no I/O device currently associated with the subchannel, and the contents of all other defined fields of the SCHIB are unpredictable.

***Device Number:*** Bit positions 16-31 of word 1 contain the binary representation of the four-digit hexadecimal device number of the device that is associated with this subchannel. The device number is a system-unique parameter that is assigned to the subchannel and the associated device when the device is installed.

***Logical-Path Mask (LPM):*** Bits 0-7 of word 2 indicate the logical availability of channel paths to the associated device. Each bit of the LPM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. A bit set to one means that the corresponding channel path is logically available; a zero means the corresponding channel path is logically not available. When a channel path is logically not available, the channel subsystem does not use that channel path to initiate performance of any clear, halt, resume, or start function, except when a dedicated allegiance exists for that channel path. When a dedicated allegiance exists at the subchannel for a channel path, the logical availability of the channel path is ignored whenever a clear, halt, resume, or start function is performed. (See "Channel-Path Allegiance" on page 15-11). If the subchannel is idle, the logical availability of the channel path is ignored whenever the control unit initiates a request to present alert status to the channel subsystem. The logical availability of a channel path associated with the subchannel can be changed by setting the corresponding LPM bit in the SCHIB and then issuing MODIFY SUBCHANNEL, or by setting the corresponding LPM bit in the ORB and then issuing START SUBCHANNEL. Initially, each installed channel path is logically available.

***Path-Not-Operational Mask (PNOM):*** Any of bits 8-15 of word 2, when one, indicates that a path-not-operational condition has been recognized on the corresponding channel path. Each bit of the PNOM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. The channel subsystem recognizes a path-not-operational condition when, during an attempted device selection in order to perform a clear, halt, resume, or start function, the device associated with the subchannel appears not operational on a channel path that is operational for the subchannel. When a path-not-operational condition is recognized, the state of the channel path changes from operational for the subchannel to not opera-



LzLabs GmbH. Ex. 1008-666

tional for the subchannel. A channel path is operational for the subchannel if the associated device appeared operational on that channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function. A device appears to be operational on a channel path when the device responds to an attempted device selection. A channel path is not operational for the subchannel if the associated device appeared not operational on that channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function. Any of bits 8-15 of word 2, when zero, indicates that a path-not-operational condition has not been recognized on the corresponding channel path.

Initially, each of the eight possible channel paths associated with each subchannel is considered to be operational, regardless of whether the respective channel paths are installed or available; therefore, unless a path-not-operational condition is recognized during initial program loading, the PMCW, if stored, contains a PNOM of all zeros if stored prior to the execution of a CLEAR SUBCHANNEL, HALT SUBCHANNEL, RESUME SUBCHANNEL, or START SUBCHANNEL instruction.

**Programming Note:** The PNOM indicates those channel paths for which a path-not-operational condition has been recognized during the performance of the most recent clear, halt, resume, or start function. That is, the PNOM indicates which of the channel paths associated with the subchannel have made a transition from the operational to the not-operational state for the subchannel during the performance of the most recent clear, halt, resume, or start function. However, the transition of a channel path from the not-operational to the operational state for the subchannel is indicated in the POM. Therefore, the POM must be examined in order to determine whether any of the channel paths that are associated with a designated subchannel are operational for the subchannel.

Furthermore, while performing either a start function or a resume function, the transition of a channel path from the not-operational to the operational state for the subchannel is recognized by the channel subsystem only during the initiation sequence for the first command specified by the start function or implied by the resume function. Therefore, a channel path that is currently not operational for the subchannel can be used by the device associated with the subchannel when reconnecting to the channel subsystem in order to continue command chaining; however, the channel subsystem does not indicate a transition of that channel path from the not-operational to the operational state for the subchannel in the POM.

| POM Value and Device State before Selection Attempt | | Value of Specified Bit Subsequent to Selection Attempt | | |
|---|---|---|---|---|
| Device State[1] | POM | POM | PNOM[2] | SCSW N Bit |
| OP | 0 | 1 | 0 | 0 |
| NOP | 0 | 0 | 0 | 0 |
| OP | 1 | 1 | 0 | 0 |
| NOP | 1 | 0 | 1 | 1[3] |

**Explanation:**

1 Device state as it appears on the corresponding channel path.

2 Prior to the attempted device selection during the performance of either a start function or a resume function while the subchannel is suspended, the channel subsystem clears all existing path-not-operational conditions, if any, at the designated subchannel.

3 The N bit (bit 15 of word 0 of the SCSW) is indicated to the program and the N condition is cleared at the subchannel when TEST SUBCHANNEL is executed the next time the subchannel is status pending for other than intermediate status alone provided that it is not also suspended.

NOP The device is not operational on the corresponding channel path.

OP The device is operational on the corresponding channel path.

*Figure 15-2. Resulting POM, PNOM, and N-Bit Values Subsequent to Selection Attempt*

***Last-Path-Used Mask (LPUM):*** Bits 16-23 of word 2 indicate the channel path that was last used for communicating or transferring information between the channel subsystem and the device. Each bit of the LPUM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. Each bit of the LPUM is stored as zero, except for the bit that corresponds to the channel path last used, whenever one of the following occurs:


LzLabs GmbH. Ex. 1008-667

1. The first command of a start or resume function is accepted by the device (see "Activity Control (AC)" on page 16-13).
2. The device and channel subsystem are actively communicating when the suspend function is performed for the channel program in execution.
3. Status has been accepted from the device and is recognized as an interruption condition, or a condition has been recognized that suppresses command chaining (see "Interruption Conditions" on page 16-2).
4. An interface-control-check condition has been recognized (see "Interface-Control Check" on page 16-28), and no subchannel-logout information is currently present in the subchannel.

The LPUM field of the PMCW contains the most recent setting. The initial value of the LPUM is zero.

***Path-Installed Mask (PIM):*** Bits 24-31 of word 2 indicate which of the channel paths 0-7 to the I/O device are physically installed. The PIM indicates the validity of the channel-path identifiers (see below) for those channel paths that are physically installed. Each bit of the PIM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. A PIM bit stored as one indicates that the corresponding channel path is installed. A PIM bit stored as zero indicates that the corresponding channel path is not installed. The PIM always reflects the full complement of installed paths to the device, regardless of how the system is configured. Therefore, some of the channel paths indicated in the PIM may not be physically available in that configuration, as indicated by the bit settings in the path-available mask (see below). The initial value of the PIM indicates all the physically installed channel paths to the device.

***Measurement-Block Index (MBI):*** Bits 0-15 of word 3 form an index value used by the measurement-block-update facility when the measurement-block-update mode is active (see "SET CHANNEL MONITOR" on page 14-12) and the subchannel is enabled for the mode (see "Measurement-Mode Enable (MM)" on page 15-3). When the measurement-block index is used, five zero bits are appended on the right, and the result is added to the measurement-block-origin address designated by SET CHANNEL MONITOR. The calculated address, called the measurement-block address, designates the beginning of a 32-byte storage area where measurement data is stored. (See "Measurement Block" on page 17-3.) The MBI can contain any value when MODIFY SUBCHANNEL is executed; the initial value is zero.

***Path-Operational Mask (POM):*** Bits 16-23 of word 3 indicate the last known operational state of the device on the corresponding channel paths. Each bit of the POM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. If the associated device appeared operational on a channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function, then the channel path is operational for the subchannel, and the bit corresponding to the channel path in the POM is one. A device appears to be operational on a channel path when the device responds to an attempted device selection. A channel path is also operational for the subchannel if MODIFY SUBCHANNEL is executed and the bit corresponding to that channel path in the POM is specified as one.

If the associated device appeared not operational on a channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function, then the channel path is not operational for the subchannel, and the bit corresponding to the channel path in the POM is zero. A channel path is also not operational for the subchannel if MODIFY SUBCHANNEL is executed and the bit corresponding to that channel path in the POM is specified as zero.

If the device associated with the subchannel appears not operational on a channel path that is operational for the subchannel during an attempted device selection in order to perform a clear, halt, resume, or start function, then the channel subsystem recognizes a path-not-operational condition. If an SCSW is subsequently stored, then bit 15 of word 0 is one, indicating the path-not-operational condition. When a path-not-operational condition is recognized, the state of the channel path changes from operational for the subchannel to not operational for the subchannel.



LzLabs GmbH. Ex. 1008-668

When the channel path is not operational for the subchannel, a path-not-operational condition cannot be recognized. Moreover, a channel path that is not operational for the subchannel may be available for selection; if the channel subsystem chooses that channel path while performing a path-management operation, and if, during the attempted device selection, the device appears to be operational again on that channel path, then the state of the channel path changes from not operational for the subchannel to operational for the subchannel.

The POM can contain any value when MODIFY SUBCHANNEL is executed. Initially, each of the eight possible channel paths associated with each subchannel is considered to be operational, regardless of whether the respective channel paths are installed or available; therefore, unless a path-not-operational condition is recognized during initial program loading, the PMCW, if stored, contains a POM of all ones if stored prior to the execution of a CLEAR SUBCHANNEL, HALT SUBCHANNEL, RESUME SUBCHANNEL, or START SUBCHANNEL instruction.

***Path-Available Mask (PAM):*** Bits 24-31 of word 3 indicate the physical availability of installed channel paths. Each bit of the PAM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. A PAM bit of one indicates that the corresponding channel path is physically available for use in accessing the device. A PAM bit of zero indicates the channel path is not physically available for use in accessing the device. When a channel path is not physically available, it may, depending on the model and the extent of failure, be used during performance of the reset-channel-path function. A channel path that is physically available may become not physically available as a result of reconfiguring the system, or this may occur as a result of the performance of the channel-path-reset function. The initial value of the PAM reflects the set of channel paths by which the I/O device is physically accessible at the time of initialization.

**Note:** The change in the availability of a channel path affects all subchannels having access to that channel path. Whenever the setting of a PAM bit is referred to in conjunction with the availability status of a channel path, for brevity, reference is made in this chapter to a single PAM bit instead of to the respective PAM bits in all of the affected subchannels.

***Channel-Path Identifiers (CHPIDs):*** Words 4 and 5 contain eight one-byte channel-path identifiers corresponding to channel paths 0-7 of the PIM. A CHPID is valid if the corresponding PIM bit is one. Each valid CHPID contains the identifier of a physical channel path to a control unit by which the associated I/O device may be accessed. A unique CHPID is assigned to each physical channel path in the system.

Different devices that are accessible by the same physical channel path have, in their respective subchannels, the same CHPID value. The CHPID value may, however, appear in each subchannel in different locations in the CHPID fields 0-7.

Subchannels that share an identical set of channel paths have the same corresponding PIM bits set to ones. The channel-path identifiers (CHPIDs) for these channel paths are the same and occupy the same respective locations in each SCHIB.

***Reserved:*** Bits 0-28 of word 6 are reserved and are stored as zeros by STORE SUBCHANNEL. They must be zeros when MODIFY SUBCHANNEL is executed; otherwise, an operand exception may be recognized.

***Measurement Block Format Control (F):*** when the extended-measurement-block facility is installed, bit 29 of word 6 specifies the format of the measurement block to be stored when the subchannel is enabled for the measurement-block-update mode, and measurement-block-update mode is active. The bit can contain any value when MODIFY SUBCHANNEL is executed. The initial value is zero. The definition of the bit is as follows:


LzLabs GmbH. Ex. 1008-669

| Bit 29 | Measurement-Block-Format Control: |
|---|---|
| 0 | Format-0 measurement block. Specifies that a format-0 measurement block is used when performing a measurement-block update for the subchannel. The address of the 32-byte measurement block is obtained using the MBI provided by MSCH in conjunction with the MBO provided by SCHM. |
| 1 | Format-1 measurement block. Specifies that a format-1 measurement block is used when performing a measurement-block update for the subchannel. The address of the 64-byte format-1 measurement block is provided by MSCH. |

If the extended-measurement-block facility is not installed, bit 29 of word 6 of the SCHIB operand must be zero when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized.

***Extended Measurement Word Mode Enable (X):*** When the extended-measurement-word facility is installed and enabled, bit 30 of word 6 enables the extended-measurement-word mode for the subchannel. Initially, the extended-measurement-word mode is not enabled. The definition of the bit is as follows:

| Bit 30 | Extended-Measurement-Word-Mode Enable: |
|---|---|
| 0 | Initialized value. The subchannel is not enabled for extended-measurement-word mode. Storing of the extended-measurement word does not occur. |
| 1 | The subchannel is enabled for extended-measurement-word mode. Measurement data is stored in the extended-measurement word at the time channel-program execution is completed or suspended at the subchannel or completed at the device, as appropriate, provided no error conditions described by subchannel logout have been detected. |

If the extended-measurement-word facility is not installed, or is installed but is not enabled, bit 30 of word 6 of the SCHIB operand must be zero when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized.

***Concurrent Sense (S):*** Bit 31 of word 6, when one, indicates that the subchannel is in the concurrent-sense mode. When the subchannel is in concurrent-sense mode, whenever the subchannel becomes status pending with alert status, and the status byte accepted from the device contains the unit-check indication, then the channel subsystem may attempt to retrieve sense information from the associated device and place that sense information in the extended-control word.

If the concurrent-sense facility is not installed, bit 31 of word 6 of the SCHIB operand must be zero when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized.

## Subchannel-Status Word

Words 7-9 of the SCHIB contain a copy of the SCSW. The format of the SCSW is described in "Subchannel-Status Word" on page 16-6. The SCSW is stored by the execution of either STORE SUBCHANNEL or TEST SUBCHANNEL (see "STORE SUBCHANNEL" on page 14-17 and "TEST SUBCHANNEL" on page 14-20).

## Model-Dependent Area/Measurement Block Address

When the extended-I/O-measurement-block facility is not installed, words 10-12 of the SCHIB contain model-dependent information.

When the extended-I/O-measurement-block facility is installed, words 10-11 are defined as the measurement-block-address field. Word 12 contains model-dependent information.

When (1) the measurement-block-update mode is active (see "SET CHANNEL MONITOR" on page 14-12), (2) the subchannel is enabled for the mode (see "Measurement-Mode Enable (MM)" on page 15-3), and (3) the format-1-measurement block is specified (see "Measurement Block Format Control (F)" on page 15-7) at the subchannel, the measurement-block-address field contains the absolute storage address of the measurement block used by the measurement-block-update facility. The measurement-block address designates the beginning of a 64-byte storage area and must be designated on 64-byte boundary. The initial value of the measurement block address is zero.



LzLabs GmbH. Ex. 1008-670

## Summary of Modifiable Fields

Figure 15-3 on page 15-9 lists the initial settings for fields in a subchannel whose device-number-valid bit is one and indicates what modifies the fields.

All of the PMCW fields contain meaningful information when STORE SUBCHANNEL is executed and the designated subchannel is idle. Subchannel fields that the channel subsystem does not modify contain valid information whenever STORE SUBCHANNEL is executed, provided that the device-number-valid bit is one. The validity of the subchannel fields that are modifiable by the channel subsystem depends on the state of the subchannel at the time STORE SUBCHANNEL is executed.

| Subchannel Field | Initial Value[1] | Program Modifies by Executing | Modified by Channel Subsystem[2] |
|---|---|---|---|
| Interruption parameter | Zeros | MSCH,SSCH | No |
| I/O-interruption-subclass code | Zeros | MSCH | No |
| Enabled (E) | Zero | MSCH | No |
| Limit mode (LM) | Zeros | MSCH[7] | No |
| Measurement mode (MM) | Zeros | MSCH | Yes[3] |
| Multipath mode (D) | Zero | MSCH | No |
| Timing facility (T) | Installed value[4] | None | No |
| Device number valid (V) | Installed value[4] | None | No |
| Device number | Installed value[4] | None | No |
| Logical-path mask (LPM) | Path-installed-mask value | MSCH,SSCH | No |
| Path-not-operational mask (PNOM) | Zeros | CSCH,SSCH,RSCH[5] | Yes |
| Last-path-used mask (LPUM) | Zeros | CSCH | Yes |
| Path-installed mask (PIM) | Installed value[4] | None | No |
| Measurement-block index (MBI) | Zeros | MSCH | No |
| Path-operational mask (POM) | Ones | CSCH,MSCH,RSCH[5] | Yes |
| Path-available mask (PAM) | Installed values[4] [6] | None | Yes[6] |
| Channel-path ID 0-7 (CHPID) | Installed value[4] | None | No |
| Concurrent sense (S) | Zero | MSCH | No |
| Subchannel-status word (SCSW) | Zero | TSCH | Yes |
| Model-dependent area | * | None | * |
| Measurement-block-format control | Zero | MSCH | No |
| Extended-measurement-word enable | Zero | MSCH | No |
| Measurement-block Address | Zeros | MSCH | No |

*Figure 15-3 (Part 1 of 2). Modification of Subchannel Fields*


LzLabs GmbH. Ex. 1008-671

**Explanation:**

* Model-dependent.

1 These fields are not meaningful if the subchannel is not valid. Initialization of a subchannel is performed when I/O-system reset occurs. (See the section "I/O-System Reset" in Chapter 17, "I/O Support Functions.") One or more of the installed-value parameters that are unmodifiable by the program may be set when the subchannel is idle. In this case, all the program-modifiable fields are set to their initialized values, and the program is notified of such a change by a channel report. (See the section "Channel-Report Word" in Chapter 17, "I/O Support Functions.")

2 Subchannel fields that are not normally modifiable by the channel subsystem may be modified as a result of dynamic configuration changes or as a result of external actions. When this occurs, the program is notified of the change by a channel report that is made pending at the time of the change.

3 When any of the following error conditions associated with the measurement-block-update mode is detected, the measurement-block-update mode is disabled by the channel subsystem (bit 11 of word 1 of the SCHIB is zero) in the affected subchannel. The device-connect-time-measurement-enable bit (bit 12 of word 1 of the SCHIB) is never modified by the channel subsystem.

   Measurement program check
   Measurement protection check
   Measurement data check
   Measurement key check

4 This information is entered when the channel-subsystem configuration is established.

5 The mask is modified by the resume function only when the subchannel is in the suspended state at the time RESUME SUBCHANNEL is executed.

6 The channel subsystem may modify the PAM to reflect changes in the system configuration caused by partitioning or unpartitioning channel paths because of reconfiguration or permanent failure of part of the I/O system.

7 The limit mode bits are modified by MSCH only when the instruction is executed when the address-limit-checking facility is installed and the CPC is operating in basic mode.

*Figure 15-3 (Part 2 of 2). Modification of Subchannel Fields*



LzLabs GmbH. Ex. 1008-672

**Programming Notes:**

1. System performance may be degraded if the LPM is not used to make channel paths for which a path-not-operational condition has been indicated in the PNOM logically not available.
2. If, during the performance of a start function, a channel path becomes not physically available because a channel-path failure has been recognized, continued performance of the start function may be precluded. That is, the program may or may not be notified, and the subchannel may remain in the subchannel-and-device-active state until cleared by the performance of the clear function.
3. If the same MBI is placed in more than one subchannel by the program, the channel-subsystem-monitoring facility updates the same locations with measurement data relating to more than one subchannel. In this case, the values stored in the measurement data are unpredictable. (See "Measurement-Block Update" on page 17-3.)
4. Modification of the I/O configuration (reconfiguration) may be accomplished in various ways depending on the model. If the reconfiguration procedure affects the physical availability of a channel path, then any change in availability can be detected by executing STORE SUBCHANNEL for a subchannel that has access to the channel path and by subsequently examining the PAM bits of the SCHIB.
5. The definitions of the PNOM, POM, and N bit are such that a path-not-operational condition is reported to the program only the first time the condition is detected by the channel subsystem after the corresponding POM bit is set to one.

   For example, if the POM bit for every channel path available for selection is one and the device appears not operational on all corresponding channel paths while the channel subsystem is attempting to initiate a start function at the device, the channel subsystem makes the subchannel status pending, with deferred condition code 3 and with the N bit stored as one. The PNOM in the SCHIB indicates the channel path or channel paths that appeared not operational, for which the corresponding POM bits have been set to zeros. The next START SUBCHANNEL causes the channel subsystem to again attempt device selection by choosing a channel path from among all of the channel paths that are available for selection. If device selection is not successful and all channel paths available for selection have again been chosen, deferred condition code 3 is set, but the N bit in the SCSW is zero. The POM contains zeros in at least those bit positions that correspond to the channel paths that are available for selection. (See "Channel-Path Availability" on page 15-13 for a description of the term "available for selection.") When the N bit in the SCSW is zero, the PNOM is also zero.
6. If the program is to detect path-not-operational conditions, the PNOM should be inspected following the execution of TEST SUBCHANNEL (which results in the setting of condition code zero and the valid storing of the N bit as one) and preceding the performance of another start, resume, halt, or clear function at the subchannel.

## Channel-Path Allegiance

The channel subsystem establishes allegiance conditions between subchannels and channel paths. The kind of allegiance established at a subchannel for a channel path or set of channel paths depends upon the state of the subchannel, the device, and the information, if any, transferred between the channel subsystem and device. The way in which path management is handled during the performance of a clear, halt, resume, or start function is determined by the kind of allegiance, if any, currently recognized between a subchannel and a channel path.

Performing the clear function at a subchannel clears any currently existing allegiance condition in the subchannel for all channel paths.

Performing the reset-channel-path function clears all currently existing allegiances for that channel path in all subchannels.

When a channel path becomes not physically available, all internal indications of prior allegiance conditions are cleared in all subchannels having access to the designated channel path.


LzLabs GmbH. Ex. 1008-673

## Working Allegiance

A subchannel has a working allegiance for a channel path when the subchannel becomes device active on that channel path. Once a working allegiance is established, the channel subsystem maintains the working allegiance at the subchannel for the channel path until either the subchannel is no longer device active or a dedicated allegiance is recognized, whichever occurs earlier. Unless a dedicated allegiance is recognized, a working allegiance for a channel path is extended to the set of channel paths that are available for selection if the device is specified to be operating in the multipath mode (that is, the multipath-mode bit is stored as one in the SCHIB). Otherwise, the working allegiance remains only for that channel path over which the start function was initiated.

Once a working allegiance is established for a channel path or set of channel paths, the working allegiance is not changed until the subchannel is no longer device active or until a dedicated allegiance is established. If the subchannel is operating in the single-path mode, a working allegiance is maintained only for a single path.

While a working allegiance exists at a subchannel, an active allegiance can occur only for a channel path for which the working allegiance is being maintained, unless the device is specified as operating in the multipath mode. When the device is specified as operating in the multipath mode, an active allegiance may also occur for a channel path that is not available for selection if the presentation of status by the device on that channel path causes an alert interruption condition to be recognized.

A working allegiance is cleared in any subchannel having access to a channel path if the channel path becomes not physically available.

## Active Allegiance

A subchannel has an active allegiance established for a channel path no later than when active communication has been initiated on that channel path with an I/O device. The subchannel can have an active allegiance to only one channel path at a time. While the subchannel has an active allegiance for a channel path, the channel subsystem does not actively communicate with that device on any other channel path. When the channel subsystem accepts a no-longer-busy indication from the device that does not cause an interruption condition, this status does not constitute the initiation of active communication. An active allegiance at a subchannel for a channel path is terminated when the channel subsystem is no longer actively communicating with the I/O device on that channel path.

A working allegiance can become an active allegiance.

## Dedicated Allegiance

If a channel path is physically available (that is, if the corresponding PAM bit is one), a dedicated allegiance may be recognized for that channel path. If a channel path is not physically available, a dedicated allegiance cannot be recognized for the corresponding channel path. The channel subsystem establishes a dedicated allegiance at the subchannel for a channel path when (1) the subchannel becomes status pending with alert status, and device status containing the unit-check indication is present but (2) concurrent-sense information is not present at the subchannel. A dedicated allegiance is maintained until the subchannel is no longer start pending (unless it becomes suspended) or resume pending following performance of the next start function, clear function, or channel-path-reset function or the next resume function if applicable. If the subchannel becomes suspended, the dedicated allegiance remains until the resume function is initiated and the subchannel is no longer resume pending. Unless a clear or channel-path-reset function is performed, the subchannel establishes a working allegiance when the dedicated allegiance ends. This occurs when the subchannel becomes device active. While a dedicated allegiance exists at a subchannel for a channel path, only that channel path is available for selection until the dedicated-allegiance condition is cleared.

A dedicated allegiance can become an active allegiance. While a dedicated allegiance exists, an active allegiance can only occur for the same channel path.

A currently existing dedicated allegiance is cleared at any subchannel having access to a channel path when the channel path becomes not phys-



LzLabs GmbH. Ex. 1008-674

ically available or whenever the device appears not operational on the channel path for which the dedicated allegiance exists.

## Channel-Path Availability

When a channel path is not physically available, the channel subsystem does not use the channel path to perform any of the basic I/O functions except, in some cases, the channel-path-reset function and does not respond to any control-unit-initiated requests on that same channel path. If a channel path is not physically available, the condition is indicated by the corresponding path-available-mask PAM bit being zero when STORE SUBCHANNEL is executed (see "Path-Available Mask (PAM)" on page 15-7). Furthermore, if the channel path is not physically available for the subchannel designated by STORE SUBCHANNEL, then it is not physically available for any subchannel that has a device which is accessible by that channel path.

Unless a dedicated allegiance exists at a subchannel for the channel path, a channel path becomes available for selection if it is logically available and physically available, as indicated by the bits in the LPM and PAM corresponding to the channel path being stored as ones when STORE SUBCHANNEL is executed. If a dedicated allegiance exists at a subchannel for the channel path, only that channel path is available for selection, and the setting of the corresponding LPM bit is ignored. If the channel path is currently being used and a dedicated allegiance exists at the subchannel for the channel path, selection of the device is delayed until the channel path is no longer being used.

The availability status of the eight logical paths to the associated device described in Figure 15-4 is determined by the hierarchical arrangement of the corresponding bit values contained in the PIM, PAM, and LPM and by existing conditions, if any, recognized by the channel subsystem.

| Value of Bit 'n' | | | Channel-Path Condition[1] | Channel-Path State |
|---|---|---|---|---|
| PIM | PAM | LPM | | |
| 0 | 0[2] | - | X | Not installed |
| 1 | 0 | - | X | Not physically available |
| 1 | 1 | 0[3] | X | Not logically available |
| 1 | 1 | 1[3] | Active | Available for selection[4] |
| 1 | 1 | 1 | Inactive | Available for selection |

**Explanation:**

- Bit value is not meaningful.

[1] If the channel path is recognized as being used in active communication with a device, the channel-path condition is described as active. Otherwise, its condition is described as inactive.

[2] A PAM bit cannot have the value one when the corresponding PIM bit has the value zero.

[3] If a dedicated allegiance exists to the channel path at the subchannel, the state of the bit is ignored, and the channel path is considered to be available for selection.

[4] The channel path may appear to be active when a channel-path-terminal condition has been recognized.

X Condition is not meaningful.

*Figure 15-4. Path Condition and Path-Availability Status for PIM, PAM, and LPM Values*

## Control-Unit Type

In "Clear Function" on page 15-14, "Halt Function" on page 15-15, and "Start Function and Resume Function" on page 15-18, reference is made to type-1, type-2, and type-3 control units. For a description of these control-unit types, see the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974. For the purposes of this definition, all control units attaching to a serial-I/O interface are considered type-2 control units.


LzLabs GmbH. Ex. 1008-675

## Clear Function

Subsequent to the execution of CLEAR SUBCHANNEL, the channel subsystem performs the clear function. Performance of the clear function consists in (1) performing a path-management operation, (2) modifying fields at the subchannel, (3) issuing the clear signal to the associated device, and (4) causing the subchannel to be made status pending, indicating the completion of the clear function.

## Clear-Function Path Management

A path-management operation is performed as part of the clear function in order to examine channel-path conditions for the associated subchannel and to attempt to choose an available channel path on which the clear signal can be issued to the associated device.

Channel-path conditions are examined in the following order:

1. If the channel subsystem is actively communicating or attempting to establish active communication with the device to be signaled, the channel path that is in use is chosen.
2. If the channel subsystem is in the process of accepting a no-longer-busy indication (which will not cause an interruption condition to be recognized) from the device to be signaled, and the associated subchannel has no allegiance to any channel path, the channel path that is in use is chosen.
3. If the associated subchannel has a dedicated allegiance for a channel path, that channel path is chosen.
4. If the associated subchannel has a working allegiance for one or more channel paths, one of those channel paths is chosen.
5. If the associated subchannel has no allegiance for any channel path, if a last-used channel path is indicated, and if that channel path is available for selection, that channel path is chosen. If that channel path is not available for selection, either no channel path is chosen or a channel path is chosen from the set of channel paths, if any, that are available for selection (as though no last-used channel path were indicated).
6. If the associated subchannel has no allegiance for any channel path, if no last-used channel path is indicated, and if there exist one or more channel paths that are available for selection, one of those channel paths is chosen.

If none of the channel-path conditions listed above apply, no channel path is chosen.

For item 4, for item 5 under the specified conditions, and for item 6, the channel subsystem chooses a channel path from a set of channel paths. In these cases, the channel subsystem may attempt to choose a channel path, provided that the following conditions *do not* apply:

1. A channel-path-terminal condition exists for the channel path.
2. Another subchannel has an active allegiance for the channel path.
3. The device to be signaled is attached to a type-1 control unit, and the subchannel for another device attached to the same control unit has an allegiance to the same channel path, unless the allegiance is a working allegiance and primary status has been accepted by that subchannel.
4. The device to be signaled is attached to a type-3 control unit, and the subchannel for another device attached to the same control unit has a dedicated allegiance to the same channel path.

## Clear-Function Subchannel Modification

Path-management-control indications at the subchannel are modified during performance of the clear function. Effectively, this modification occurs after the attempt to choose a channel path, but prior to the attempt to select the device to issue the clear signal. The path-management-control indications that are modified are as follows:

1. The state of all eight possible channel paths at the subchannel is set to operational for the subchannel.
2. The last-path-used indication is reset to indicate no last-used channel path.
3. Path-not-operational conditions, if any, are reset.



LzLabs GmbH. Ex. 1008-676

## Clear-Function Signaling and Completion

Subsequent to the attempt to choose a channel path and the modification of the path-management-control fields, the channel subsystem, if conditions allow, attempts to select the device to issue the clear signal. (See "Clear Signal" on page 17-12.) Conditions associated with the subchannel and the chosen channel path, if any, affect (1) whether an attempt is made to issue the clear signal, and (2) whether the attempt to issue the clear signal is successful. Independent of these conditions, the subchannel is subsequently set status pending, and the performance of the clear function is complete. These conditions and their effect on the clear function are described as follows:

***No Attempt Is Made to Issue the Clear Signal:*** The channel subsystem does not attempt to issue the clear signal to the device if any of the following conditions exist:

1. No channel path was chosen. (See "Clear-Function Path Management" on page 15-14.)
2. The chosen channel path is no longer available for selection.
3. A channel-path-terminal condition exists for the chosen channel path.
4. The chosen channel path is currently being used to actively communicate with a different device.
5. The device to be signaled is attached to a type-1 control unit, and the subchannel for another device attached to the same control unit has an allegiance to the same channel path, unless the allegiance is a working allegiance and primary status has been accepted by that subchannel.
6. The device to be signaled is attached to a type-3 control unit, and the subchannel for another device attached to the same control unit has a dedicated allegiance to the same channel path.

If any of the conditions above exist, the subchannel remains clear pending and is set status pending, and the performance of the clear function is complete.

***The Attempt to Issue the Clear Signal Is Not Successful:*** When the channel subsystem attempts to issue the clear signal to the device, the attempt may not be successful because of the following conditions:

1. The control unit or device signals a busy condition when the channel subsystem attempts to select the device to issue the clear signal.
2. A path-not-operational condition is recognized when the channel subsystem attempts to select the device to issue the clear signal.
3. An error condition is encountered when the channel subsystem attempts to issue the clear signal.

If any of the conditions above exists and the channel subsystem either determines that the attempt to issue the clear signal was not successful or cannot determine whether the attempt was successful, the subchannel remains clear pending and is set status pending, and the performance of the clear function is complete.

***The Attempt to Issue the Clear Signal Is Successful:*** When the channel subsystem determines that the attempt to issue the clear signal was successful, the subchannel is no longer clear pending and is set status pending, and the performance of the clear function is complete. When the subchannel becomes status pending, the I/O operation, if any, with the associated device has been terminated.

**Programming Note:** Subsequent to the performance of the clear function, any nonzero status, except control unit end alone, that is presented to the channel subsystem by the device is passed to the program as unsolicited alert status. Unsolicited status consisting of control unit end alone or zero status is not presented to the program.

## Halt Function

Subsequent to the execution of HALT SUBCHANNEL, the channel subsystem performs the halt function. Performance of the halt function consists of (1) performing a path-management operation, (2) issuing the halt signal to the associated device, and (3) causing the subchannel to be made status pending, indicating the completion of the halt function.


LzLabs GmbH. Ex. 1008-677

## Halt-Function Path Management

A path-management operation is performed as part of the halt function to examine channel-path conditions for the associated subchannel and to attempt to choose a channel path on which the halt signal can be issued to the associated device.

Channel-path conditions are examined in the following order:

1. If the channel subsystem is actively communicating or attempting to establish active communication with the device to be signaled, the channel path that is in use is chosen.
2. If the channel subsystem is in the process of accepting a no-longer-busy indication (which will not cause an interruption condition to be recognized) from the device to be signaled, and the associated subchannel has no allegiance to any channel path, the channel path that is in use is chosen.
3. If the associated subchannel has a dedicated allegiance for a channel path, that channel path is chosen.
4. If the associated subchannel has a working allegiance for one or more channel paths, one of those channel paths is chosen.
5. If the associated subchannel has no allegiance for any channel path, if a last-used channel path is indicated, and if that channel path is available for selection, that channel path is chosen. If that channel path is not available for selection, either no channel path is chosen or a channel path is chosen from the set of channel paths, if any, that are available for selection (as though no last-used channel path were indicated).
6. If the associated subchannel has no allegiance for any channel path, if no last-used channel path is indicated, and if there exist one or more channel paths that are available for selection, one of those channel paths is chosen.

If none of the channel-path conditions listed above apply, no channel path is chosen.

For item 4, for item 5 under the specified conditions, and for item 6, the channel subsystem chooses a channel path from a set of channel paths. In these cases, the channel subsystem may attempt to choose a channel path for which the following conditions *do not* apply:

1. A channel-path-terminal condition exists for the channel path.
2. Another subchannel has an active allegiance for the channel path.
3. The device to be signaled is attached to a type-1 control unit, and the subchannel for another device attached to the same control unit has an allegiance to the same channel path, unless the allegiance is a working allegiance and primary status has been accepted by that subchannel.
4. The device to be signaled is attached to a type-3 control unit, and the subchannel for another device attached to the same control unit has a dedicated allegiance to the same channel path.

## Halt-Function Signaling and Completion

Subsequent to the attempt to choose a channel path, the channel subsystem, if conditions allow, attempts to select the device to issue the halt signal. (See “Halt Signal” on page 17-12.) Conditions associated with the subchannel and the chosen channel path, if any, affect (1) whether an attempt is made to issue the halt signal, (2) whether the attempt to issue the halt signal is successful, and (3) whether the subchannel is made status pending to complete the halt function. These conditions and their effect on the halt function are described as follows:

***No Attempt Is Made to Issue the Halt Signal:***
The channel subsystem does not attempt to issue the halt signal to the device if any of the following conditions exist:

1. No channel path was chosen. (See “Halt-Function Path Management.”)
2. The chosen channel path is no longer available for selection.
3. A channel-path-terminal condition exists for the chosen channel path.
4. The associated subchannel is status pending with other than intermediate status alone.
5. The device to be signaled is attached to a type-1 control unit, and the subchannel for



LzLabs GmbH. Ex. 1008-678

another device attached to the same control unit has an allegiance to the same channel path, unless the allegiance is a working allegiance and primary status has been accepted by that subchannel.

6. The device to be signaled is attached to a type-3 control unit, and the subchannel for another device attached to the same control unit has a dedicated allegiance to the same channel path.

If the conditions described in items 3 on page 15-16, 5 on page 15-16, or 6 exist, the associated subchannel remains halt pending until those conditions no longer exist. When the conditions no longer exist (for the channel-path-terminal condition, when the condition no longer exists as a result of executing RESET CHANNEL PATH), the channel subsystem attempts to issue the halt signal to the device.

If any of the remaining conditions above exist, the subchannel remains halt pending and is set status pending, and the halt function is complete.

***The Attempt to Issue the Halt Signal Is Not Successful:*** When the channel subsystem attempts to issue the halt signal to the device, the attempt may not be successful because of the following conditions:

1. The control unit or device signals a busy condition when the channel subsystem attempts to select the device to issue the halt signal.
2. A path-not-operational condition is recognized when the channel subsystem attempts to select the device to issue the halt signal.
3. An error condition is encountered when the channel subsystem attempts to issue the halt signal.

If the control unit or device signals a busy condition (item 1), the subchannel remains halt pending until the internal indication of busy is reset. When this event occurs, the channel subsystem again attempts to issue the halt signal to the device.

If any of the remaining conditions above exists and the channel subsystem either determines that the attempt to issue the halt signal was not successful or cannot determine whether the attempt was successful, then the subchannel remains halt pending and is set status pending, and the halt function is complete.

***The Attempt to Issue the Halt Signal Is Successful:*** When the channel subsystem determines that the attempt to issue the halt signal was successful and ending status, if appropriate, has been received at the subchannel, the subchannel is no longer halt pending and is set status pending, and the halt function is complete. When the subchannel becomes status pending, the I/O operation, if any, with the associated device has been terminated. The conditions that affect the receipt of ending status at the subchannel, and the effect of the halt signal at the device are described in the following discussion.

When the subchannel is subchannel-and-device active or only device active during the performance of the halt function, the state continues until the subchannel is made status pending because (1) the device has provided ending status or (2) the channel subsystem has determined that ending status is unavailable. When the subchannel is idle, start pending, start pending and resume pending, suspended, or suspended and resume pending, or when the halt signal is issued during command chaining after the receipt of device end but before the next command is transferred to the device, no operation is in progress at the device, and therefore no status is generated by the device as a result of receiving the halt signal. When the subchannel is neither subchannel active, nor status pending with intermediate status, and no errors are detected during the attempt to issue the halt signal to the device, an interruption condition indicating status pending alone is generated after the halt signal is issued.

The effect of the halt signal at the device depends partially on the type of device and its state. The effect of the halt signal on a device that is not active or that is performing a mechanical operation in which data is not transferred across the channel path, such as rewinding tape or positioning a disk-access mechanism, depends upon the control-unit or device model. If the device is performing a type of operation that is unpredictable in duration or in which data is transferred across the channel path, the control unit interprets the signal as one to terminate the operation. Pending status conditions at the device are not reset. When the control unit recognizes the halt signal, it immediately ceases all communication with the channel subsystem until it has reached the normal ending point. The control unit then requests selection by


LzLabs GmbH. Ex. 1008-679

the channel subsystem to present any generated status.

If the subchannel is involved in the data-transfer portion of an I/O operation, data transfer is terminated during the performance of the halt function, and the device is logically disconnected from the channel path. If the halt function is addressed to a subchannel performing a chain of I/O operations and the device has already provided channel end for the current I/O operation, the channel subsystem causes the device to be disconnected and command chaining or command retry to be suppressed. If the subchannel is performing a chain of I/O operations with the device and the halt signal is issued during command chaining at a point after the receipt of device end for the previous I/O operation but before the next command is transferred to the device, the subchannel is made status pending with primary and secondary status immediately after the halt signal is issued. The device-status field of the SCSW contains zeros in this case. If the halt function is addressed to a subchannel that is start pending and the halt-pending condition is recognized before initiation of the start function, initiation of the start function is not attempted, and the subchannel becomes status pending after the device has been signaled.

When the subchannel is not performing an I/O operation with the associated device, the device is selected, and an attempt is made to issue the halt signal as the device responds. If the subchannel is in the device-active state, the subchannel does not become status pending until it receives the device-end status from the halted device. If the subchannel is neither subchannel-and-device active nor device active, the subchannel becomes status pending immediately after selecting the device and issuing the halt signal. The SCSW for the latter case has the status-pending bit set to one (see "Status-Pending (Bit 31)" on page 16-18).

The termination of an I/O operation by performing the halt function may result in two distinct interruption conditions.

The first interruption condition occurs when the device generates the channel-end condition. The channel subsystem handles this condition as it would any other interruption condition from the device, except that the command address in the associated SCSW designates the point at which the I/O operation is terminated, and the subchannel-status bits may reflect unusual conditions that were detected. If the halt signal was issued before all data designated for the operation had been transferred, incorrect length is indicated, subject to the control of the SLI flag in the current CCW. The value in the count field of the associated SCSW is unpredictable.

The second interruption condition occurs if device-end status was not presented with the channel-end interruption condition. In this situation, the subchannel-key, command-address, and count fields of the associated SCSW are not meaningful.

When HALT SUBCHANNEL terminates an I/O operation, the method of termination differs from that used upon exhaustion of count or upon detection of programming errors to the extent that termination by HALT SUBCHANNEL is not contingent on the receipt of a service request from the associated device.

**Programming Notes:**

1. When, after an operation is terminated by HALT SUBCHANNEL, the subchannel is status pending with primary, primary and secondary, or secondary status, the extent of data transferred as described by the count field is unpredictable.
2. When the path that is chosen by the path-management operation has a channel-path-terminal condition associated with it, the halt function remains pending until the condition no longer exists. Until the condition is cleared, the associated subchannel cannot be used to perform I/O operations, even if other channel paths become available for selection. CLEAR SUBCHANNEL can be executed to terminate the halt-pending condition and make the subchannel usable.

## Start Function and Resume Function

Subsequent to the execution of START SUBCHANNEL and RESUME SUBCHANNEL, the channel subsystem performs the start and resume functions, respectively, to initiate an I/O operation with the associated device. Performance of a



LzLabs GmbH. Ex. 1008-680

start or resume function consists of: (1) performing a path-management operation, (2) performing an I/O operation or chain of I/O operations with the associated device, and (3) causing the subchannel to be made status pending, indicating the completion of the start function. (Completion of a start function is described in Chapter 16, "I/O Interruptions" on page 16-1.) The start function initiates the execution of a channel program that is designated in the ORB, which in turn is designated as the operand of START SUBCHANNEL, in contrast to the resume function that initiates the execution of a suspended channel program, if any, beginning at the CCW that caused suspension; otherwise, the resume function is performed as if it were a start function (see "Resume-Pending (Bit 20)" on page 16-13).

## Start-Function and Resume-Function Path Management

A path-management operation is performed by the channel subsystem during the performance of either a start or a resume function to choose an available channel path that can be used for device selection to initiate an I/O operation with that device. The actions taken are as follows:

1. If the subchannel is currently start pending and device active, the start function remains pending at the subchannel until the secondary status for the previous start function has been accepted from the associated device and the subchannel is made start pending alone. When the status is accepted and does not describe an alert interruption condition, the subchannel is not made status pending, and the performance of the pending start function is subsequently initiated. If the status describes an alert interruption condition, the subchannel becomes status pending with secondary and alert status, the pending start function is not initiated, deferred condition code 1 is set, and the start-pending bit remains one. If the subchannel is currently start pending alone, the performance of the start function is initiated as described below.
2. If a dedicated allegiance exists at the subchannel for a channel path, the channel subsystem chooses that path for device selection. If a busy condition is encountered while attempting to select the device and a dedicated allegiance exists at the subchannel, the start function remains pending until the internal indication of busy is reset for that channel path. When the internal indication of busy is reset, the performance of the pending start function is initiated on that channel path.
3. If no channel path is available for selection and no dedicated allegiance exists in the subchannel for a channel path, a channel path is not chosen.
4. If all channel paths that are available for selection have been tried and one or more of them are being used to actively communicate with other devices, or, alternatively, if the channel subsystem has encountered either a control-unit-busy or a device-busy condition on one or more of those channel paths, or a combination of those conditions on one or more of those channel paths, the start function remains pending at the subchannel until a channel path, control unit, or device, as appropriate, becomes available.
5. If (1) the start function is to be initiated on a channel path with a device attached to a type-1 control unit and (2) no other device is attached to the same control unit whose subchannel has either a dedicated allegiance to the same channel path or a working allegiance to the same channel path where primary status has not been received for that subchannel, then that channel path is chosen if it is available for selection; otherwise, that channel path is not chosen. If, however, another channel path to the device is available for selection and no allegiances exist as described above, that channel path is chosen. If no other channel path is available for selection, the start or resume function, as appropriate, remains pending until a channel path becomes available.
6. If the device is attached to a type-3 control unit, and if at least one other device is attached to the same control unit whose subchannel has a dedicated allegiance to the same channel path, another channel path that is available for selection may be chosen, or the start function remains pending until the dedicated allegiance for the other device is cleared.
7. If a channel path has been chosen and a busy indication is received during device selection


LzLabs GmbH. Ex. 1008-681

to initiate the execution of the first command of a pending channel program, the channel path over which the busy indication is received is not used again for that device or control unit (depending on the device-busy or control-unit-busy indication received) until the internal indication of busy is reset.

8. If, during an attempt to select the device in order to initiate the execution of the first command specified for the start or implied for the resume function (as described in action 7 on page 15-19), the channel subsystem receives a busy indication, it performs one of the following actions:

   a. If the device is specified to be operating in the multipath mode and the busy indication received is device busy, then the start or resume function remains pending until the internal indication of busy is reset. (See "Multipath Mode (D)" on page 15-3.)

   b. If the device is specified to be operating in the multipath mode and the busy indication received is control unit busy, or if the device is specified to be operating in the single-path mode, the channel subsystem attempts selection of the device by choosing an alternate channel path that is available for selection and continues the path-management operation until either the start or the resume function is initiated or selection of the device has been attempted on all channel paths that are available for selection. If the start or resume function has not been initiated by the channel subsystem after all channel paths available for selection have been chosen, the start or resume function remains pending until the internal indication of busy is reset.

   c. If the subchannel has a dedicated allegiance, then action 2 on page 15-19 applies.

9. When, during the selection attempt to transfer the first command, the device appears not operational and the corresponding channel path is operational for the subchannel, a path-not-operational condition is recognized, and the state of the channel path changes at the subchannel from operational for the subchannel to not operational for the subchannel (see "Path-Not-Operational Mask (PNOM)" on page 15-4). The path-not-operational conditions at the subchannel, if any, are preserved until the subchannel next becomes clear pending, start pending, or resume pending (if the subchannel was suspended), at which time the path-not-operational conditions are cleared. If, however, the corresponding channel path is not operational for the subchannel, a path-not-operational condition is not recognized. When the device appears not operational during the selection attempt to transfer the first command on a channel path that is available for selection, one of the following actions occurs:

   a. If a dedicated allegiance exists for that channel path, then it is the only channel path that is available for selection; therefore, further attempts to initiate the start or resume function are abandoned, and an interruption condition is recognized.

   b. If no dedicated allegiance exists and there are alternate channel paths available for selection that have not been tried, one of those channel paths is chosen to attempt device selection and transfer the first command.

   c. If no dedicated allegiance exists, no alternate channel paths are available for selection that have not been tried, and the device has appeared operational on at least one of the channel paths that were tried, the start or resume function remains pending at the subchannel until a channel path, a control unit, or the device, as appropriate, becomes available.

   d. If no dedicated allegiance exists, no alternate channel paths are available for selection that have not been tried, and the device has appeared not operational on all channel paths that were tried, further attempts to initiate the start or resume function are abandoned, and an interruption condition is recognized.

10. When the subchannel is active and an I/O operation is to be initiated with a device, all device selections occur according to the LPUM indication if the multipath mode is not specified at the subchannel. For example, if command chaining is specified, the channel subsystem transfers the first and all subse-



LzLabs GmbH. Ex. 1008-682

quent commands describing a chain of I/O operations over the same channel path.

## Execution of I/O Operations

After a channel path is chosen, the channel subsystem, if conditions allow, initiates the execution of an I/O operation with the associated device. Execution of additional I/O operations may follow the initiation and execution of the first I/O operation. The channel subsystem can execute seven commands: write, read, read backward, control, sense, sense ID, and transfer in channel. Each command, except transfer in channel, initiates a corresponding I/O operation. Except for periods when channel-program execution is suspended at the subchannel (see "Suspension of Channel-Program Execution" on page 15-38), the subchannel is active from the acceptance of the first command until the primary interruption condition is recognized at the subchannel. If the primary interruption condition is recognized before the acceptance of the first command, the subchannel does not become active. Normally, the primary interruption condition is caused by the channel-end signal or, in the case of command chaining, the channel-end signal for the last CCW of the chain. (See "Primary Interruption Condition" on page 16-4.) The device is active until the secondary interruption condition is recognized at the subchannel. Normally, the secondary interruption condition is caused by the device-end signal or, in the case of command chaining, the device-end signal for the last CCW of the chain. (See "Secondary Interruption Condition" on page 16-4.)

**Programming Notes:**

In the single-path mode, all transfers of commands, data, and status for the I/O operation or chain of I/O operations occur on the channel path over which the first command was transferred to the device.

When the device has the dynamic-reconnection feature installed, an I/O operation or chain of I/O operations may be performed in the multipath mode. To operate in the multipath mode, MODIFY SUBCHANNEL must have been previously executed for the subchannel with bit 13 of word 1 of the SCHIB specified as one. (See "Multipath Mode (D)" on page 15-3.) In addition, the device must be set up for the multipath mode by the execution of certain model-dependent commands appropriate to that type of device. The general procedures for handling multipath-mode operations are as follows:

1. Setup

   a. A set-multipath-mode type of command must be successfully executed by the device on each channel path that is to be a member of the multipath group being set up; otherwise, the multipath mode of operation may give unpredictable results at the subchannel. If, for any reason, one or more physically available channel paths to the device are not included in the multipath group, these channel paths must not be available for selection while the subchannel is operating in the multipath mode. A channel path can be made not available for selection by having the corresponding LPM bit set to zero either in the SCHIB prior to the execution of MODIFY SUBCHANNEL or in the ORB prior to the execution of START SUBCHANNEL.

   b. When a set-multipath-mode type of command is transferred to a device, only a single channel path must be logically available in order to avoid alternate channel-path selection for the execution of that start function; otherwise, device-busy conditions may be detected by the channel subsystem on more than one channel path, which may cause unpredictable results for subsequent multipath-mode operations. This type of setup procedure should be used whenever the membership of a multipath group is changed.

2. Leaving the Multipath Mode

   To leave the multipath mode and continue processing in the single-path mode, either of the following two procedures may be used:

   a. A disband-multipath-mode type of command may be executed for any channel path of the multipath group. This command must be followed by either (1) the execution of MODIFY SUBCHANNEL with bit 13 of word 1 of the SCHIB specified as zero, or (2) the specification of only a single channel path as logically available in the LPM. A start function must not be performed at a subchannel operating in the multipath mode


LzLabs GmbH. Ex. 1008-683

with multiple channel paths available for selection while the device is operating in single-path mode; otherwise, unpredictable results may occur at the subchannel for that function or subsequent start functions.

b. A resign-multipath-mode type of command is executed on each channel path of the multipath group (the reverse of the setup described in item 1 on page 15-21). This command must be followed by either (1) the execution of MODIFY SUBCHANNEL with bit 13 of word 1 of the SCHIB specified as zero, or (2) the specification of only a single channel path as logically available in the LPM. No start function may be performed at a subchannel operating in the multipath mode with multiple channel paths available for selection while the device is operating in single-path mode; otherwise, unpredictable results may occur at the subchannel for that or subsequent start functions.

## Blocking of Data

Data recorded by an I/O device is divided into blocks. The length of a block depends on the device; for example, a block can be a card, a line of printing, or the information recorded between two consecutive gaps on magnetic tape.

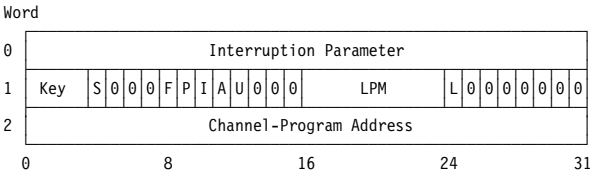
The maximum amount of information that can be transferred in one I/O operation is one block. An I/O operation is terminated when the associated main-storage area is exhausted or the end of the block is reached, whichever occurs first. For some operations, such as writing on a magnetic-tape unit or at an inquiry station, blocks are not defined, and the amount of information transferred is controlled only by the program.

## Operation-Request Block

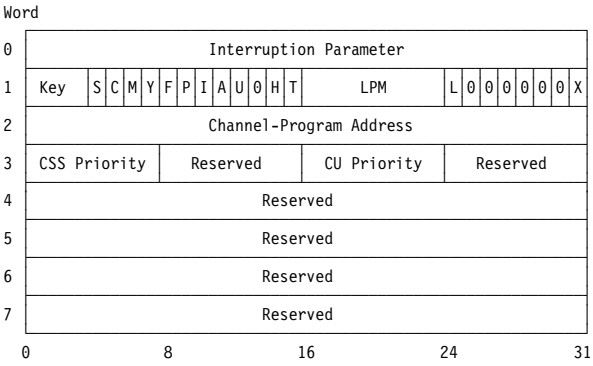
The operation-request block (ORB) is the operand of START SUBCHANNEL. The ORB specifies the parameters to be used in controlling that particular start function. These parameters include the interruption parameter, the subchannel key, the address of the first CCW, operation-control bits, and a specification of the logical availability of channel paths.

The contents of the ORB are placed at the designated subchannel during the execution of START SUBCHANNEL, prior to the setting of condition code 0. If the execution will result in a nonzero condition code, the contents of the ORB are not placed at the designated subchannel.

The two rightmost bits of the ORB address must be zeros, placing the ORB on a word boundary; otherwise, a specification exception is recognized. When the ORB-extension facility is not installed, the ORB is three words having the following format:




When the ORB-extension facility is installed, the ORB also contains additional operation-control bits, and numbers controlling priority, and it is eight words having the following format:




The fields in the ORB are defined as follows:

***Interruption Parameter:*** Bits 0-31 of word 0 are preserved unmodified in the subchannel until replaced by a subsequent START SUBCHANNEL or MODIFY SUBCHANNEL instruction. These bits are placed in word 1 of the interruption code when an I/O interruption occurs and when an interruption request is cleared by the execution of TEST PENDING INTERRUPTION.

***Subchannel Key:*** Bits 0-3 of word 1 form the subchannel key for all fetching of CCWs, IDAWs, and output data and for the storing of input data associated with the start function initiated by START SUBCHANNEL. This key is matched with



LzLabs GmbH. Ex. 1008-684

a storage key during these storage references. For details, see the section "Key-Controlled Protection" on page 3-9.

***Suspend Control (S):*** Bit 4 of word 1 controls the performance of the suspend function for the channel program designated in the ORB. The setting of the S bit applies to all CCWs of the channel program designated by the ORB (see "Commands and Flags" on page 15-40). When bit 4 is one, suspend control is specified, and channel-program suspension occurs when a suspend flag set to one is detected in a CCW. When bit 4 is zero, suspend control is not specified, and the presence of a suspend flag set to one in any CCW of the channel program causes a program-check condition to be recognized.

***Streaming-Mode Control (C):*** When the FICON-channel facility is installed, bit 5 of word 1 controls streaming-mode enablement for subchannels configured to FICON-converted-I/O-interface channel paths during performance of the specified start function. When bit 5 is zero, the streaming mode is enabled at the subchannel. When bit 5 is one, the streaming mode is disabled at the subchannel. Bit 5 is meaningful only for subchannels configured to FICON-converted-I/O-interface channel paths and is ignored for subchannels configured to other channel-path types.

When the streaming mode is enabled, the channel path considers the first command of the designated channel program to be in progress at the associated device when the channel path receives the indication that the command has been accepted at the device. In addition, the channel path's acceptance of status, under certain conditions, is recognized by the channel path without receiving acknowledgement of status acceptance from the device.

When the streaming mode is not enabled, the channel path does not consider the first command of the designated channel program to be in progress at the associated device until the appropriate channel-path response, indicating that the device-command response or status has been accepted at the channel path, is sent to the device. In addition, when the device sends status to the channel path, the channel path's acceptance of that status is not recognized at the channel path until the channel path's confirmation of acceptance is received and acknowledged by the device.

***Modification Control (M):*** When the FICON-channel facility is installed, bit 6 of word 1 specifies whether modification control is required for the channel program. When bit 6 is zero, modification control is specified. When bit 6 is one, modification control is not specified.

When modification control is specified, the channel subsystem forces command synchronization with the addressed I/O device each time a command is executed and the previously executed command has the PCI and chain-command flags set to one and the chain-data and suspend flags set to zero. When this condition is recognized, the channel subsystem signals a synchronization request to the I/O device for the current command. The channel subsystem temporarily suspends command chaining and does not fetch (or refetch) the next command-chained CCW until after normal ending status is received for the synchronizing command.

When modification control is not specified, then command synchronization is not required, and the channel subsystem may transfer commands to the I/O device without waiting for status.

The M bit is meaningful only for subchannels configured to FICON-I/O-interface or FICON-converted-I/O-interface channel paths and is ignored for other subchannels configured to other channel-path types.

**Programming Notes:**

1. FICON-converted-I/O-interface channel paths, modification control provides the capability to optimize dynamically modified channel programs that use the PCI flag in the CCW to initiate channel-program modification. Specifically, it allows the program to delay the channel-subsystem fetching and transferring of commands until after status is received for the command following the command with the PCI bit set. This increases the likelihood that a program-controlled interruption will be accepted by a CPU and acted upon by the program that dynamically modifies one or more command-chained CCWs that follow the synchronizing command.


LzLabs GmbH. Ex. 1008-685

In order to increase the probability that any dynamically modified CCWs are fetched after their modification and not prior to their modification, the modifying program should be executed as soon as possible following the CPU's acceptance of the program-controlled interruption. Additionally, the program should minimize the periods during which the configured CPUs are disabled for I/O interruptions.

For channel paths other than FICON-I/O-interface or FICON-converted-I/O-interface channel paths, command synchronization is implicit in the signalling protocol between the channel subsystem and the I/O device; therefore no explicit programming action is required to force command synchronization. Regardless, the program should still attempt to accept and process program-controlled interruptions for these channel-path types in as timely a manner as possible for the same reason as stated above.

2. In order to allow the channel subsystem to optimize the execution of channel programs for FICON-I/O-interface or FICON-converted-I/O-interface channel paths, use of the modification-control facility is discouraged except for channel programs that require dynamic modification.

***Synchronization Control (Y):*** When the FICON-channel facility is installed, bit 7 of word 1 specifies whether synchronization control is required for the channel program. When bit 7 is zero and the prefetch-control bit, bit 9 of word 1, is one, synchronization control is specified. When bit 7 is one and bit 9 is one, synchronization control is not specified.

When synchronization control is specified, the channel subsystem forces command synchronization with the addressed I/O device whenever the current command in execution describes an input operation and the next CCW to be fetched describes an output operation. When this condition is recognized, the channel subsystem signals a synchronization request to the I/O device when the input command is transferred. The transfer of the output command is held pending at the subchannel until normal ending status, signaling the completion of the performance of the input operation by the I/O device, is received. Upon receipt of the ending status, the channel subsystem fetches (or refetches) the data associated with the output command and transfers it to the I/O device.

When synchronization control is not specified, the channel subsystem may transfer commands of the channel program without awaiting status that would signal the completion of the I/O operation for each command.

The Y bit is meaningful only when the subchannel is configured to FICON-I/O-interface or FICON-converted-I/O-interface channel paths and the prefetch-control bit, bit 9 of word 1, is one. The Y bit is ignored for subchannels configured to other channel-path types and when the prefetch-control bit is zero.

When the FICON-channel facility is not installed, bits 5-7 of word 1 must be zeros; otherwise, either an operand exception or a program-check condition is recognized.

***Format Control (F):*** Bit 8 of word 1 specifies the format of the channel-command words (CCWs) that make up the channel program designated by the channel-program-address field. When bit 8 of word 1 is zero, format-0 CCWs are specified. When bit 8 is one, format-1 CCWs are specified. (See “Channel-Command Word” on page 15-27 for the definition of the CCW formats.)

***Prefetch Control (P):*** Bit 9 of word 1 specifies whether or not unlimited prefetching of CCWs is allowed for the channel program. When bit 9 is one, unlimited prefetching of CCWs is allowed. (Unlimited prefetching of data and IDAWs associated with the current and prefetched CCWs is always allowed.) It is model dependent whether prefetching is actually performed.

When bit 9 of word 0 is zero, no prefetching is allowed, except in the case of data chaining on output, where the prefetching of one CCW describing a data area is allowed. When bit 9 of word 0 is zero, the synchronization-control bit, bit 7 of word 1, is ignored.

Additional controls may limit the scope of prefetching.

***Initial-Status-Interruption Control (I):*** Bit 10 of word 1 specifies whether or not the channel subsystem must verify to the program that the device has accepted the first command associated with a



LzLabs GmbH. Ex. 1008-686

start or resume function. When the I bit is specified as one in the ORB, then, when the subchannel becomes active, indicating that the first command has been accepted for this start or resume function, the Z bit (see "Zero Condition Code (Z)" on page 16-11) is set to one at this subchannel, and the subchannel becomes status pending with intermediate status.

If the subchannel does not become active — for example, when the device signals channel end immediately upon receiving the first command, command chaining is not specified in the CCW, and command retry is not signaled — the command-accepted condition (Z bit set to one) is not generated; instead, the subchannel becomes status pending with primary status. Intermediate status may also be indicated in this case when the command is accepted if the first CCW contained the PCI flag set to one.

***Address-Limit-Checking Control (A):*** When the address-limit-checking facility is installed, bit 11 of word 1 specifies whether or not address-limit checking is specified for the channel program. When this bit is zero, no address-limit checking is performed for the execution of the channel program, independent of the setting of the limit-mode bits in the subchannel (see "Limit Mode (LM)" on page 15-3). When this bit is one, address-limit checking is allowed for the channel program, subject to the setting of the limit-mode bits in the subchannel.

When the address-limit-checking facility is not installed, the address-limit-checking-control bit (A) must be zero in the ORB when START SUBCHANNEL is executed; otherwise, an operand exception is recognized.

***Suppress-Suspended-Interruption Control (U):*** Bit 12 of word 1, when one, specifies that the channel subsystem is to suppress the generation of an intermediate interruption condition due to suspension if the subchannel becomes suspended. When bit 12 is zero, the channel subsystem generates an intermediate interruption condition whenever the subchannel becomes suspended during the execution of the channel program.

***Format-2-IDAW Control (H):*** When the CPU has the z/Architecture architectural mode installed, bit 14 of word 1 specifies the format of IDAWs for CCWs that specify indirect data addressing. When bit 14 of word 1 is one, format-2 (64-bit data address) IDAWs are provided for all CCWs that have the IDAW flag set to one in the CCW. When bit 14 of word 1 is zero, format-1 (31-bit data address) IDAWs are provided for all CCWs that have the IDAW flag set to one in the CCW.

**Programming Note:** The format-2-IDAW-control bit may be one when the CPU is in either the ESA/390 architectural mode or the z/Architecture architectural mode. However, when the CPU is in the ESA/390 architectural mode, program access to any data locations at addresses greater than $2^{31} - 1$ is not possible until the CPU is placed into the z/Architecture architectural mode.

***2K-IDAW Control (T):*** When the CPU has the z/Architecture architectural mode installed, bit 15 of word 1 specifies the main-storage block size for format-2-IDAW data areas. Bit 15 is meaningful only when bit 14 (format-2 IDAW control) is one and is ignored when bit 14 is zero. When bit 15 of word 1 is one, all format-2 IDAWs designate 2K-byte storage blocks. When bit 15 of word 1 is zero, all format-2 IDAWs designate 4K-byte storage blocks.

Bit 13 of word 1 is reserved for future use and must be zero. When the z/Architecture architectural mode is not installed, bits 14 and 15 of word 1 must be zeros. Otherwise, either an operand exception or a program-check condition is recognized.

***Logical-Path Mask (LPM):*** Bits 16-23 of word 1 are preserved unmodified in the subchannel and specify to the channel subsystem which of the logical paths 0-7 are to be considered logically available, as viewed by the program. A bit setting of one means that the corresponding channel path is logically available; a zero specifies that the corresponding channel path is logically not available. If a channel path is specified by the program as being logically not available, the channel subsystem does not use that channel path to perform clear, halt, resume, or start functions when requested by the program, except when a dedicated-allegiance condition exists for that channel path. If a dedicated-allegiance condition exists, the setting of the LPM is ignored, and a resume, start, halt, or clear function is performed by using the channel path having the dedicated allegiance.


LzLabs GmbH. Ex. 1008-687

***Incorrect-Length-Suppression Mode (L):*** When bit 8 of word 1 is one, then bit 24 of word 1, when one, specifies the incorrect-length-suppression mode. When the subchannel is in this mode when an immediate operation occurs (that is, when a device signals the channel-end condition during initiation of the command) and the current CCW contains a nonzero value in bit positions 16-31, indication of an incorrect-length condition is suppressed.

When bit 8 of word 1 is one, then bit 24 of word 1, when zero, specifies the incorrect-length-indication mode. When the subchannel is in this mode when an immediate operation occurs (that is, when a device signals the channel-end condition during initiation of the command) and the current CCW contains a nonzero value in bit positions 16-31, indication of an incorrect-length condition is recognized. Command chaining is suppressed unless the SLI flag in the CCW is one and the chain-data flag is zero.

When bit 8 of word 1 is zero, the value of bit 24 is ignored by the channel subsystem, and the subchannel is in the incorrect-length-suppression mode.

***ORB-Extension Control (X):*** When the ORB-extension facility is installed, bit 31 of word 1 specifies whether the ORB is extended. When bit 31 of word 1 is zero, the ORB consists of words 0-2, and words 3-7 are ignored. When bit 31 of word 1 is one, the ORB consists of words 0-7. Words 0 and 1 are described above. Words 2-7 are described below.

***Reserved:*** Bits 25-30 of word 1 are reserved for future use and must be set to zeros. Bit 31 of word 1 must be zero if the ORB-extension facility is not installed. Otherwise, an operand exception or program-check condition is recognized.

***Channel-Program Address:*** Bits 1-31 of word 2 specify the absolute address of the first CCW in main storage. Bit 0 of word 2 must be zero; otherwise, either an operand exception or a program-check condition is recognized. If format-0 CCWs are specified by bit 8 of word 1, then bits 1-7 of word 2 also must be zeros; otherwise, a program-check condition is recognized.

The three rightmost bits of the channel-program address must be zeros, designating the CCW on a doubleword boundary; otherwise, a program-check condition is recognized.

If the channel-program address designates a location protected against fetching or designates a location outside the storage of the particular installation, the start function is not initiated at the device. In this situation, the subchannel becomes status pending with primary, secondary, and alert status.

***Channel-Subsystem (CSS) Priority:*** When the channel-subsystem-I/O-priority facility is installed and bit 31 (X) of word 1 of the ORB is one, byte 0 of word 3 contains an unsigned binary integer, called the channel-subsystem-priority number, that is assigned to the designated subchannel and used to order the selection of subchannels when either a start function or a resume function is to be initiated for one or more subchannels that are start pending or resume pending.

The specified channel-subsystem-priority number can be any number in the range of 0 to 255. The numbers 0 and 255 designate the lowest and highest priorities, respectively.

Depending on the model and the configuration:

1. Fewer than 256 priority levels may be provided. For such models, the ORB-specified priority number may be ignored, and an alternative priority number may be implicitly assigned to the subchannel when the subchannel becomes start pending.
2. When bit 31 (X) of word 1 of the ORB is zero, an implicit priority number is assigned to the subchannel.

See “Channel-Subsystem-I/O-Priority Facility” on page 17-25 for details about how the priority number is assigned for both of these cases.

For models that provide the ORB-extension facility but do not provide the channel-subsystem-I/O-priority facility, byte 0 of word 3 of the ORB must contain zeros; otherwise, either an operand exception or a program-check condition is recognized.

***Control-Unit (CU) Priority:*** When the channel-subsystem-I/O-priority facility is installed and bit 31 (X) of word 1 of the ORB is one, byte 2 of word 3 contains an unsigned binary integer, called the control-unit-priority number, that speci-



LzLabs GmbH. Ex. 1008-688

fies, for an associated control unit attached by a FICON channel path, the priority level that is applied at the associated control unit for all I/O operations associated with the start function.

The specified control-unit-priority number can be any integer in the range of 1 to 255. The numbers 1 and 255 designate the lowest and highest priorities, respectively. The number 0 designates that no priority is assigned to the I/O operations associated with the start function. The handling of I/O operations when the priority number is 0 depends on the control-unit model.

Also depending on the control-unit model, fewer than 255 priority levels may be supported by the control unit. See the control-unit's System Library publication for additional information regarding the range of priority numbers supported and how this priority number is used.

The specified control-unit-priority number is ignored if any of the following conditions exists:

1. Bit 31 (X) of word 1 is zero. In this case, a control-unit-priority number of 0 is transmitted in the associated outbound frames.
2. The designated subchannel is not associated with a control unit configured to a FICON channel path.
3. The associated control unit does not provide prioritized performance of I/O operations. In this case, the control-unit-priority number in the associated outbound frames is ignored at the control unit.
4. The channel-subsystem model does not provide for the transmission of the control-unit-priority number.
5. The channel-subsystem-I/O-priority facility is not operational due to an operator action.

For models that provide the ORB-extension facility but do not provide the channel-subsystem-I/O-priority facility, byte 2 of word 3 of the ORB must contain zeros; otherwise, either an operand exception or a program-check condition is recognized.

***Reserved:*** All fields in the ORB that are defined as either “0” or “Reserved” must contain zeros when START SUBCHANNEL is executed; otherwise, either an operand exception or a program-check condition is recognized.

**Programming Notes:**

1. Bit positions of the ORB that presently are specified to contain zeros may in the future be assigned for the control of new functions.
2. The interruption parameter may contain any information, but ordinarily the information is of significance to the program handling the I/O interruption.

## Channel-Command Word

The channel-command word (CCW) specifies the command to be executed and, for commands initiating certain I/O operations, it designates the storage area associated with the operation, the action to be taken whenever transfer to or from the area is completed, and other options.

A channel program consists of one or more CCWs that are logically linked such that they are fetched and executed by the channel subsystem in either a sequential or a nonsequential order. Sequential (contiguous) CCWs are linked by the use of the chain-data and chain-command flags, and nonsequential (noncontiguous) CCWs are linked by a CCW specifying the transfer-in-channel command.

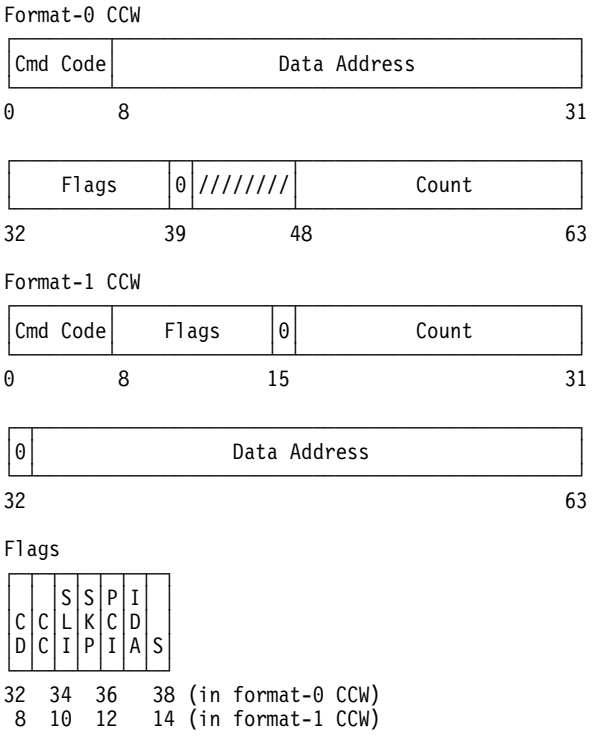
As each CCW is executed, it is recognized as the current CCW. A CCW becomes current (1) when it is the first CCW of a channel program and has been fetched, (2) when, during command chaining, the new CCW is logically fetched, or (3) when, during data chaining, the new CCW takes over control of the I/O operation (see “Data Chaining” on page 15-33). When chaining is not specified, a CCW is no longer current after TEST SUBCHANNEL clears the start-function bit in the subchannel.

The location of the first CCW of the channel program is designated in the ORB that is the operand of START SUBCHANNEL. The first CCW is fetched subsequent to the execution of the instruction. The format of the CCWs fetched by the channel subsystem is specified by bit 8 of word 1 of the ORB. Each additional CCW in the channel program is obtained when the CCW is needed. Fetching of the CCWs by the channel subsystem does not affect those locations in main storage.


LzLabs GmbH. Ex. 1008-689

CCWs have either of two different formats, format 0 or format 1. The two formats do not differ in the information contained in the CCW, but they do differ in the size of the address and the arrangement of fields within the CCW.

The formats are defined as follows:




Format-0 CCWs can be located anywhere in the first $2^{24}$ (16M) bytes of absolute storage, and format-1 CCWs can be located anywhere in the first $2^{31}$ (2G) bytes of absolute storage.

Bit 39 (format 0) or bit 15 (format 1) of every CCW other than a format-0 CCW specifying transfer in channel must be zero. If indirect data addressing is specified and the format-2-IDAW-control bit is zero in the ORB associated with the CCW, then:

1. Bits 30 and 31 (format 0) or 62 and 63 (format 1) of the CCW must be zeros, designating a word boundary,
2. Bit 0 of the first entry of the indirect-data-address list must be zero.

If indirect data addressing is specified and the format-2-IDAW-control bit is one in the ORB associated with the CCW, bits 29-31 (format 0) or bits 61-63 (format 1) of the CCW must be zeros, designating a doubleword boundary. When any of these requirements is not met, a program-check condition may be recognized (see "CCW Indirect Data Addressing" on page 15-36). Detection of this condition during data chaining causes the I/O device to be signaled to conclude the operation. When the absence of these zeros is detected during command chaining or subsequent to the execution of START SUBCHANNEL, the new operation is not initiated, and an interruption condition is generated.

The contents of bit positions 40-47 of a format-0 CCW are ignored.

The fields in the CCWs are defined as follows:

***Command Code:*** Bits 0-7 (both formats) specify the operation to be performed.

***Data Address:*** Bits 8-31 (format 0) or bits 33-63 (format 1) designate a location in absolute storage. The designated location is the first location referred to in the area designated by the CCW. Bit 32 of a format-1 CCW must be zero; otherwise, a program-check condition is recognized. If a byte count of zero is specified, this field is not checked.

See the section "CCW Indirect Data Addressing" on page 15-36 for information about the specification of data addresses greater than 2G bytes.

***Chain-Data (CD) Flag:*** Bit 32 (format 0) or bit 8 (format 1), when one, specifies chaining of data. The bit causes the storage area designated by the next CCW to be used with the current I/O operation. When the CD flag is one in a CCW, the chain-command and suppress-length-indication flags (see below) are ignored.

***Chain-Command (CC) Flag:*** Bit 33 (format 0) or bit 9 (format 1), when one, and when the CD flag and S flag are both zeros, specifies chaining of commands. The bit causes the operation specified by the command code in the next CCW to be initiated upon the normal completion of the current operation.

***Suppress-Length-Indication (SLI) Flag:*** Bit 34 (format 0) or bit 10 (format 1) controls whether an incorrect-length condition is to be indicated to the program. When this bit is one and the CD flag is zero, the incorrect-length indication is suppressed. When both the CC and SLI flags are ones and the



LzLabs GmbH. Ex. 1008-690

CD flag is zero, command chaining takes place, regardless of the presence of an incorrect-length condition. This bit should be specified in *all* CCWs where suppression of the incorrect-length indication is desired.

***Skip (SKP) Flag:*** Bit 35 (format 0) or bit 11 (format 1), when one, specifies the suppression of transfer of information to storage during a read, read-backward, sense ID, or sense operation.

***Program-Controlled-Interruption (PCI) Flag:*** Bit 36 (format 0) or bit 12 (format 1), when one, causes the channel subsystem to generate an intermediate interruption condition when the CCW containing the bit takes control of the I/O operation. When the PCI flag bit is zero, normal operation takes place.

***Indirect-Data-Address (IDA) Flag:*** Bit 37 (format 0) or bit 13 (format 1), when one, specifies indirect data addressing.

***Suspend (S) Flag:*** Bit 38 (format 0) or bit 14 (format 1), when one, specifies suspension of channel-program execution. When valid, it causes channel-program execution to be suspended prior to the execution of the CCW containing the S flag. A one value of the S flag is valid when bit 4 of word 1 of the associated ORB is one.

***Count:*** Bits 48-63 (format 0) or bits 16-31 (format 1) specify the number of bytes in the storage area designated by the CCW.

**Programming Note:** Bit 39 of a format-0 CCW or bit 15 of a format-1 CCW, which presently must be zero, may in the future be assigned for the control of new functions. It is recommended, therefore, that this bit not be set to one for the purpose of obtaining an intentional program-check indication.

## Command Code

The command code, bit positions 0-7 of the CCW, specifies to the channel subsystem and the I/O device the operation to be performed.

The two rightmost bits or, when these bits are zeros, the four rightmost bits of the command code identify the operation to the channel subsystem. The channel subsystem distinguishes among the following four operations:

- Output forward (write, control)
- Input forward (read, sense, sense ID)
- Input backward (read backward)
- Branching (transfer in channel)

The channel subsystem ignores the leftmost bits of the command code, except in a format-1 CCW specifying transfer in channel. In this case, all bits of the command code are decoded by the channel subsystem.

Commands that initiate I/O operations (write, read, read backward, control, sense, and sense ID) cause all eight bits of the command code to be transferred to the control unit. In these command codes, the leftmost bit positions contain modifier bits. The modifier bits specify to the device how the command is to be executed. They may, for example, cause the device to compare data received during a write operation with data previously recorded, and they may specify such attributes as recording density and parity. For the control command, the modifier bits may contain the order code specifying the control function to be performed. The meaning of the modifier bits depends on the type of I/O device and is specified in the System Library publication for the device.

The command-code assignment is listed in Figure 15-5. The symbol x indicates that the bit position is ignored; m identifies a modifier bit.

| Code | Command |
|---|---|
| x x x x 0 0 0 0 | Invalid |
| m m m m m m 0 1 | Write |
| m m m m m m 1 0 | Read |
| m m m m 1 1 0 0 | Read backward |
| m m m m m m 1 1 | Control |
| m m m m 0 1 0 0 | Sense |
| 1 1 1 0 0 1 0 0 | Sense ID |
| x x x x 1 0 0 0 | Transfer in channel[1] |
| 0 0 0 0 1 0 0 0 | Transfer in channel[2] |
| m m m m 1 0 0 0 | Invalid[3] |

**Explanation:**

m Modifier bit

x Ignored

[1] Format-0 CCW

[2] Format-1 CCW

[3] Format-1 CCW with any of bits 0-3 nonzero

*Figure 15-5. Command-Code Assignment*


LzLabs GmbH. Ex. 1008-691

Whenever the channel subsystem detects an invalid command code during the initiation of command execution, the program-check-interruption condition is generated, and channel-program execution is terminated. The command code is ignored during data chaining, unless it specifies transfer in channel.

## Designation of Storage Area

The main-storage area associated with an I/O operation is defined by one or more CCWs. A CCW defines an area by specifying the address of the first byte to be transferred and the number of consecutive bytes contained in the area. The address of the first byte of data to be transferred is specified either directly in the data-address field of the CCW or indirectly in an indirect-data-address word (IDAW) designated by the data-address field of the CCW. The number of bytes contained in the storage area is specified in the count field.

In write, read, control, and sense operations, storage locations are used in ascending order of addresses. As information is transferred to or from main storage, the address from the address field is incremented, and the count from the count field is decremented. The read-backward operation places data in storage in a descending order of addresses, and both the count and the address are decremented. When the count reaches zero, the storage area defined by the CCW is exhausted.

Any main-storage location available to the start function can be used in the transfer of data to or from an I/O device, provided that the location is not protected against that type of reference. Format-0 CCWs can be located in any available part of the first $2^{24}$ (16M) bytes of absolute storage, and format-1 CCWs can be located in any available part of the first $2^{31}$ (2G) bytes of absolute storage, provided that, in both cases, the location is not protected against a fetch-type reference. When the channel subsystem attempts to refer to a protected location, the protection-check condition is generated, and the device is signaled to terminate the operation.

A main-storage location is available if it is available in the configuration and, in the case of a data location (not a CCW or IDAW), access to it is not prevented by the address-limit-checking facility. If a main-storage location is not available, it is said to have an invalid address.

If the channel subsystem refers to a location that is not available, the program-check condition is generated. When the first CCW designated by the channel-program address is at an unavailable location, the start function is not initiated at the device, the status portion of the SCSW is updated with the program-check indication, the subchannel becomes status pending with primary, secondary, and alert status, and deferred condition code 1 is indicated. Invalid data addresses, as well as any invalid CCW addresses detected on chaining or subsequent to the execution of START SUBCHANNEL, cause the channel subsystem to signal the device to conclude the operation the next time the device requests or offers a byte of data or status. In this situation, the subchannel is made status pending with program check indicated in the subchannel status, and the device status is a function of the status received from the device. The program-check condition causes command chaining and command retry to be suppressed.

During an output operation, the channel subsystem may fetch data from main storage before the time the I/O device requests the data. Any number of bytes specified by the current CCW may be prefetched and buffered. When data chaining during an output operation, the channel subsystem may fetch one CCW describing a data area at any time during the execution of the current CCW. If unlimited prefetching is allowed by the setting of the prefetch-control bit in the ORB, any number of CCWs and IDAWs and the associated data may be prefetched by the channel subsystem. When the I/O operation uses data and CCWs from locations near the end of the available storage, such prefetching may cause the channel subsystem to refer to locations that do not exist. Invalid addresses detected during prefetching do not affect the performance of the I/O operation and do not cause error indications until the operation actually attempts to use the information. If the operation is concluded by the I/O device or by the execution of HALT SUBCHANNEL or CLEAR SUBCHANNEL before the invalid information is needed, the condition is not brought to the attention of the program.

The count field in the CCW can specify any number of bytes up to 65,535. In format-0 CCWs,



LzLabs GmbH. Ex. 1008-692

the count field is always nonzero unless the command code specifies transfer in channel, in which case the count field is ignored. In format-1 CCWs, the count field may contain the value zero unless data chaining is specified or the CCW is fetched while data chaining. Whenever (1) the count field in a format-1 CCW is zero, (2) data chaining is either not specified or not in effect, and (3) data transfer is requested by the device, the device is signaled to stop, and the I/O operation is terminated. The channel subsystem sets the incorrect-length condition if the SLI flag is not one in the CCW. No data is transferred. If the device does not request data transfer, the operation proceeds to the normal ending point.

If a zero byte count is contained in a format-0 CCW that does not specify transfer in channel, or if a zero byte count is contained in a format-1 CCW that specifies data chaining or was fetched while data chaining, a program-check condition is recognized, and the subchannel is made status pending with combinations of primary, secondary, and alert status as a function of the state of the subchannel and the status received from the device.

**Note:** For a description of the storage area associated with a CCW when indirect data addressing is used, see "CCW Indirect Data Addressing" on page 15-36.

**Programming Notes:**

1. Since a format-1 CCW with a count of zero is valid, the program can use the CCW count field to specify that no data be transferred to the I/O device. If the device requests a data transfer, the device is signaled to terminate data transfer. If the SLI and chain-command flags are also specified as ones, and no unusual conditions are encountered subsequent to signaling the device to terminate data transfer, the new operation is initiated upon receipt of device end from the device.
2. If the subchannel is in the incorrect-length-suppression mode, the chain-data flag in the current CCW is zero, and the operation is performed as an immediate operation, then incorrect length is not indicated, regardless of the setting of the SLI flag.

   If the subchannel is in the incorrect-length-indication mode, if the chain-data flag in the current CCW is zero, and if the operation is performed as an immediate operation, then incorrect length is indicated if the count field of the current CCW specifies a nonzero value, unless suppressed by the SLI flag of the CCW; incorrect length is not indicated, however, if the count field of the CCW specifies a value of zero.

   If a new CCW that has a count field of zero is fetched during data chaining or if a CCW is fetched with the chain-data flag set to one and a count field of zero, a program-check condition is recognized by the channel subsystem.

## Chaining

When the channel subsystem has completed the transfer of information specified by a CCW, it can continue performing the start function by fetching a new CCW. Such fetching of a new CCW is called chaining, and the CCWs belonging to such a sequence are said to be chained.

Chaining takes place between CCWs located in successive doubleword locations in storage. It proceeds in an ascending order of addresses; that is, the address of the new CCW is obtained by adding 8 to the address of the current CCW. Two chains of CCWs located in noncontiguous storage areas can be coupled for chaining purposes by a transfer-in-channel command. All CCWs in a chain apply to the I/O device that is associated with the subchannel designated by the original START SUBCHANNEL instruction.

Two types of chaining are provided: chaining of data and chaining of commands. Chaining is controlled by the chain-data (CD) and chain-command (CC) flags in conjunction with the suppress-length-indication (SLI) flag in the CCW. These flags specify the action to be taken by the channel subsystem upon the exhaustion of the current CCW and upon receipt of ending status from the device, as shown in Figure 15-6 on page 15-32.

The specification of chaining is effectively propagated through a transfer-in-channel command. When, in the process of chaining, a transfer-in-channel command is fetched, the CCW designated by the transfer-in-channel command is used for the type of chaining specified in the CCW preceding the transfer-in-channel command.


LzLabs GmbH. Ex. 1008-693

The CD and CC flags are ignored in a format-0 CCW specifying the transfer-in-channel command. In a format-1 CCW specifying the transfer-in-channel command, the CD and CC flags must be zeros; otherwise, a program-check condition is recognized.

| Flags in Current CCW | | | Action at the Subchannel upon Exhaustion of Count or Receipt of Channel End | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Immediate Operation | | | | Non-immediate Operation | | |
| | | | Incorrect-Length-Suppression Mode[1] | | Incorrect-Length-Indication Mode | | Count Exhausted | | Count Not Exhausted and CE Received |
| CD | CC | SLI | CCW Count≠0 | CCW Count=0 | CCW Count≠0 | CCW Count=0 | CE Not Received | CE Received | |
| 0 | 0 | 0 | End, NIL | End, NIL | End, IL | End, NIL | Stop, IL | End, NIL | End, IL |
| 0 | 0 | 1 | End, NIL | End, NIL | End, NIL | End, NIL | Stop,NIL | End, NIL | End, NIL |
| 0 | 1 | 0 | CC | CC | End, IL | CC | Stop, IL | CC | End, IL |
| 0 | 1 | 1 | CC | CC | CC | CC | Stop, CC | CC | CC |
| 1 | - | - | End, NIL | PC | End, IL | PC | CD | * | End, IL |

**Explanation:**

- The selected bit is ignored and may be either zero or one.

* These situations cannot validly occur. When data chaining is specified, the new CCW takes control of the operation after transferring the last byte of data designated by the current CCW, but before the next request for data or status transfer from the device. The new CCW (which cannot contain a count of zero unless a program-check condition is also recognized) is in control of the operation.

1 The count field must contain a nonzero value when format-0 CCWs are specified; otherwise, the operation is terminated with a program-check condition.

CC Command chaining is performed by the channel subsystem upon receipt of device end.

CD The chain-data flag causes the channel subsystem to immediately fetch a new CCW for the same operation. The operation continues unless the CCW thus fetched has a count field of zero, in which case the operation is terminated with a program-check condition.

CE Channel end from the device that indicates end of block.

End Operation is terminated.

IL Incorrect length is indicated with the subsequent interruption condition generated at the subchannel.

NIL Incorrect length is not indicated with the subsequent interruption condition generated at the subchannel.

PC These situations cannot validly occur. The channel subsystem recognizes a program-check condition when a CCW is fetched that has the chain-data flag set to one and a count field of zero.

Stop Device is signaled to terminate data transfer, but subchannel remains subchannel active until channel end is received.

*Figure 15-6. Subchannel Chaining Action*



LzLabs GmbH. Ex. 1008-694

**Programming Note:** When bit 9 of word 1 of the ORB is one, unlimited prefetching of chained CCWs (including CCWs linked by a transfer-in-channel command) by the channel subsystem is permitted. When prefetching is allowed by the ORB, no modification of the channel program should be performed after START SUBCHANNEL is executed and before the primary interruption condition for the operation has been received unless the subchannel is currently suspended and is not resume pending.

## Data Chaining

During data chaining, the new CCW fetched by the channel subsystem defines a new storage area for the original I/O operation. If the channel path is of the parallel-I/O-interface type, the performance of the operation at the I/O device is not affected. If the channel path is of the serial-I/O-interface type, then the performance of the operation at the I/O device either is not affected or, depending on the device model, may be terminated with unit-check status. When the operation at the I/O device is not affected and all data designated by the current CCW has been transferred to main storage or to the device, data chaining causes the operation to continue, using the storage area designated by the new CCW. The contents of the command-code field of the new CCW are ignored, unless they specify transfer in channel.

Data chaining is considered to occur immediately after the last byte of data designated by the current CCW has been transferred to main storage or to the device. When the last byte of the data transfer has been placed in main storage or accepted by the device, the new CCW takes over the control of the operation. If the device sends channel end after exhausting the count of the current CCW but before transferring any data to or from the storage area designated by the new CCW, the SCSW associated with the concluded operation pertains to the new CCW.

If programming errors are detected in the new CCW or during its fetching, the error indication is generated, and the device is signaled to conclude the operation when it attempts to transfer data designated by the new CCW. If the device signals the channel-end condition before transferring any data designated by the new CCW, program check or protection check is indicated in the SCSW associated with the termination. The contents of the SCSW pertain to the new CCW unless the address of the new CCW is invalid, the location is protected against fetching, or programming errors are detected in an intervening transfer-in-channel command. A data address referring to a nonexistent or protected area causes an error indication only after the I/O device has attempted to transfer data to or from the invalid location.

Data chaining during an input operation causes the new CCW to be fetched when all data designated by the current CCW has been placed in main storage. On an output operation, the channel subsystem may fetch the new CCW from main storage before data chaining occurs. Any programming errors in the prefetched CCW, however, do not affect the performance of the operation until all data designated by the current CCW has been transferred to the I/O device. If the device concludes the operation before all data designated by the current CCW has been transferred, the conditions associated with the prefetched CCW are not indicated to the program. Unlimited prefetching is allowed under the control of the prefetch bit specified in the ORB. (See “Prefetch Control (P)” on page 15-24.) When unlimited prefetching is not allowed and an output operation is specified, only one CCW describing a data area may be prefetched. If a prefetched CCW specifies transfer in channel, only one more CCW may be fetched before the exhaustion of the current CCW.

**Programming Notes:**

1. If the ORB does not specify unlimited prefetching, no prefetching of CCWs is performed, except in the case of data chaining on an output operation where one CCW describing a data area may be prefetched at a time.

   If the ORB for the I/O operation specifies that prefetching is allowed, any number of CCWs and IDAWs and the associated data areas may be prefetched and buffered in the channel subsystem.

   The same actions for signaling errors and terminating operations take place when unlimited prefetching is allowed by the ORB as when it is not allowed. However, when unlimited prefetching is specified and an error condition is detected, both the channel subsystem and the program must recognize that the points of ter-


LzLabs GmbH. Ex. 1008-695

mination at the channel subsystem and at the I/O device may be different in terms of the channel command in execution at the point of error. The channel subsystem indicates the point of termination at the channel subsystem by storing the appropriate CCW address in word 1 of the subchannel-status word and the point of termination at the device by storing the secondary-CCW address in word 4 of the format-0 extended-status word.

When prefetching has been specified in the ORB, the result of modifications to CCWs after START SUBCHANNEL has been executed or after self-describing channel programs have been used is unpredictable. (See note 2 for the definition of self-describing channel programs.)

2. Data chaining may be used to rearrange information as it is transferred between main storage and an I/O device. Data chaining permits blocks of information to be transferred to or from noncontiguous areas of storage, and, when used in conjunction with the skipping function, data chaining allows the program to place in main storage specified portions of a block of data.

   When, during an input operation, the program specifies data chaining to a location in which data has been placed under the control of the current CCW, the channel subsystem, in fetching the next CCW, fetches the new contents of the location. This is true even if the location contains the last byte transferred under the control of the current CCW. When a channel program data-chains to a CCW placed in storage by the CCW specifying data chaining, the input block is said to be self-describing. A self-describing block contains one or more CCWs that designate storage locations and counts for subsequent data in the same input block.

   The use of self-describing blocks is equivalent to the use of unchecked data. An I/O data-transfer malfunction that affects validity of a block of information is signaled only at the completion of data transfer. The error condition normally does not prematurely terminate or otherwise affect the performance of the operation. Thus, there is no assurance that a CCW read as data is valid until the operation is completed. If the CCW thus read is in error, use of the CCW in the current operation may cause subsequent data to be placed at wrong locations in main storage with resultant destruction of its contents, subject only to the control of the protection key and the address-limit-checking facility, if used.

3. When, during data chaining, a device transfers data by using the data-streaming feature, an overrun or chaining-check condition may be recognized when a small byte-count value is specified in the CCW. The minimum acceptable number of bytes that can be specified varies as a function of the system model and system activity.

## Command Chaining

During command chaining, the new CCW fetched by the channel subsystem specifies a new I/O operation. The channel subsystem fetches the new CCW upon the receipt of the device-end signal for the current operation. If the new CCW does not have its S flag set to one and no unusual conditions are detected, the channel subsystem initiates the new operation. The presence of the S flag set to one or unusual conditions causes command chaining to be suppressed. When command chaining takes place, the completion of the current operation does not cause an I/O interruption, and the count indicating the amount of data transferred during the current operation is not made available to the program. For operations involving data transfer, the new command always applies to the next block of data at the device.

Command chaining takes place and the new operation is initiated only if no unusual conditions have been detected in the current operation. In particular, the channel subsystem initiates a new I/O operation by command chaining upon receipt of a status byte containing only the following bit combinations: (1) device end, (2) device end and status modifier, (3) device end and channel end, and (4) device end, channel end, and status modifier. In the first two cases, channel end is signaled before device end, with all other status bits zeros. If a condition such as attention, unit check, unit exception, incorrect length, program check, or protection check has occurred, the sequence of operations is concluded, and the status associated with the current operation causes an interruption condition to be generated. The new CCW in this case is not fetched. The incorrect-length condition does not suppress command chaining if the current CCW has the SLI flag set to one.



LzLabs GmbH. Ex. 1008-696

An exception to sequential chaining of CCWs occurs when the I/O device presents the status-modifier condition with the device-end signal or channel-end and device-end signals. When command chaining is specified and no unusual conditions have been detected, or when command retry has been previously signaled and an immediate retry could not be performed, the combination of status-modifier and device-end bits causes the channel subsystem to alter the sequential execution of CCWs. If command chaining was specified, status modifier and device end cause the channel subsystem to fetch and chain to the CCW whose main-storage address is 16 higher than that of the CCW that specified chaining. If command retry was previously signaled and immediate retry could not be performed, the status causes the channel subsystem to command chain to the CCW whose storage address is 8 higher than that of the CCW for which retry was initially signaled.

When both command and data chaining are specified, the first CCW associated with the operation specifies the operation to be performed, and the last CCW specifies whether another operation follows.

**Programming Note:** Command chaining makes it possible for the program to initiate transfer of multiple blocks of data by issuing a single START SUBCHANNEL instruction. It also permits a subchannel to be set for execution of other commands, such as positioning the disk-access mechanism, and for data-transfer operations without interference by the program at the end of each operation. Command chaining, in conjunction with the status-modifier condition, permits the channel subsystem to modify the normal sequence of operations in response to signals provided by the I/O device.

## Skipping

Skipping causes the suppression of main-storage references during an I/O operation. It is defined only for read, read-backward, sense-ID, and sense operations, and is controlled by the skip flag, which can be specified individually for each CCW. When the skip flag is one, skipping occurs; when it is zero, normal operation takes place. The setting of the skip flag is ignored in all other operations.

Skipping affects only the handling of information by the channel subsystem. The operation at the I/O device proceeds normally, and information is transferred. The channel subsystem keeps updating the count but does not place the information in main storage. Chaining is not precluded by skipping. In the case of data chaining, normal operation is resumed if the skip flag in the new CCW is zero.

No checking for invalid or protected data addresses takes place during skipping.

**Programming Note:** Skipping, when combined with data chaining, permits the program to place in main storage specified portions of a block of information from an I/O device.

## Program-Controlled Interruption

The program-controlled-interruption (PCI) function permits the program to cause an I/O interruption during the performance of an I/O operation. The function is controlled by the PCI flag of the CCW. Neither the value of the PCI flag nor the associated interruption request affects the performance of the current operation.

The value of the PCI flag can be one either in the first CCW designated for the current start or resume function or in a CCW fetched during chaining. If the PCI flag is one in a CCW that has become current, the subchannel becomes status pending with intermediate status, and an I/O-interruption request is generated. The point at which the subchannel becomes status pending depends on the progress of the current start or resume function as follows:

1. If the PCI flag is one in the first CCW associated with a start function or a resume function, the subchannel becomes status pending with intermediate status only after the command has been accepted.
2. If the PCI flag is one in a CCW that has become current while data chaining, the subchannel becomes status pending with intermediate status after all data designated by the preceding CCW has been transferred.
3. If the PCI flag is one in a CCW that has become current while command chaining, the subchannel becomes status pending with intermediate status as that CCW becomes current.


LzLabs GmbH. Ex. 1008-697

In all cases, if the subchannel is enabled for I/O interruptions, the point of interruption depends on the current activity in the system and may be delayed. No predictable relationship exists between the point at which the interruption request is generated because of the PCI flag and the extent to which data transfer has been completed to or from the area designated by the CCW. However, all the fields within the SCSW pertain to the same instant.

An intermediate interruption condition that is made pending because of a PCI flag remains pending during chaining if not cleared by TEST SUBCHANNEL or CLEAR SUBCHANNEL. If another CCW containing a PCI flag that is one becomes current prior to the clearing of the intermediate interruption condition, only one interruption condition is preserved.

An intermediate interruption may occur while the subchannel is subchannel-and-device active with the operation specified by the CCW causing the intermediate interruption condition or with the operation specified by a CCW that has subsequently become current. If the intermediate interruption condition is not cleared prior to the conclusion of the operation or chain of operations, the condition is indicated together with the primary interruption condition at the conclusion of the operation or chain of operations. The intermediate interruption condition may be cleared by TEST SUBCHANNEL while the subchannel is subchannel active.

If the SCSW stored by TEST SUBCHANNEL indicates that the subchannel is status pending with intermediate status and the operation or chain of operations has not been concluded (that is, the activity-control field indicates subchannel-and-device active or suspended), then the CCW-address field contains an address that is 8 higher than the address of the most recent CCW to become current and have a PCI flag that is one, or the CCW-address field contains an address that is 8 higher than the address of a CCW that has subsequently become current. Unless the SCSW also contains the primary-status bit set to one, the device-status field contains zeros, and the count is unpredictable.

Subchannel-status conditions other than PCI may be indicated when the SCSW is stored. If the subchannel is not also status pending with primary status, these conditions may or may not be indicated again. If the subchannel-status condition is detected while prefetching and the operation or chain of operations is concluded before the condition affects an operation, the condition is reset and is not indicated when the subchannel subsequently becomes status pending with primary status. If the subchannel-status condition affects an operation, the condition is indicated when the subchannel becomes status pending with primary status.

If the program-controlled-interruption condition remains pending until the operation or chain of operations is concluded at the subchannel, a single interruption request exists. When TEST SUBCHANNEL is subsequently executed, the status-control field of the SCSW stored indicates both the primary interruption condition and the intermediate interruption condition, and the PCI bit of the subchannel-status field is one.

The value of the PCI flag is inspected in every CCW except for those CCWs that specify the transfer-in-channel command. The PCI flag is ignored during initial program loading.

**Programming Notes:**

1. The program-controlled interruption provides a means of alerting the program to the progress of chaining during an I/O operation. It permits programmed dynamic main-storage allocation.
2. A CCW with a PCI flag set to one may, if retried because of command retry, cause multiple PCI interruptions to occur. (See “Command Retry” on page 15-42.)

## CCW Indirect Data Addressing

CCW indirect data addressing permits a single channel-command word to control the transfer of data that spans noncontiguous 2K-byte or 4K-byte blocks in main storage. The use of CCW indirect data addressing also allows the program to designate data addresses above 16M bytes when using format-0 CCWs.

CCW indirect data addressing is specified by a flag in the CCW which, when one, indicates that the data address is not used to directly address data. Instead, the address points to a list of words or doublewords, called indirect-data-address words (IDAWs), each of which con-



LzLabs GmbH. Ex. 1008-698

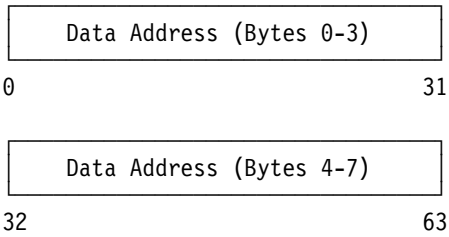
tains an absolute address designating a data area in main storage.

IDAWs have either of two formats, called format 1 and format 2, as determined by the format-2-IDAW control, bit 14 of word 1 of the ORB associated with the channel program being executed. When the format-2-IDAW control is zero, the IDAW is format 1 and is a word containing a 31-bit address. When the control is one, the IDAW is format 2 and is a doubleword containing a 64-bit address. The IDAW formats are as follows:

Format-1 IDAW




Format-2 IDAW




Bit 0 (format 1) is reserved for future use and must be zero; otherwise, a program-check condition may be recognized, as described below.

A format-1 IDAW designates a data area within a 2K-byte block of main storage and is capable of addressing storage in the range of 0 to $2^{31} - 1$.

A format-2 IDAW designates a data area within a 2K-byte or 4K-byte block of main storage, as determined by the 2K-IDAW control, bit 15 of word 1 of the ORB associated with the channel program being executed, and is capable of addressing storage in the range of 0 to $2^{64} - 1$. When the 2K-IDAW-control bit is zero, each format-2 IDAW of the designated channel program designates a 4K-byte block of main storage. When the 2K-IDAW-control bit is one, each format-2 IDAW designates a 2K-byte data-area block. All IDAWs associated with the designated channel program must have the same IDAW format, and all of those IDAWs specify the same size of storage block.

When the indirect-data-addressing bit in the CCW is one, the data-address field of the CCW designates the location of the first IDAW to be used for data transfer for the command. Additional IDAWs, if needed for completing the data transfer for the CCW, are in successive locations in storage. The number of IDAWs required for a CCW is determined by the IDAW format as specified in the ORB, by the count field of the CCW, and by the data address in the initial IDAW. When, for example, (1) the ORB specifies format-2 IDAWs with 4K-byte blocks, (2) the CCW count field specifies 8K bytes, and (3) the first IDAW designates a location in the middle of a 4K-byte block, then three IDAWs are required.

The IDAW designated by the CCW can designate any location. Data is then transferred, for read, write, control, sense ID, and sense commands, to or from successively higher storage locations or, for a read-backward command, to successively lower storage locations, until a 2K-byte block boundary (format-1 or format-2 IDAW) or a 4K-byte block boundary (format-2 IDAW) is reached. The control of data transfer is then passed to the next IDAW. The second and any subsequent IDAWs must designate, depending on the command, the first byte, or the last byte for read backward, of a 2K-byte block (format-1 or format-2 IDAW) or a 4K-byte block (format-2 IDAW). Thus, for read, write, control, sense ID, and sense commands, such format-1 IDAWs must have zeros in bit positions 21-31, and such format-2 IDAWs must have zeros in bit positions 53-63 (2K-byte blocks) or 52-63 (4K-byte blocks). For a read-backward command, such format-1 IDAWs must have ones in bit positions 21-31, and such format-2 IDAWs must have ones in bit positions 53-63 (2K-byte blocks) or 52-63 (4K-byte blocks). If any of these rules is violated, a program-check condition is recognized.

Except for the unique restrictions on the designation of the data address by the IDAW, all other actions taken for the data address, such as for protected storage and invalid addresses, and the actions taken for data prefetching are the same as when indirect data addressing is not used.

IDAWs pertaining to the current CCW or a prefetched CCW may be prefetched. The number of IDAWs that can be prefetched cannot exceed that required to satisfy the count in the CCW that points to the IDAWs. An IDAW takes control of


LzLabs GmbH. Ex. 1008-699

data transfer when the last byte has been transferred for the previous IDAW. The same actions take place as with data chaining regarding when an IDAW takes control of data transfer during an I/O operation. That is, when the count for the CCW has not reached zero, a new IDAW takes control of the data transfer when the last byte has been transferred for the previous IDAW for that CCW, even in situations where (1) channel end, (2) channel end and device end, or (3) channel end, device end, and status modifier are received prior to the transfer of any data bytes pertaining to the new IDAW.

A prefetched IDAW does not take control of an I/O operation if the count in the CCW has reached zero with the transfer of the last byte of data for the previous IDAW for that CCW. Program or access errors detected in prefetched IDAWs are not indicated to the program until the IDAW takes control of data transfer. However, when the channel subsystem detects an invalid CBC on the contents of a prefetched IDAW or its associated key, the condition may be indicated to the program, when detected, before the IDAW takes control of data transfer. For a description of the indications provided when an invalid CBC is detected on the contents of an IDAW or its associated key, see "Channel-Control Check" on page 16-27.

Bits 1-31 (format 1) or bits 0-63 (format 2) designate the absolute storage location of the first byte to be used in the data transfer.

When the IDAW flag of the CCW is set to one and any of the following conditions occurs:

1. Format-1 IDAWs are specified in the ORB, and the address in the CCW does not designate the first IDAW on a word boundary,
2. Format-2 IDAWs are specified in the ORB, and the address in the CCW does not designate the first IDAW on a doubleword boundary,
3. The address in the CCW designates a storage location that is not physically available,
4. Access to the storage location specified by the address in the CCW is prohibited by protection, or
5. Bit 0 (format 1 only) of the first IDAW is not zero,

then, depending on the model, one of the following two actions is taken independent of the setting of the skip flag (if condition 5 above is true, action 2 must be taken).

1. The above conditions are checked before initiating the operation at the device. If any of these conditions is recognized, initiation of the I/O operation does not occur, and the subchannel is made status pending with primary, secondary, and alert status.
2. The operation is initiated at the device prior to checking for these conditions. If the device attempts to transfer data, the device is signaled to terminate the I/O operation, and the subchannel is made status pending with primary, secondary, and alert status as a function of the subchannel state and the status presented by the device.

## Suspension of Channel-Program Execution

The suspend function, when used in conjunction with RESUME SUBCHANNEL, provides the program with a means to stop and restart the execution of a channel program. The initiation of the suspend function is controlled by the setting of the suspend control, bit 4 of word 1 of the ORB. The suspend function is signaled when suspend control has been specified for the subchannel in the ORB and a CCW containing an S flag set to one becomes the current CCW. The flag can be indicated either in the first CCW of the channel program or in a CCW fetched while command chaining. The S flag is not valid and causes a program-check condition to be recognized if (1) the ORB contains the suspend-control bit set to zero, or (2) the CCW is fetched while data chaining (see "Data Chaining" on page 15-33, concerning the handling of programming errors detected during data chaining).

Upon recognition of the suspend function, suspension of channel-program execution occurs when the CCW becomes current (see "Channel-Command Word" on page 15-27, for a definition of when a CCW becomes current). If suspension occurs during command chaining, the device is signaled that command chaining is no longer in effect.



LzLabs GmbH. Ex. 1008-700

RESUME SUBCHANNEL signals that the CCW that caused channel-program suspension may have been modified, that the CCW must be refetched, and that the contents of the CCW must be examined to determine the settings of the flags. If the S flag is one, execution of that CCW does not occur. If the CCW is valid and the S flag in the CCW is zero, execution is initiated (see "RESUME SUBCHANNEL" on page 14-10 and "Start Function and Resume Function" on page 15-18).

When a valid CCW that contains an S flag validly set to one becomes the current CCW during command chaining and the resume-pending condition is not recognized, the suspend function is performed and causes the following actions to occur in the order given:

1. The device is signaled that the chain of operations has been concluded.
2. Channel-program execution is suspended at the subchannel; all prefetched IDAWs, CCWs, and data are discarded; and the subchannel is set up such that the resume function can be performed when the subchannel is next recognized to be resume pending.
3. If the measurement-block-update mode is active and the subchannel is enabled for the mode, the accrued values of the measurement data, including the start-subchannel and sample count, are added to the accumulated values in the measurement block for the subchannel. The start-subchannel count is the only measurement data that is updated in the measurement block if the channel-subsystem-timing facility is not available for the subchannel. (See "Channel-Subsystem Monitoring" on page 17-1 for more information.)

   If a measurement-check condition is detected during the measurement-block update, the channel program is terminated at the subchannel. The subchannel is made status pending with primary, secondary, and alert status, the device-status and subchannel-status fields are set to zero, and one of the measurement-check conditions is indicated in the extended-status flags of the format-0 ESW. The subchannel is not placed in the suspended state. (See "Subchannel-Control Field" on page 16-11.)
4. The subchannel is placed in the suspended state.
5. If the subchannel is not resume pending at this point, the intermediate interruption condition due to suspension is recognized if the suppress-suspended-interruption bit of the ORB is zero; otherwise, the resume function is performed.

When a valid CCW that contains an S flag validly set to one becomes the current CCW during command chaining and the resume-pending condition is recognized, the resume function is performed instead of the suspend function.

When the first CCW of a channel program contains an S flag validly set to one and the resume-pending condition is not recognized, the suspend function is performed and causes the following actions to occur in the order given:

1. Channel-program execution is suspended prior to the selection of the device.
2. The subchannel is set up such that the resume function can be performed when the subchannel is next recognized to be resume pending.
3. If the measurement-block-update mode is active and the subchannel is enabled for the mode, the SSCH+RSCH count is incremented, and the accrued function-pending time (a function of the setting of the timing-facility bit) is added to the accumulated value in the measurement block for the subchannel.

   If a measurement-check condition is detected during the measurement-block update, the channel program is not started at the subchannel. The subchannel is made status pending with primary, secondary, and alert status. Deferred condition code one is set, and the start-pending bit remains set to one. The device-status and subchannel-status fields are set to zero, and one of the measurement-check conditions is indicated in the extended-status flags of the format-0 ESW. The subchannel is not placed in the suspended state. (See "Subchannel-Control Field" on page 16-11.)
4. The subchannel is placed in the suspended state.
5. If the subchannel is not resume pending at this point, the subchannel is made status


LzLabs GmbH. Ex. 1008-701

pending with intermediate status due to suspension if the suppress-suspended-interruption-control bit of the ORB is zero; otherwise, the resume function is performed.

When the first CCW of a channel program contains an S flag validly set to one and the resume-pending condition is recognized, the resume function is performed instead of the suspend function.

**Programming Notes:**

1. The execution of MODIFY SUBCHANNEL and START SUBCHANNEL completes with condition code 2 set if the designated subchannel is suspended. The start function is indicated at the subchannel while the subchannel is in the suspended state.
2. In certain situations, normal resumption of the execution of a channel program that has been suspended may not be desired. Normal termination of the suspended channel-program execution may be accomplished by:
   a. Executing HALT SUBCHANNEL and designating the subchannel.
   b. Modifying the CCWs in storage such that, when channel-program execution is resumed, the command transferred to the device is a control command with all modifier bits specified as zeros (no-operation) and with the chain-command flag specified as zero; and then executing RESUME SUBCHANNEL.
   c. When an IRB indicates measurement check along with zero device status, zero subchannel status, and status pending with primary, secondary, and alert status, it may indicate that the measurement check was detected during an attempt to place the subchannel into the suspended state.
3. If the suspended interruption is suppressed, the N condition and DCTI values applicable to the preceding subchannel-active period are not made available to the program. The execution of RESUME SUBCHANNEL when the subchannel is in the suspended state causes path-not-operational conditions and the N condition to be reset to zeros. Path-not-operational conditions and the N condition are not reset when RESUME SUBCHANNEL is executed and the designated subchannel is not in the suspended state.

## Commands and Flags

Figure 15-7 lists the command codes for the seven commands and indicates which flags are defined for each command. Except for a format-1 CCW specifying transfer in channel, the flags are ignored for all commands for which they are not defined. The flags are reserved in a format-1 CCW specifying transfer in channel and must be zeros.

| Name | Code | Flags |
|---|---|---|
| Write | M M M M M M 0 1 | CD CC SLI PCI IDA S |
| Read | M M M M M M 1 0 | CD CC SLI SK PCI IDA S |
| Read backward | M M M M 1 1 0 0 | CD CC SLI SK PCI IDA S |
| Control | M M M M M M 1 1 | CD CC SLI PCI IDA S |
| Sense | M M M M 0 1 0 0 | CD CC SLI SK PCI IDA S |
| Sense ID | 1 1 1 0 0 1 0 0 | CD CC SLI SK PCI IDA S |
| Transfer in channel | X X X X 1 0 0 0 | (See note below) |

**Explanation:**

| | |
|---|---|
| CC | Chain command |
| CD | Chain data |
| IDA | Indirect data addressing |
| M | Modifier bit |
| PCI | Program-controlled interruption |
| S | Suspend |
| SK | Skip |
| SLI | Suppress-length indication |
| X | Ignored in a format-0 CCW; must be zero in a format-1 CCW |

**Note:** Flags are ignored in a format-0 transfer-in-channel CCW and must be zeros in a format-1 transfer-in-channel CCW.

*Figure 15-7. Command Codes and Flags*

All flags have individual significance, except that the CC and SLI flags are ignored when the CD flag is set to one, and, for output forward operations the SK flag is ignored. The presence of the SLI flag is ignored for immediate operations involving format-0 CCWs, in which case the incorrect-length indication is suppressed regardless of the setting of the flag. The incorrect-length indication may be suppressed for immediate operations when executing a format-1 CCW, depending on the incorrect-length-suppression mode. The PCI flag is ignored during initial program loading. All flags, except the PCI flag, are ignored when the S flag is one.



LzLabs GmbH. Ex. 1008-702

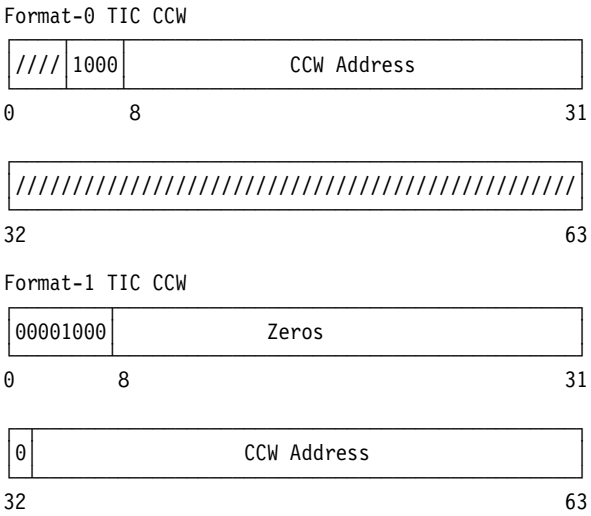
**Programming Notes:**

1. A malfunction that affects the validity of data transferred in an I/O operation is signaled at the end of the operation by means of unit check or channel-data check, depending on whether the device (control unit) or the channel subsystem detected the error. In order to make use of the checking facilities provided in the system, data read in an input operation should not be used until the end of the operation has been reached and the validity of the data has been checked. Similarly, on writing, the copy of data in main storage should not be destroyed until the program has verified that no malfunction affecting the transfer and recording of data was detected.
2. An error condition may be recognized and the I/O operation terminated when 256 or more chained commands are executed with a device and none of the executed commands result in the transfer of any data. When this condition is recognized, program check is indicated.
3. All CCWs that require suppression of incorrect-length indications must use the SLI flag.

## Branching in Channel Programs

The channel subsystem provides two methods to modify the normal sequential execution of the CCWs in a channel program. One is the transfer-in-channel (TIC) command (described in "Transfer in Channel"), which can be used to loop back to a previously executed CCW, or to connect discontiguous segments of the channel program. The other method, which uses the status-modifier device-status bit (described in the publication *ESA/390 Common I/O-Device Commands*, SA22-7204), allows conditions at the device to cause the channel to bypass the next CCW in the channel program.

### Transfer in Channel




The next CCW is fetched from the location in absolute main storage designated by the data-address field of the CCW specifying transfer in channel. The transfer-in-channel command does not initiate any I/O operation, and the I/O device is not signaled of the execution of the command. The purpose of the transfer-in-channel command is to provide chaining between CCWs not located in adjacent doubleword locations in an ascending order of addresses. The command can occur in both data and command chaining.

Bits 29-31 (format 0) or bits 61-63 (format 1) of a CCW that specifies the transfer-in-channel command must be zeros, designating a CCW on a doubleword boundary. Furthermore, a CCW specifying transfer in channel may not be fetched from a location designated by an immediately preceding transfer in channel. When either of these errors is detected or when an invalid address is designated in the transfer-in-channel command, the program-check condition is generated. When a CCW that specifies the transfer-in-channel command designates a CCW at a location protected against fetching, the protection-check condition is generated. Detection of these errors during data chaining causes the operation at the I/O device to be terminated and an interruption condition to be generated, while during command chaining it causes only an interruption condition to be generated.

The contents of the second half of the format-0 CCW, bit positions 32-63, are ignored. Similarly,


LzLabs GmbH. Ex. 1008-703

the contents of bit positions 0-3 of the format-0 CCW are ignored.

Bit positions 0-3 and 8-32 of the format-1 CCW must contain zeros; otherwise, a program-check condition is generated.

## Command Retry

The channel subsystem has the capability to perform command retry, a procedure that causes a command to be retried without requiring an I/O interruption. This retry is initiated by the control unit presenting either of two status-bit combinations by means of a special sequence. When immediate retry can be performed, it presents a channel-end, unit-check, and status-modifier status-bit combination, together with device end. When immediate retry cannot be performed, the presentation of device end is delayed until the control unit is prepared. When device end is presented alone, the previous command is transferred again. If device end is accompanied by status modifier, command retry is not performed, and the channel subsystem command-chains to the CCW following the one for which command retry was signaled (for information on status modifier, see the publication *ESA/390 Common I/O-Device Commands*, SA22-7204). When the channel subsystem is not capable of performing command retry due to an error condition, or when any status bit other than device end or device end and status modifier accompanies the requested command-retry initiation, the retry is suppressed, and the subchannel becomes status pending. The SCSW stored by TEST SUBCHANNEL contains the status provided by the I/O device.

**Programming Note:** The following possible results of a command retry must be anticipated by the program:

1. A CCW containing a PCI may, if retried because of command retry, cause multiple PCI interruptions to occur.
2. If a CCW used in an operation is changed before that operation has been successfully completed, the results are unpredictable.

## Concluding I/O Operations before Initiation

Subsequent to the execution of START SUBCHANNEL or RESUME SUBCHANNEL and before the first command is accepted, the start function can be ended at the subchannel by CANCEL SUBCHANNEL (if the instruction is installed), CLEAR SUBCHANNEL, or HALT SUBCHANNEL. If the I/O operation is ended by CANCEL SUBCHANNEL, there is no subsequent interruption condition from the I/O operation, and the subchannel is available for the initiation of another start function. However, the device may have signaled a busy condition while the canceled operation was start pending. In this case, the device owes a no-longer-busy signal to the channel subsystem. This may result in unsolicited device-end status before the next operation is initiated at the device. (See also “Clear Function” on page 15-14 and “Halt Function” on page 15-15.)

## Concluding I/O Operations during Initiation

After the designated subchannel has been determined to be in a state such that START SUBCHANNEL can be executed, certain tests are performed on the validity of the information specified by the program and on the logical availability of the associated device. This testing occurs during or subsequent to the execution of START SUBCHANNEL and during command chaining and command retry.

A data-transfer operation is initiated at the subchannel and device only when no programming or equipment errors are detected by the channel subsystem and when the device responds with zero status during the initiation sequence. When the channel subsystem detects or the device signals any unusual condition during the initiation of an I/O operation, the command is said to be not accepted. In this case, the subchannel becomes status pending with primary, secondary, and alert status. Deferred condition code 1 is set, and the start-pending bit remains set to one.

Conditions that preclude the initiation of an I/O operation are detailed in the SCSW stored by TEST SUBCHANNEL. In this situation, the device is not started, no interruption conditions are generated subsequent to TEST SUBCHANNEL, and the



LzLabs GmbH. Ex. 1008-704

subchannel is idle. The device is immediately available for the initiation of another operation, provided the command was not rejected because of the busy or not-operational condition.

When an unusual condition causes a command to be not accepted during the initiation of an I/O operation by command chaining or command retry, an interruption condition is generated, and the subchannel becomes status pending with combinations of primary, secondary, and alert status as a function of the status signaled by the device. The status describing the condition remains at the subchannel until cleared by TEST SUBCHANNEL. The conditions are indicated to the program by means of the corresponding status bits in the SCSW. A path-not-operational condition recognized during command chaining is signaled to the program by means of an interface-control-check indication. The new I/O operation at the device is not started.

START SUBCHANNEL is executed independent of its associated device. Tests on most program-specified information, on device availability and unit status, and on most error conditions are performed subsequent to the execution of START SUBCHANNEL. When any conditions are detected that preclude the performance of the start function, an interruption condition is generated by the channel subsystem and placed at the subchannel, causing it to become status pending.

## Immediate Conclusion of I/O Operations

During the initiation of an I/O operation, the device can accept the command and signal the channel-end condition immediately upon receipt of the command code. An I/O operation causing the channel-end condition to be signaled during the initiation sequence is called an *immediate operation*. Status generated by the device for the immediate command, when command chaining is not specified and command retry is not signaled, causes the subchannel to become status pending with combinations of primary, secondary, intermediate, and alert status as a result of information specified in the ORB and CCW and status presented by the device. If the immediate operation is the first operation of the channel program, deferred condition code 1 is set and accompanies the status indications. If intermediate status is indicated, the indication can occur only as a result of the CCW having the PCI flag set to one (see "Program-Controlled Interruption" on page 15-35).

Whenever command chaining is specified after an immediate operation and no unusual conditions have been detected during the operation, or when command retry occurs for an immediate operation, an interruption condition is not generated. The subsequent commands in the chain are handled normally, and, usually, the channel-end condition for the last CCW generates a primary interruption condition. If device end is signaled with channel end, a secondary interruption condition is also generated.

Whenever immediate completion of an I/O operation is signaled, no data has been transferred to or from the device, and the data address in the CCW is not checked for validity. If the subchannel is in the incorrect-length-suppression mode, incorrect length is not indicated to the program, and command chaining is performed when specified. If the subchannel is in the incorrect-length-indication mode, incorrect length and command chaining are under control of the SLI and chain-command flags. The conditions that cause the incorrect-length indication to be suppressed are summarized in Figure 15-6 on page 15-32.

**Programming Note:** I/O operations for which the entire operation is specified in the command code may be performed as immediate operations. Whether the command is executed as an immediate operation depends on the operation and type of device.

## Concluding I/O Operations during Data Transfer

When the subchannel has been passed the contents of an ORB, the subchannel is said to be start pending. When the I/O operation has been initiated and the command has been accepted, the subchannel becomes subchannel-and-device active and remains in that state unless (1) the channel subsystem detects an equipment malfunction, (2) the operation is concluded by the execution of CLEAR SUBCHANNEL or HALT SUBCHANNEL, or (3) status that causes a primary interruption condition to be recognized (usually channel end) is accepted from the device. When command chaining and command retry are not


LzLabs GmbH. Ex. 1008-705

specified or when chaining is suppressed because of unusual conditions, the status that is recognized as primary status causes the operation at the subchannel to be concluded and an interruption condition to be generated. The status bits in the associated SCSW indicate primary status and the unusual conditions, if any. The device can present status that is recognized as primary status at any time after the initiation of the I/O operation, and the presentation of status may occur before any data has been transferred.

For operations not involving data transfer, the device normally controls the timing of the channel-end condition. The duration of data-transfer operations may be variable and may be controlled by the device or the channel subsystem.

Excluding equipment errors and the execution of the CLEAR SUBCHANNEL, HALT SUBCHANNEL, and RESET CHANNEL PATH instructions, the channel subsystem signals the device to conclude the performance of an I/O operation during data transfer whenever any of the following conditions occurs:

- The storage areas designated for the operation are exhausted or filled.
- A program-check condition is detected.
- A protection-check condition is detected.
- A chaining-check condition is detected.
- A channel-control-check condition is detected that does not affect the control of the I/O operation.

The first of these conditions occurs when the channel subsystem has decremented the count to zero in the last CCW associated with the operation. A count of zero indicates that the channel subsystem has transferred all information specified by the I/O operation. The other four conditions are due to errors and cause premature conclusion of data transfer. In either case, the conclusion is signaled in response to a service request from the device and causes data transfer to cease. If the device has no blocks defined for the operation (such as writing on magnetic tape), it concludes the operation and presents channel-end status.

The device can control the duration of an operation and the timing of channel end by blocking of data. On certain operations for which blocks are defined (such as reading on magnetic tape), the device does not present channel-end status until the end of the block is reached, regardless of whether the device has been previously signaled to conclude data transfer.

Checking for the validity of the data address is performed only as data is transferred to or from main storage. When the initial data address in the CCW is invalid, no data is transferred during the operation, and the device is signaled to conclude the operation in response to the first service request. On writing, devices such as magnetic-tape units request the first byte of data before any mechanical motion is started, and, if the initial data address is invalid, the operation is terminated by the channel subsystem before the recording medium has been advanced. However, since the operation has been initiated at the device, the device presents channel-end status, causing the channel subsystem to recognize a primary interruption condition. Subsequently, the device also presents device-end status, causing the channel subsystem to recognize a secondary interruption condition. Whether a block at the device is advanced when no data is transferred depends on the type of device.

When command chaining takes place, the subchannel is in the subchannel-and-device-active state from the time the first I/O operation is initiated at the device until the device presents channel-end status for the last I/O operation of the chain. The subchannel remains in the device-active state until the device presents the device-end status for the last I/O operation of the chain.

Any unusual conditions cause command chaining to be suppressed and a primary interruption condition to be generated. The unusual conditions can be detected by either the channel subsystem or the device, and the device can provide the indications with channel end, control unit end, or device end. When the channel subsystem is aware of the unusual condition by the time the channel-end status for the operation is accepted, the chain is ended as if the operation during which the condition occurred were the last operation of the chain. The device-end status is recognized as a secondary interruption condition whether presented together with the channel-end status or separately. If the device presents unit check or unit exception together with either control unit end



LzLabs GmbH. Ex. 1008-706

or device end as status that causes the channel subsystem to recognize the primary interruption condition, then the subchannel-and-device-active state of the subchannel is terminated, and the subchannel is made status pending with primary, secondary, and alert status. Intermediate status may also be indicated if an intermediate interruption condition previously existed at the subchannel for the initial-status-interruption condition or the PCI condition and that condition still remains pending at the subchannel. The channel-end status that was presented to the channel subsystem previously when command chaining was signaled is not made available to the program.

## Channel-Path-Reset Function

Subsequent to the execution of RESET CHANNEL PATH, the channel-path-reset function is performed. The performance of the function consists of: (1) issuing the reset signal on the designated channel path and (2) causing a channel report to be made pending, indicating the completion of the channel-path-reset function.

## Channel-Path-Reset-Function Signaling

The channel subsystem issues the reset signal on the designated channel path. As part of this operation, the following actions are taken:

1. All internal indications associated with control-unit-busy, device-busy, and allegiance conditions for the designated channel path are reset. These indications are reset at all subchannels that have access to the designated channel path. The reset function has no other effect on subchannels, including those having I/O operations in progress.
2. If the channel path fails to respond properly to the reset signal (see "I/O-System Reset" on page 17-13 for a detailed description) or, because of a malfunction, the reset signal could not be issued, the channel path is made physically not available at each applicable subchannel.
3. If an I/O operation is in progress at the device and the device is actively communicating on the channel path in the performance of that I/O operation when the reset signal is received on that channel path, the I/O operation is reset, and the control unit and device immediately terminate current communication with the channel subsystem. (To avoid possible misinterpretation of unsolicited device-end status, programming measures can be taken as described in programming note 2 on page 15-46.)
4. If an I/O operation is in progress in the multipath mode at the device and the device is not currently communicating over the channel path in the performance of that I/O operation when the reset signal is received, then the I/O operation may or may not be reset depending on whether another channel path is available for selection in the same multipath group for the device. If there is at least one other channel path in the multipath group for the device that is available for selection, the I/O operation is not reset. However, the channel path on which the system reset is received is removed from the current set of channel paths that form the multipath group. If the channel path on which the reset signal is received is the only channel path of a multipath group, or if the device is operating in the single-path mode, the I/O operation is reset.
5. The channel-path-reset function causes I/O operations to be terminated at the device as described above; however, I/O operations are *never* terminated at the subchannel by the channel-path-reset function.

If an I/O operation is in progress at the subchannel and the channel path designated for the performance of the channel-path-reset function is being used for that I/O operation, the subchannel may or may not accurately reflect the progress of the I/O operation up to that instant. The subchannel remains in the state that exists at the time the channel-path-reset function is performed until the state is changed because of some action taken by the program or by the device.

## Channel-Path-Reset-Function-Completion Signaling

After the reset signal has been issued and an attempt has been made to issue the reset signal, or after it has been determined that the reset signal cannot be issued, the channel-path-reset function is completed. (See "Reset Signal" on page 17-13.)


LzLabs GmbH. Ex. 1008-707

As a result of the channel-path-reset function being performed, a channel report is made pending (see “Channel-Subsystem Recovery” on page 17-21) to report the results. If the channel path responds properly to the system-reset signal, the channel report indicates that the channel path has been initialized and is physically available for use. If the reset signal was issued but either the channel path failed to respond properly or the channel path was already not physically available at each subchannel having access to the channel path, the channel report indicates that the channel path has been initialized but is not physically available for use. If, because of a malfunction or because the designated channel path is not in the configuration, the reset signal could not be issued, the channel report indicates that the channel path has not been initialized and is not physically available for use.

**Programming Notes:**

1. If an I/O operation is in progress in the multipath mode when the channel-path-reset function is performed on a channel path of the multipath group, it is possible for the I/O operation to be continued on a remaining channel path of the group.
2. When the performance of the channel-path-reset function causes the I/O operation at the device to be reset, unsolicited device-end status presented by the device, if any, may be erroneously interpreted by the channel subsystem to be chaining status and thus cause the channel subsystem to continue the chain of commands. If this situation occurs, then the device-end status is not made available to the program, and the device is selected again by the channel subsystem; however, the device may interpret the initiation sequence as the beginning of a new channel program instead of as command chaining. This possibility can be avoided by issuing CLEAR SUBCHANNEL or HALT SUBCHANNEL, designating the affected subchannels, prior to issuing RESET CHANNEL PATH.
3. The performance of the channel-path-reset function may, on some models, cause overruns to occur on other channel paths.
4. Even though reset is signaled on the designated channel path, allegiances to that channel path by one or more devices may not have been reset because of a malfunction at a control unit or a malfunction at the physical channel path to the control unit.



LzLabs GmbH. Ex. 1008-708

# Chapter 16. I/O Interruptions


Interruption Conditions . . . . . 16-2
Intermediate Interruption Condition . . . . 16-4
Primary Interruption Condition . . . . . 16-4
Secondary Interruption Condition . . . . . 16-4
Alert Interruption Condition . . . . . 16-4
Priority of Interruptions . . . . . 16-4
Interruption Action . . . . . 16-5
Interruption-Response Block . . . . . 16-6
Subchannel-Status Word . . . . . 16-6
Subchannel Key . . . . . 16-8
Suspend Control (S) . . . . . 16-8
Extended-Status-Word Format (L) . . . 16-8
Deferred Condition Code (CC) . . . . . 16-8
Format (F) . . . . . 16-10
Prefetch (P) . . . . . 16-10
Initial-Status-Interruption Control (I) . . 16-11
Address-Limit-Checking Control (A) . 16-11
Suppress-Suspended Interruption (U) .16-11
Subchannel-Control Field . . . . . 16-11
Zero Condition Code (Z) . . . . . 16-11
Extended Control (E) . . . . . 16-11
Path Not Operational (N) . . . . . 16-12
Function Control (FC) . . . . . 16-12
Activity Control (AC) . . . . . 16-13
Status Control (SC) . . . . . 16-16
CCW-Address Field . . . . . 16-18
Device-Status Field . . . . . 16-23
Subchannel-Status Field . . . . . 16-23
Program-Controlled Interruption . . . . 16-23
Incorrect Length . . . . . 16-23
Program Check . . . . . 16-24
Protection Check . . . . . 16-26
Channel-Data Check . . . . . 16-26
Channel-Control Check . . . . . 16-27
Interface-Control Check . . . . . 16-28
Chaining Check . . . . . 16-29
Count Field . . . . . 16-29
Extended-Status Word . . . . . 16-32
Extended-Status Format 0 . . . . . 16-32
Subchannel Logout . . . . . 16-32
Extended-Report Word . . . . . 16-36
Failing-Storage Address . . . . . 16-37
Secondary-CCW Address . . . . . 16-38
Extended-Status Format 1 . . . . . 16-38
Extended-Status Format 2 . . . . . 16-38
Extended-Status Format 3 . . . . . 16-39
Extended-Control Word . . . . . 16-40
Extended-Measurement Word . . . . . 16-40


When an I/O operation or sequence of I/O operations initiated by the execution of START SUBCHANNEL is ended, the channel subsystem and the device generate status conditions. The generation of these conditions can be brought to the attention of the program by means of an I/O interruption or by means of the execution of the TEST PENDING INTERRUPTION instruction. (During certain abnormal situations, these conditions can be brought to the attention of the program by means of a machine-check interruption. See “Channel-Subsystem Recovery” on page 17-21 for details.) The status conditions, as well as an address and a count indicating the extent of the operation sequence, are presented to the program in the form of a subchannel-status word (SCSW). The SCSW is stored in an interruption-response block (IRB) during the execution of TEST SUBCHANNEL.

Normally an I/O operation is being performed until the device signals primary interruption status. Primary interruption status can be signaled during initiation of an I/O operation, or later. An I/O operation can be terminated by the channel subsystem performing a clear or halt function when it detects an equipment malfunction, a program check, a chaining check, a protection check, or an incorrect-length condition, or by performing a clear, halt, or channel-path-reset function as a result of the execution of CLEAR SUBCHANNEL, HALT SUBCHANNEL, or RESET CHANNEL PATH, respectively.

I/O interruptions provide a means for the CPU to change its state in response to conditions that occur at I/O devices or subchannels. These conditions can be caused by the program, by the channel subsystem, or by an external event at the device.

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-709

## Interruption Conditions

The conditions causing requests for I/O interruptions to be initiated are called I/O-interruption conditions. When an interruption condition is recognized by the channel subsystem, it is indicated at the appropriate subchannel. The subchannel is then said to be status pending. The subchannel becoming status pending causes the channel subsystem to generate an I/O-interruption request. An I/O-interruption request can be brought to the attention of the program only once.

An I/O-interruption request remains pending until it is accepted by a CPU in the configuration, is withdrawn by the channel subsystem, or is cleared by means of the execution of TEST PENDING INTERRUPTION, TEST SUBCHANNEL, or CLEAR SUBCHANNEL, or by means of subsystem reset. When a CPU accepts an interruption request and stores the associated interruption code, the interruption request is cleared. Alternatively, an I/O-interruption request can be cleared by means of the execution of TEST PENDING INTERRUPTION. In either case, the subchannel remains status pending until the associated interruption condition is cleared when TEST SUBCHANNEL or CLEAR SUBCHANNEL is executed or when the subchannel is reset.

An I/O-interruption condition is normally cleared by means of the execution of TEST SUBCHANNEL. If TEST SUBCHANNEL is executed, designating a subchannel that has an I/O-interruption request pending, both the interruption request and the interruption condition at the subchannel are cleared. The interruption request and the interruption condition can also be cleared by CLEAR SUBCHANNEL.

A device-end status condition generated by the I/O device and presented following the conclusion of the last I/O operation of a start function is reset at the subchannel by the channel subsystem without generating an I/O-interruption condition or I/O-interruption request if the subchannel is currently start pending and if the status contains device end either alone or accompanied by control unit end. If any other status bits accompany the device-end status bit, then the channel subsystem generates an I/O-interruption request with deferred condition code 1 indicated.

When an I/O operation is terminated because of an unusual condition detected by the channel subsystem during the command-initiation sequence, status describing the interruption condition is placed at the subchannel, causing it to become status pending. If the unusual condition is detected by the device, the device-status field of the associated SCSW identifies the condition.

When command chaining takes place, the generation of status by the device does not cause an interruption, and the status is not made available to the program.

When the channel subsystem detects any of the following interruption conditions, it initiates a request for an I/O interruption without necessarily communicating with, or having received the status byte from, the device:

- A programming error associated with the contents of the ORB passed to the subchannel by the previous execution of START SUBCHANNEL
- A suspend flag set to one in the first CCW fetched that initiates channel-program execution for either START SUBCHANNEL or RESUME SUBCHANNEL, and suppress suspended interruption not specified in the ORB
- A programming error associated with the first CCW or first IDAW

These interruption conditions from the subchannel, except for the suspended condition, can be accompanied by other subchannel-status indications, but the device-status indications are all stored as zeros.

The channel subsystem issues the clear signal to the device when status containing unit check is presented to a subchannel that is disabled or when the device is not associated with any subchannel. However, if the presented status does not contain unit check, the status is accepted by the channel subsystem and discarded without causing the subchannel to become status pending.

An interruption condition caused by the device may be accompanied by multiple device-status conditions. Furthermore, more than one interruption condition associated with the same device can be accepted by the channel subsystem without an intervening I/O interruption. As an



LzLabs GmbH. Ex. 1008-710

example, when the channel-end condition is not cleared at the device by the time device end is generated, both conditions may be cleared at the device concurrently and indicated in the SCSW together. Alternatively, channel-end status may have been previously accepted at the subchannel, and an I/O interruption may have occurred; however, the associated status-pending condition may not have been cleared by TEST SUBCHANNEL by the time device-end status was accepted at the subchannel. In this situation, the device-end status may be merged with the channel-end status without causing an additional I/O interruption. Whether an interruption condition may be merged at the subchannel with other existing interruption conditions depends upon whether the interruption condition is unsolicited or solicited.

***Unsolicited Interruption Condition:*** An unsolicited interruption condition is any interruption condition that is unrelated to the performance of a clear, halt, resume, or start function. An unsolicited interruption condition is identified at the subchannel as alert status. An unsolicited interruption condition can be generated only when the subchannel is not device active.

The subchannel and device status associated with an unsolicited interruption condition is never merged with that of any currently existing interruption condition. If the subchannel is currently status pending, the unsolicited interruption condition is held in abeyance in either the channel subsystem or the device, as appropriate, until the status-pending condition has been cleared. Whenever the subchannel is idle and zero status is presented by the device, the status is discarded.

***Solicited Interruption Condition:*** A solicited interruption condition is any interruption condition generated as a direct consequence of performing or attempting to perform a clear, halt, resume, or start function. Solicited interruption conditions include any interruption condition generated while the subchannel is either subchannel-and-device active or device active. The subchannel and device status associated with a solicited interruption condition may be merged at the subchannel with that of another currently existing solicited interruption condition. Figure 16-1 describes the interruption condition that results from any combination of bits in the status-control field of the SCSW.

| Status-Control Field | Status-Control-Bit Combinations | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Alert | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Primary | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Secondary | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| Intermediate | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| Status pending | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Resulting interruption condition | E | S | S | S | S | S | - | S | S | S | S | S | S | - | S | S |

**Explanation:**

- `-` Combination does not occur.
- `E` Unsolicited or solicited interruption condition.
- `S` Solicited interruption condition.
- `0` Indicates the bit stored as zero.
- `1` Indicates the bit stored as one.

*Figure 16-1. Interruption Condition for Status-Control-Bit Combinations*


LzLabs GmbH. Ex. 1008-711

## Intermediate Interruption Condition

An intermediate interruption condition is a solicited interruption condition that indicates that an event has occurred for which the program had previously requested notification. An intermediate interruption condition is described by solicited subchannel status, the Z bit, the subchannel-suspended condition, or any combination of the three. An intermediate interruption condition can occur only after it has been requested by the program through the use of flags in the ORB or a CCW. Depending on the state of the subchannel, the performance or suspension of the I/O operation continues, unaffected by the setting of the intermediate-status bit.

An intermediate interruption condition can be indicated only together with one of the following indications:

1. Subchannel active
2. Status pending with primary status alone
3. Status pending with primary status together with alert status or secondary status or both
4. Suspended

If only the intermediate-status bit and the status-pending bit of the status-control field are ones during the execution of TEST SUBCHANNEL, the device-status field is zero.

## Primary Interruption Condition

A primary interruption condition is a solicited interruption condition that indicates the performance of the start function is completed at the subchannel. A primary interruption condition is described by the SCSW stored as a result of the execution of TEST SUBCHANNEL while the subchannel is status pending with primary status. Once the primary interruption condition is indicated at the subchannel, the channel subsystem is no longer actively participating in the I/O operation by transferring commands or data. When a subchannel is status pending with a primary interruption condition, the execution of any of the following instructions results in the setting of a nonzero condition code: HALT SUBCHANNEL, MODIFY SUBCHANNEL, RESUME SUBCHANNEL, and START SUBCHANNEL. Once the primary interruption condition is cleared by the execution of TEST SUBCHANNEL, the subchannel accepts the START SUBCHANNEL instruction. (See “START SUBCHANNEL” on page 14-14.)

## Secondary Interruption Condition

A secondary interruption condition is a solicited interruption condition that normally indicates the completion of an I/O operation at the device. A secondary interruption condition is also generated by the channel subsystem if the start function is terminated because a solicited alert interruption condition is recognized prior to initiating the first I/O operation at the device. A secondary interruption condition is described by the SCSW stored as a result of the execution of TEST SUBCHANNEL while the subchannel is status pending with secondary status. Once the channel subsystem has accepted status from the device that causes a secondary interruption condition to be recognized, the start function is completed at the device.

## Alert Interruption Condition

An alert interruption condition is either a solicited interruption condition that indicates the occurrence of an unusual condition in a halt, resume, or start function or an unsolicited interruption condition that describes a condition unrelated to the performance of a halt, resume, or start function. An alert interruption condition is described by the SCSW stored as a result of the execution of TEST SUBCHANNEL while the subchannel is status pending with alert status. An alert interruption condition may be generated by either the channel subsystem or the device. Nonzero alert status is always brought to the attention of the program.

---

## Priority of Interruptions

All requests for an I/O interruption are asynchronous to any activity in any CPU, and interruption requests associated with more than one subchannel can exist at the same time. The priority of interruptions is controlled by two types of mechanisms — one establishes within the channel subsystem the priority among interruption requests from subchannels associated with the same I/O-interruption subclass, and another establishes within a given CPU the priority among requests from subchannels of different I/O-interruption sub-



LzLabs GmbH. Ex. 1008-712

classes. The channel subsystem requests an I/O interruption only after it has established priority among requests from its subchannels. The conditions responsible for the I/O-interruption requests associated with subchannels are preserved at the subchannels until cleared by a CPU's execution of TEST SUBCHANNEL or CLEAR SUBCHANNEL or I/O-system reset is performed.

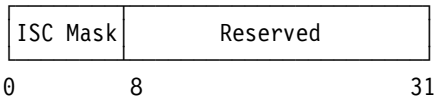
The assignment of priority among requests for interruption from subchannels of the same I/O-interruption subclass is in the order that the need for interruption is recognized by the channel subsystem. The order of recognition by the channel subsystem is a function of the type of interruption condition and the type of channel path. For the parallel-I/O-interface type of channel path, the order depends on the electrical position of the device on the channel path to which it is attached. (A device's electrical position on the parallel-I/O interface is not related to its device address.)

The assignment of priority among requests for interruption from subchannels of different I/O-interruption subclasses is made by the CPU according to the numerical value of the I/O-interruption subclass codes (with zero having highest priority), in conjunction with the I/O-interruption-subclass mask in control register 6. The numerical value of the I/O-interruption-subclass code directly corresponds to the bit position in the I/O-interruption-subclass mask in control register 6 of a CPU. If, in any CPU, an I/O-interruption-subclass-mask bit is zero, then all subchannels having an I/O-interruption-subclass code numerically equal to the associated position in the mask register are said to be masked off in the respective CPU. Therefore, a CPU accepts the highest-priority I/O-interruption request from a subchannel that has the lowest-numbered I/O-interruption subclass code that is not masked off by a corresponding bit in control register 6 of that CPU. When the highest-priority interruption request is accepted by a CPU, it is cleared so that the interruption request is not accepted by any other CPU in the configuration.

The priority of interruption handling can be modified by the execution of either TEST SUBCHANNEL or CLEAR SUBCHANNEL. When either of these instructions is executed and the designated subchannel has an interruption request pending, that interruption request is cleared, without regard to any previous established priority. The relative priority of the remaining interruption requests is unchanged.

**Programming Notes:**

1. The I/O-interruption subclass mask is in control register 6, which has the following format:




2. Control register 6 is set to all zeros during initial CPU reset.

## Interruption Action

An I/O interruption can occur only when the I/O-interruption-subclass-mask bit associated with the subchannel is one and the CPU is enabled for I/O interruptions.

The interruption occurs at the completion of a unit of operation (see “Point of Interruption” on page 5-17). If the channel subsystem establishes the priority among requests for interruption from subchannels while the CPU is disabled for I/O interruptions, the interruption occurs immediately after the completion of the instruction enabling the CPU and before the next instruction is executed, provided that the I/O-interruption subclass-mask bit associated with the subchannel is one. Alternatively, if the channel subsystem establishes the priority among requests for interruption from subchannels while the I/O-interruption-subclass-mask bit is zero for each subchannel that is status pending, the interruption occurs immediately after the completion of the instruction that sets at least one of the I/O-interruption-subclass-mask bits to one, provided that the CPU is also enabled for I/O interruptions. This interruption is associated with the highest-priority I/O-interruption request, as established by the CPU.

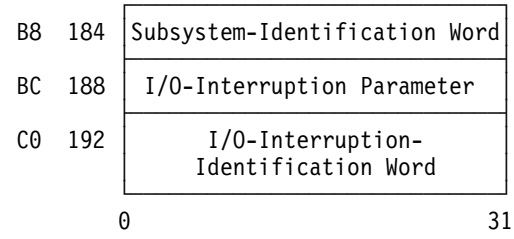
If the channel subsystem has not established the priority among requests for interruption from the subchannels by the time the interruption is allowed, the interruption does not necessarily occur immediately after the completion of the instruction enabling the CPU. A delay can occur


LzLabs GmbH. Ex. 1008-713

regardless of how long the interruption condition has existed at the subchannel.

The interruption causes the current PSW to be stored as the input/output old PSW at real locations 56-63 and causes the I/O-interruption code associated with the interruption to be stored at real locations 184-195 of the CPU allowing the interruption. Subsequently, a new input/output PSW is loaded from real locations 120-127, and processing resumes in the CPU state indicated by that PSW. The subchannel causing the interruption is identified by the interruption code.

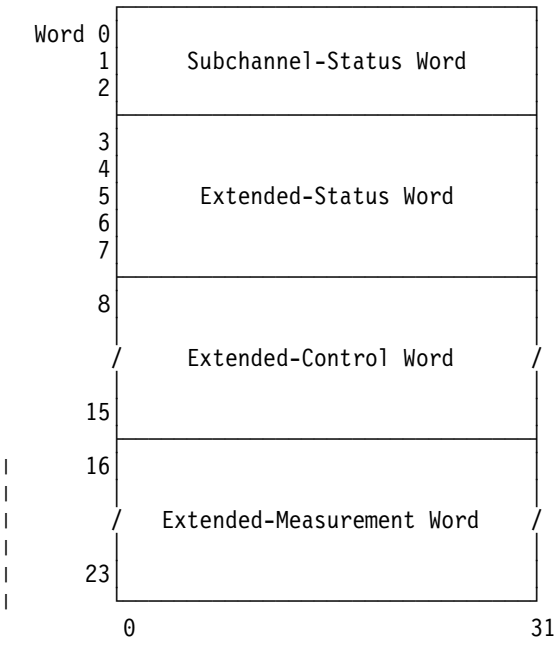
The I/O-interruption code has the following format when it is stored. The code is described in "TEST PENDING INTERRUPTION" on page 14-18.




**Programming Note:** The I/O-interruption subclass code for all subchannels is set to zero by I/O-system reset. It may be set to any of the values 0-7 by the execution of MODIFY SUBCHANNEL. (The operation of the instruction is described in "MODIFY SUBCHANNEL" on page 14-7.)

## Interruption-Response Block

The interruption-response block (IRB) is the operand of TEST SUBCHANNEL. The two rightmost bits of the IRB address are zeros, designating the IRB on a word boundary. The IRB contains three major fields: the subchannel-status word, the extended-status word, and the extended-control word. When the extended-I/O-measurement-word mode is enabled at the subchannel, the IRB contains a fourth major field, the extended-measurement word. The format of the IRB is as follows:




The length of the subchannel-status and extended-status words is 12 bytes and 20 bytes, respectively. The length of the extended-control word is 32 bytes. When the extended-control bit, bit 14 of word 0 of the SCSW, is zero, words 8-15 of the interruption-response block may or may not be stored. The length of the extended-measurement word is 32 bytes. When the conditions for storing the extended-measurement word are not met (see "Extended-Measurement Word" on page 16-40), words 16-23 of the interruption-response block may or may not be stored.

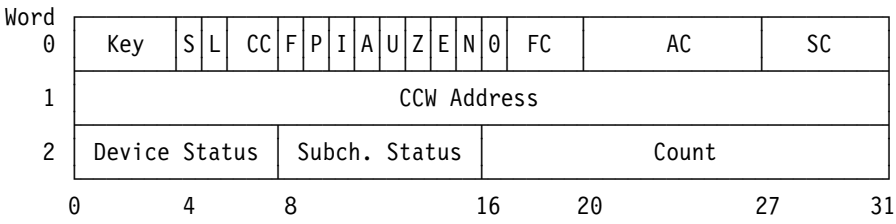
## Subchannel-Status Word

The subchannel-status word (SCSW) provides to the program indications describing the status of a subchannel and its associated device. If performance of a halt, resume, or start function has occurred, the SCSW may describe the conditions under which the operation was concluded.

The SCSW is stored when TEST SUBCHANNEL is executed and the designated subchannel is operational. The SCSW is placed in words 0-2 of the IRB that is designated as the TEST SUBCHANNEL operand. When STORE SUBCHANNEL is executed, the SCSW is stored in words 7-9 of the subchannel-information block (described in "Subchannel-Information Block" on page 15-1). Figure 16-2 on page 16-7 shows the format of the SCSW and summarizes its contents.



LzLabs GmbH. Ex. 1008-714




| Bits | Name |
|---|---|
| **Word 0** | |
| 0-3 | Subchannel key |
| 4 | Suspend control (S) |
| 5 | ESW format (L) |
| 6-7 | Deferred condition code (CC) |
| 8 | Format (F) |
| 9 | Prefetch (P) |
| 10 | Initial-status interruption control (I) |
| 11 | Address-limit-checking control (A) |
| 12 | Suppress-suspended interruption (U) |
| 13 | Zero condition code (Z) |
| 14 | Extended control (E) |
| 15 | Path not operational (N) |
| 16 | Reserved |
| 17-19 | Function control (FC)<br>(bit 17, start function; bit 18, halt function;<br>bit 19, clear function) |
| 20-26 | Activity control (AC)<br>(bit 20, resume pending; bit 21, start pending;<br>bit 22, halt pending; bit 23, clear pending;<br>bit 24, subchannel active; bit 25, device active;<br>bit 26, suspended) |
| 27-31 | Status control (SC)<br>(bit 27, alert status; bit 28, intermediate status;<br>bit 29, primary status; bit 30, secondary status;<br>bit 31, status pending) |
| **Word 1** | |
| 0-31 | CCW address |
| **Word 2** | |
| 0-7 | Device status<br>(bit 0, attention; bit 1, status modifier;<br>bit 2, control unit end; bit 3, busy;<br>bit 4, channel end; bit 5, device end;<br>bit 6, unit check; bit 7, unit exception) |
| 8-15 | Subchannel status<br>(bit 8, program-controlled interruption; bit 9, incorrect length;<br>bit 10, program check; bit 11, protection check;<br>bit 12, channel-data check; bit 13, channel-control check;<br>bit 14, interface-control check; bit 15, chaining check) |
| 16-31 | Count |

*Figure 16-2. SCSW Format*

The contents of the subchannel-status word (SCSW) depend on the state of the subchannel when the SCSW is stored. Depending on the state of the subchannel and the device, the specific fields of the SCSW may contain (1) information pertaining to the last operation, (2) information unrelated to the performance of an operation, (3) zeros, or (4) meaningless values. The following descriptions indicate when an SCSW field contains meaningful information.


LzLabs GmbH. Ex. 1008-715

### Subchannel Key

When the start-function bit, bit 17 of word 0, is one, bit positions 0-3 of word 0 contain the access key used during performance of the associated start function. These bits are identical with the key specified in bit positions 0-3 of word 1 of the ORB. The subchannel key is meaningful only when the start-function bit, bit 17 of word 0, is one.

### Suspend Control (S)

When the start-function bit, bit 17 of word 0, is one, bit 4 of word 0, when one, indicates that the suspend function can be initiated at the subchannel. Bit 4 is meaningful only when bit 17 is one. If bit 17 is one and bit 4 is one, channel-program execution can be suspended if the channel subsystem recognizes an S flag set to one in a CCW. If bit 4 is zero, channel-program execution cannot be suspended, and, if an S flag set to one in a CCW is encountered, a program-check condition is recognized.

### Extended-Status-Word Format (L)

When the status-pending bit, bit 31 of word 0, is one, bit 5 of word 0, when one, indicates that a format-0 ESW has been stored. A format-0 ESW is stored when an interruption condition containing any of the following indications is cleared by TEST SUBCHANNEL:

- Channel-data check
- Channel-control check
- Interface-control check
- Measurement-block-program check
- Measurement-block-data check
- Measurement-block-protection check
- Path verification required
- Authorization check
- Program check for QDIO subchannels
- Protection check for QDIO subchannels

The extended-status-word-format bit is meaningful whenever the subchannel is status pending. The extended-status information that is used to form a format-0 ESW is cleared at the subchannel by TEST SUBCHANNEL or CLEAR SUBCHANNEL.

### Deferred Condition Code (CC)

When the start-function bit, bit 17 of word 0, is one and the status-pending bit, bit 31 of word 0, is also one, bits 6 and 7 of word 0 indicate the general reason that the subchannel was status pending when TEST SUBCHANNEL or STORE SUBCHANNEL was executed. The deferred condition code is meaningful when the subchannel is status pending with any combination of status and only when the start-function bit of the function-control field in the SCSW is one. The meaning of the deferred condition code for each value when the subchannel is status pending is given in Figure 16-3 on page 16-10.

The deferred condition code, if not zero, is used to indicate whether conditions have been encountered that preclude the subchannel becoming subchannel-and-device active while the subchannel is either start pending or suspended.

***Deferred Condition Code 0:*** A normal I/O interruption has taken place.

***Deferred Condition Code 1:*** Status is present in the SCSW that was presented by the associated device or generated by the channel subsystem subsequent to the setting of condition code 0 for START SUBCHANNEL or RESUME SUBCHANNEL. If only the alert-status bit and the status-pending bit of the status-control field of the SCSW are ones, the status present is not related to the execution of a channel program. If the intermediate-status bit, the primary-status bit, or both are ones, then the status is related to the execution of the channel program specified by the most recently executed START SUBCHANNEL instruction or implied by the most recently executed RESUME SUBCHANNEL instruction. (See "Immediate Conclusion of I/O Operations" on page 15-43.) If the secondary-status bit is one and the primary-status bit is zero, the status present is related to the channel program specified by the START SUBCHANNEL instruction or implied by the RESUME SUBCHANNEL instruction that preceded the most recently executed START SUBCHANNEL instruction.

***Deferred Condition Code 2:*** This code does not occur and is reserved for future use.

***Deferred Condition Code 3:*** An attempted device selection has occurred, and the device appeared not operational on all of the channel



LzLabs GmbH. Ex. 1008-716

paths that were available for selection of the device.

A device appears not operational when it does not respond to a selection attempt by the channel subsystem. This occurs when the control unit is not provided in the system, when power is off in the control unit, or when the control unit has been logically switched off the channel path. The not-operational state is also indicated when the control unit is provided and is capable of attaching the device, but the device has not been installed and the control unit is not designed to recognize the device being selected as one of its attached devices. (See also "I/O Addressing" on page 13-5.)

A deferred condition code 3 also can be set by the channel subsystem if no channel paths to the device are available for selection. (See Figure 16-3 on page 16-10.)

**Programming Notes:**

1. If, during performance of a start function, the I/O device being selected is not installed or has been logically removed from the control unit, but the associated control unit is operational and the control unit recognizes the I/O device being selected as one of its I/O devices, the control unit, depending upon the model, either fails to recognize the address of the I/O device or considers the I/O device to be not ready. In the former case, a path-not-operational condition is recognized, subject to the setting of the path-operational mask. (See "Path-Operational Mask (POM)" on page 15-6.) In the latter case, the not-ready condition is indicated when the control unit responds to the selection and indicates unit check whenever the not-ready state precludes successful initiation of the operation at the I/O device. In this case, unit-check status is indicated in the SCSW, the subchannel becomes status pending with primary, secondary, and alert status, and with deferred condition code 1 indicated. (See the publication *ESA/390 Common I/O-Device Commands*, SA22-7204, for a description of unit-check status.) Refer to the System Library publication for the control unit to determine how the condition is indicated.
2. The deferred condition code is 1, and the status-control field contains the status-pending and intermediate-status bits or the status-pending, intermediate-status, and alert-status bits as ones when HALT SUBCHANNEL has been executed and the designated subchannel is suspended and status pending with intermediate status. If the alert-status bit is one, then subchannel-logout information was generated as a result of attempting to issue the halt signal to the device.


LzLabs GmbH. Ex. 1008-717

| Bit 6 | Bit 7 | Status Control[1] | Meaning |
|---|---|---|---|
| 0 | 0 | A I P S X<br>A I P - X<br>A - P S X<br>A - P - X<br>- I P S X<br>- I P - X<br>- I - - X<br>- - P S X<br>- - P - X | Normal I/O interruption |
| 0 | 1 | A I P S X<br>A I P - X<br>A I - - X[2]<br>A - P S X<br>A - P - X<br>A - - S X<br>A - - - X<br>- I P S X<br>- I P - X<br>- I - - X[2]<br>- - P S X<br>- - P - X<br>- - - S X[3]<br>- - - - X[3] [2] | Either an immediate operation, with chaining not specified, has ended normally, or the setting of some status condition precluded the initiation or resumption of a requested I/O operation at the device. |
| 1 | 0 | Reserved | Reserved |
| 1 | 1 | - - P S X<br>- I P S X | The device is not operational on any available path or, if a dedicated-allegiance condition exists, the device is not operational on the path to which the dedicated allegiance is owed. |

**Explanation:**

- \- Bit is zero.
- [1] The allowed combinations of status-control-bit settings when the start-function bit is one in the function-control field.
- [2] The condition is encountered after the execution of HALT SUBCHANNEL when the subchannel is currently suspended.
- [3] The condition is encountered after the execution of HALT SUBCHANNEL when the subchannel is currently start pending.
- A Alert status.
- I Intermediate status.
- P Primary status.
- S Secondary status.
- X Status pending.

*Figure 16-3. Deferred-Condition-Code Meaning for Status-Pending Subchannel*

## Format (F)

When the start-function bit, bit 17 of word 0, is one, bit 8 of word 0 indicates the format of the CCWs associated with an I/O operation. The format bit is meaningful only when bit 17 is one. If bit 8 of word 0 is zero, format-0 CCWs are indicated. If it is one, format-1 CCWs are indicated. (See "Channel-Command Word" on page 15-27 for the description of the two CCW formats.)

## Prefetch (P)

When the start-function bit, bit 17 of word 0, is one, bit 9 of word 0 indicates whether or not unlimited prefetching of CCWs, IDAWs, and associated data is allowed. The prefetch bit is meaningful only when bit 17 is one. If bit 9 is zero, prefetching of one CCW describing a data area is allowed during output-data-chaining operations



LzLabs GmbH. Ex. 1008-718

and is not allowed during any other operations. If bit 9 is one, unlimited prefetching of CCWs, IDAWs, and associated data is allowed. It is model dependent whether prefetching is actually performed for any or all of the CCWs, IDAWs, and associated data that comprise the channel program.

### Initial-Status-Interruption Control (I)

When the start-function bit, bit 17 of word 0, is one, bit 10 of word 0, when one, indicates that the channel subsystem is to generate an intermediate interruption condition if the subchannel becomes subchannel active (see "Initial-Status-Interruption Control (I)" on page 15-24). Bit 10 of word 0, when zero, indicates that the subchannel becoming subchannel active is not to cause an intermediate interruption condition to be generated.

The program requests the intermediate interruption condition by means of the ORB. An I/O interruption that results from that request may be due to the channel subsystem performing either a start function or a resume function. (See "Zero Condition Code (Z)" for details of the indication given by the channel subsystem when the intermediate interruption condition is cleared by TEST SUBCHANNEL.)

### Address-Limit-Checking Control (A)

When the start-function bit, bit 17 of word 0, is one, bit 11 of word 0, when one, indicates that the channel subsystem has been requested by the program to perform address-limit checking, subject to the setting of the limit mode at the subchannel (see "Address-Limit-Checking Control (A)" on page 15-25). The address-limit-checking-control bit is meaningful only when bit 17 is one.

### Suppress-Suspended Interruption (U)

When the start-function bit, bit 17 of word 0, is one, bit 12 of word 0, when one, indicates that the channel subsystem has been requested by the program to suppress the generation of a subchannel-suspended interruption condition when the subchannel is suspended (see "Suppress-Suspended-Interruption Control (U)" on page 15-25). When bit 12 is zero, the channel subsystem generates an intermediate interruption condition whenever the subchannel is suspended during the execution of the associated channel program. The suppress-suspended-interruption bit is meaningful only when bit 17 is one.

## Subchannel-Control Field

The following subchannel-control-information descriptions apply to the subchannel-control field, bits 13-31 of word 0 of the SCSW.

### Zero Condition Code (Z)

Bit 13 of word 0, when one, indicates that the subchannel has become subchannel active and the channel subsystem has recognized an initial-status-interruption condition at the subchannel. The Z bit is meaningful only when the intermediate-status bit, bit 28 of word 0, and the start-function bit, bit 17 of word 0, are both ones.

If the initial-status-interruption-control bit, bit 10 of word 1 of the ORB, is one when START SUBCHANNEL is executed, then the subchannel becoming subchannel active causes the subchannel to be made status pending with intermediate status indicating the initial-status-interruption condition. The initial-status-interruption condition remains at the subchannel until the intermediate interruption condition is cleared by the execution of TEST SUBCHANNEL or CLEAR SUBCHANNEL. If the initial-status-interruption-control bit of the ORB is zero when START SUBCHANNEL is executed, then the subchannel becoming subchannel active does not cause an intermediate interruption condition to be generated, and the initial-status-interruption condition is not recognized.

### Extended Control (E)

Bit 14 of word 0, when one, indicates that model-dependent information or concurrent-sense information is stored in the extended-control word (ECW). When bit 14 is zero, the contents of words 0-7 of the ECW, if stored, are unpredictable. The E bit is meaningful whenever the subchannel is status pending with alert status either alone or together with primary status, secondary status, or both.

**Programming Note:** During the execution of TEST SUBCHANNEL, the storing of words 0-7 of the ECW is a model-dependent function subject to the setting of bit 14 as described above. Therefore, the program should always provide sufficient storage to accommodate the storing of a 64-byte IRB.


LzLabs GmbH. Ex. 1008-719

## Path Not Operational (N)

Bit 15 of word 0, when one, indicates that the N condition has been recognized by the channel subsystem. The N condition, in turn, indicates that one or more path-not-operational conditions have been recognized. The channel subsystem recognizes a path-not-operational condition when, during an attempted device selection in order to perform a clear, halt, resume, or start function, the device associated with the subchannel appears not operational on a channel path that is operational for the subchannel. A channel path is operational for the subchannel if the associated device appeared operational on that channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function. A channel path is not operational for the subchannel if the associated device appeared not operational on that channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function. A device appears to be operational on a channel path when the device responds to an attempted device selection.

The N bit is meaningful whenever the status-control field contains any of the indications listed below and at least one basic I/O function is also indicated at the subchannel:

- Status pending with any combination of primary, secondary, or alert status
- Status pending alone
- Status pending with intermediate status when the subchannel is also suspended

The N condition is reset whenever the execution of TEST SUBCHANNEL results in the setting of condition code 0 and the N bit is meaningful as described above.

**Notes:**

1. A path-not-operational condition does not imply a malfunctioning channel path. A malfunctioning channel path causes the generation of an error indication, such as interface-control check.
2. When a path-not-operational condition has been recognized and the subchannel subsequently becomes status pending with only intermediate status, the path-not-operational condition (a) continues to be recognized until the subchannel becomes status pending with primary status or becomes suspended and (b) is indicated by storing the path-not-operational bit as a one during the execution of TEST SUBCHANNEL. When a path-not-operational condition has been recognized and the channel-program execution subsequently becomes suspended, the path-not-operational condition does not remain pending if channel-program execution is subsequently resumed. Instead, the old indication is lost, and the path-not-operational indication, if any, pertains to the attempt by the channel subsystem to resume channel-program execution.

## Function Control (FC)

The function-control field indicates the basic I/O functions that are indicated at the subchannel. This field may indicate the acceptance of as many as two functions. The function-control field is contained in bit positions 17-19 of the first word of the SCSW. The function-control field is meaningful at an installed subchannel whenever the subchannel is valid (see “Device Number Valid (V)” on page 15-4). The function-control field contains all zeros whenever both the activity- and status-control fields contain all zeros. The meaning of the individual bits is as follows:

***Start Function (Bit 17):*** When one, bit 17 indicates that a start function has been requested and is either pending or in progress at the subchannel. A start function is requested by the execution of START SUBCHANNEL. A start function is indicated at the subchannel when condition code 0 is set during the execution of START SUBCHANNEL. The start-function indication is cleared at the subchannel when TEST SUBCHANNEL is executed and the subchannel is either status pending alone or status pending with any combination of alert, primary, or secondary status. The start-function indication is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

***Halt Function (Bit 18):*** When one, bit 18 indicates that a halt function has been requested and is either pending or in progress at the subchannel. A halt function is requested by the execution of HALT SUBCHANNEL. A halt function is indicated at the subchannel when condition code 0 is set for HALT SUBCHANNEL. The halt-function indication is cleared at the subchannel when the next status-pending condition that occurs is cleared by the execution of TEST SUBCHANNEL. The next



LzLabs GmbH. Ex. 1008-720

status-pending condition depends on the state of the subchannel when HALT SUBCHANNEL is executed. If the subchannel is subchannel active when HALT SUBCHANNEL is executed, then the next status-pending condition is status pending with at least primary status indicated. If the subchannel is device active when HALT SUBCHANNEL is executed, then the next status-pending condition is status pending with at least secondary status indicated. If the subchannel is suspended and status pending with intermediate status when HALT SUBCHANNEL is executed, then the next status-pending condition is status pending with intermediate status. If the subchannel is idle when HALT SUBCHANNEL is executed, then the next status-pending condition is status pending alone. The halt-function indication is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL. In normal operations, this function is indicated together with bit 17; that is, there is a start function either pending or in progress that is to be halted.

***Clear Function (Bit 19):*** When one, bit 19 indicates that a clear function has been requested and is either pending or in progress at the subchannel. A clear function is requested by the execution of CLEAR SUBCHANNEL. A clear function is indicated at the subchannel when condition code 0 is set for CLEAR SUBCHANNEL (see "CLEAR SUBCHANNEL" on page 14-5). The clear-function indication is cleared at the subchannel when the resulting status-pending condition is cleared by TEST SUBCHANNEL.

## Activity Control (AC)

The activity-control field is contained in bit positions 20-26 of the first word of the SCSW. This field indicates the current progress of a basic I/O function previously accepted at the subchannel. By using the contents of this field, the program can determine the degree of completion of the basic I/O function. The activity-control field is meaningful at an installed subchannel whenever the subchannel is valid (see "Device Number Valid (V)" on page 15-4). However, if an IFCC or CCC condition is detected during the performance of a basic I/O function and that function is indicated as pending, I/O operations may or may not have been performed at the device. The activity-control bits are defined as follows:

| Bit | Meaning |
|---|---|
| 20 | Resume pending |
| 21 | Start pending |
| 22 | Halt pending |
| 23 | Clear pending |
| 24 | Subchannel active |
| 25 | Device active |
| 26 | Suspended |

When an SCSW is stored that has the status-pending bit of the status-control field zero and all zeros in the activity-control field, the subchannel is said to be idle or in the idle state.

**Note:** All conditions that are represented by the bits in the function-control field and by the resume-pending, start-pending, halt-pending, clear-pending, subchannel-active, and suspended bits in the activity-control field are reset at the subchannel when TEST SUBCHANNEL is executed and the subchannel is (1) status pending alone, (2) status pending with primary status, (3) status pending with alert status, or (4) status pending with intermediate status and is also suspended.

***Resume-Pending (Bit 20):*** When one, bit 20 indicates that the subchannel is resume pending. The channel subsystem may or may not be in the process of performing the start function. The subchannel becomes resume pending when condition code 0 is set for RESUME SUBCHANNEL. The point at which the subchannel is no longer resume pending is a function of the subchannel state existing when the resume-pending condition is recognized and the state of the device if channel-program execution is resumed.

If the subchannel is in the suspended state when the resume-pending condition is recognized, the CCW that caused the suspension is refetched, the setting of the suspend flag is examined, and one of the following actions is taken by the channel subsystem:

1. If the CCW suspend flag is one, the device is not selected, the subchannel is no longer resume pending, and the channel-program execution remains suspended.
2. If the CCW suspend flag is zero, the channel subsystem attempts to resume channel-program execution by performing a modified start function. The resumption of channel-program execution appears to the device as the initiation of a new channel-program execution. The resume function causes the channel subsystem to perform the path-management operation as if a new start func-


LzLabs GmbH. Ex. 1008-721

tion were being initiated, using the ORB parameters previously passed to the subchannel by START SUBCHANNEL with the exception that the channel-program address is the address of the CCW that caused the suspension of the channel-program execution.

The subchannel remains resume pending when, during the performance of the start function, the channel subsystem (1) determines that it is not possible to attempt to initiate the I/O operation for the first command, (2) determines that an attempt to initiate the I/O operation for the first command does not result in the command being accepted, or (3) detects an IFCC or CCC condition and is unable to determine whether the first command has been accepted. (See "Start Function and Resume Function" on page 15-18.)

The subchannel is no longer resume pending when any of the following events occurs:

a. While performing the start function, the subchannel becomes subchannel-and-device active or device active only, or the first command is accepted with channel-end and device-end initial status and the CCW does not specify command chaining.

b. CLEAR SUBCHANNEL is executed.

c. TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.

d. TEST SUBCHANNEL clears intermediate status while the subchannel is suspended.

e. CANCEL SUBCHANNEL is executed with a resulting condition code 0.

If the subchannel is not in the suspended state when the resume-pending condition is recognized, the CCW suspend flag of the most recently fetched CCW, if any, is examined, and one of the following actions is taken by the channel subsystem:

1. If a CCW has not been fetched or the suspend flag of the most recently fetched CCW is zero, the subchannel is no longer resume pending, and the resume function is not performed.

2. If the suspend flag of the most recently fetched CCW is one, the subchannel is no longer resume pending, and the CCW is refetched. The subchannel proceeds with channel-program execution if the suspend flag of the refetched CCW is zero. The subchannel suspends channel-program execution if the suspend flag of the refetched CCW is one.

Some models recognize a resume-pending condition only after a CCW having an S flag validly set to one is fetched. Therefore, if a subchannel is resume pending and, during the execution of the channel program, no CCW is fetched that has an S flag validly set to one, the subchannel remains resume pending until the primary interruption condition is cleared by TEST SUBCHANNEL.

***Start-Pending (Bit 21):*** When one, bit 21 indicates that the subchannel is start pending. The channel subsystem may or may not be in the process of performing the start function. The subchannel becomes start pending when condition code 0 is set for START SUBCHANNEL. The subchannel remains start pending when, during the performance of the start function, the channel subsystem (1) determines that it is not possible to attempt to initiate the I/O operation for the first command, (2) determines that an attempt to initiate the I/O operation for the first command does not result in the command being accepted, or (3) detects an IFCC or CCC condition and is unable to determine whether the first command has been accepted. (See "Start Function and Resume Function" on page 15-18.)

The subchannel becomes no longer start pending when any of the following occurs:

1. While performing the start function, the subchannel becomes subchannel-and-device active or device active only, or the first command is accepted with channel-end and device-end initial status and the CCW does not specify command chaining.

2. The subchannel becomes suspended because of a suspend flag validly set to one in the first CCW.

3. CLEAR SUBCHANNEL is executed.

4. TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.



LzLabs GmbH. Ex. 1008-722

5. CANCEL SUBCHANNEL is executed with a resulting condition code 0.

***Halt-Pending (Bit 22):*** When one, bit 22 indicates that the subchannel is halt pending. The channel subsystem may or may not be in the process of performing the halt function. The subchannel becomes halt pending when condition code 0 is set for HALT SUBCHANNEL. The subchannel remains halt pending when, during the performance of the halt function, the channel subsystem (1) determines that it is not possible to attempt to issue the halt signal to the device, (2) determines that the attempt to issue the halt signal to the device is not successful, or (3) detects an IFCC or CCC condition and is unable to determine whether the halt signal is issued to the device. (See "Halt Function" on page 15-15.)

The subchannel is no longer halt pending when any of the following occurs:

1. While performing the halt function, the channel subsystem determines that the halt signal has been issued to the device.
2. CLEAR SUBCHANNEL is executed.
3. TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.
4. TEST SUBCHANNEL clears intermediate status while the subchannel is suspended.

***Clear-Pending (Bit 23):*** When one, bit 23 indicates that the subchannel is clear pending. The channel subsystem may or may not be in the process of performing the clear function. The subchannel becomes clear pending when condition code 0 is set for CLEAR SUBCHANNEL. The subchannel remains clear pending when, during performance of the clear function, the channel subsystem (1) determines that it is not possible to attempt to issue the clear signal to the device, (2) determines that the attempt to issue the clear signal to the device is not successful, or (3) detects an IFCC or CCC condition and is unable to determine whether the clear signal is issued to the device. (See "Clear Function" on page 15-14.)

The subchannel is no longer clear pending when either of the following occurs:

1. While performing the clear function, the channel subsystem determines that the clear signal has been issued to the device.
2. TEST SUBCHANNEL clears the status-pending condition alone.

***Subchannel Active (Bit 24):*** When one, bit 24 indicates that the subchannel is subchannel active. A subchannel is said to be subchannel active when an I/O operation is currently being performed at the subchannel. The subchannel becomes subchannel active when the first command is accepted and the start function or resume function is not immediately concluded at the subchannel. (See "Immediate Conclusion of I/O Operations" on page 15-43.) The subchannel is no longer subchannel active when any of the following occurs:

1. The subchannel becomes suspended.
2. The subchannel becomes status pending with primary status.
3. CLEAR SUBCHANNEL is executed.
4. The device appears not operational during performance of a halt function.

The subchannel does not become subchannel active during performance of the function specified by either a HALT SUBCHANNEL or a CLEAR SUBCHANNEL instruction.

***Device Active (Bit 25):*** When one, bit 25 indicates that the subchannel is device active. A subchannel is said to be device active when an I/O operation is currently in progress at the associated device. The subchannel becomes device active when the first command is accepted. The subchannel is no longer device active when any of the following occurs:

1. The subchannel becomes suspended.
2. The subchannel becomes status pending with secondary status.
3. CLEAR SUBCHANNEL is executed.
4. The device appears not operational during performance of a halt function.

If the subchannel is not start pending or if the status accepted from the device also describes an alert condition, the subchannel becomes status pending with secondary status. After the status has been accepted from the device, the device is capable of accepting a command for performing a


LzLabs GmbH. Ex. 1008-723

new I/O operation. If the subchannel is start pending and the status is device end or device end with control unit end, then the channel subsystem discards the status and performs the start function for the new channel program. (See "Start Function and Resume Function" on page 15-18) In this situation, the subchannel does not become status pending with the secondary interruption condition, and the status is not made available to the program.

The subchannel does not become device active during performance of the functions specified by either a HALT SUBCHANNEL or a CLEAR SUBCHANNEL instruction.

***Suspended (Bit 26):*** When one, bit 26 indicates that the subchannel is suspended. A subchannel is said to be suspended when channel-program execution is currently suspended. The subchannel becomes suspended as part of the suspend function. (See "Suspension of Channel-Program Execution" on page 15-38.)

The subchannel is no longer suspended when any of the following occurs:

1. As part of the resume function following the execution of RESUME SUBCHANNEL when the subchannel becomes subchannel-and-device active or device active only, or the first command is accepted for channel-end and device-end initial status, with or without status modifier, and the CCW does not specify command chaining.
2. CLEAR SUBCHANNEL is executed.
3. TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.
4. TEST SUBCHANNEL clears intermediate status while the halt function is specified.
5. CANCEL SUBCHANNEL is executed with a resulting condition code 0.

**Programming Note:** When an SCSW is stored by STORE SUBCHANNEL or TEST SUBCHANNEL following CLEAR SUBCHANNEL but prior to the subchannel becoming status pending, and the subchannel-active bit, bit 24 of word 0, is stored as zero, this does not mean that data transfer has stopped for the device. The program cannot determine whether data transfer has stopped until the subchannel becomes status pending as a result of performing the clear function.

## Status Control (SC)

The status-control field is contained in bit positions 27-31 of the first word of the SCSW. This field provides the program with a summary-level indication of the interruption condition described by either subchannel or device status, the Z bit, or, in the case of the subchannel-suspended interruption, the suspended bit, bit 26. More than one summary indication may be signaled as a result of existing conditions at the subchannel. Whenever the subchannel is enabled (see "Enabled (E)" on page 15-2) and at least bit 31 is one, the subchannel is said to be status pending. Whenever the subchannel is disabled, the subchannel is not made status pending. Bit 31 of SCSW word 0 is meaningful at an installed subchannel whenever the subchannel is valid (see "Device Number Valid (V)" on page 15-4); bits 27-30 are meaningful when bit 31 is one. The status-control bits are defined as follows:

***Alert Status (Bit 27):*** When one (and when the status-pending bit is also one), bit 27 indicates an alert interruption condition exists. In such a case, the subchannel is said to be status pending with alert status. An alert interruption condition is recognized when alert status is present at the subchannel. Alert status may be subchannel status or device status. Alert status is status generated by either the channel subsystem or the device under any of the following conditions:

- The subchannel is idle (activity-control bits 20-26 and status-control bit 31 are zeros).
- The subchannel is start pending, and the status condition precludes initiation of the I/O operation.
- The subchannel is subchannel-and-device active, and the status condition has suppressed command chaining or would have suppressed command chaining if chaining had been specified (see "Chaining" on page 15-31).
- The subchannel is subchannel-and-device active, command chaining is not specified, the execution of the channel program has just been concluded, and the status presented by the device is attempting to alter the sequential execution of commands (see the publication *ESA/390 Common I/O-Device Commands*,



LzLabs GmbH. Ex. 1008-724

SA22-7204, for more information on the use of status modifier to alter the sequential execution of commands).

- The subchannel is device active only, and the status presented by the device is other than device end, control unit end, or device end and control unit end.
- The subchannel is suspended (bit 26 is one).

If the subchannel is start pending when an alert interruption condition is recognized, the subchannel becomes status pending with alert status, deferred condition code 1 is set, the start-pending bit remains one, and the performance of the pending I/O operation is not initiated.

When TEST SUBCHANNEL is executed and stores an SCSW with the alert-status bit and the status-pending bit as ones in the IRB, the alert interruption condition is cleared at the subchannel. The alert interruption condition is also cleared during the execution of CLEAR SUBCHANNEL.

Whenever alert status is present at the subchannel, it is brought to the attention of the program. Examples of alert status include attention, device end (which signals a transition from the not-ready to the ready state), incorrect length, program check, and unit check.

***Intermediate Status (Bit 28):*** When one (and when the status-pending bit is also one), bit 28 indicates an intermediate interruption condition exists. In such a case, the subchannel is said to be status pending with intermediate status. Intermediate status can be indicated when the Z bit (of the subchannel-control field), the suspended bit (of the activity-control field), or the PCI bit (of the subchannel-status field) is one.

When the initial-status-interruption-control bit is one in the ORB, the subchannel becomes status pending with intermediate status (the Z bit indicated) only after the subchannel is subchannel active. If the subchannel does not become subchannel active, the Z condition is not generated.

When suspend control is specified and the generation of an intermediate interruption condition due to suspension is not suppressed in the ORB, then the subchannel can become status pending with intermediate status due to suspension if a CCW becomes current that contains the suspend flag set to one. When the suspend flag is specified in the first CCW of a channel program, channel-program execution is suspended, and the subchannel becomes status pending with intermediate status (the suspended bit indicated) before the command in the first CCW is transferred to the device. When the suspend flag is specified in a CCW fetched during command chaining, then channel-program execution is suspended, and the subchannel becomes status pending with intermediate status (the suspended bit is indicated), only after the execution of the preceding CCW is complete.

When the PCI flag is specified in a CCW, the generation of an intermediate interruption condition due to PCI depends on whether the CCW is the first CCW of the channel program. When the PCI flag is specified in the first CCW of a channel program, the subchannel becomes status pending with intermediate status (the PCI bit indicated) only after initial status is received for the first CCW of the channel program indicating the command has been accepted. When the PCI flag is specified in a CCW fetched while chaining, the subchannel becomes status pending with intermediate status (the PCI bit indicated) only after the execution of the preceding CCW is complete. If chaining occurs before an interruption condition containing PCI is cleared by TEST SUBCHANNEL, the condition is carried over to the next CCW. This carry-over occurs during both data and command chaining, and, in either case, the condition is propagated through the transfer-in-channel command.

If the subchannel is status pending with intermediate status when HALT SUBCHANNEL is executed, the intermediate interruption condition remains at the subchannel, but the interruption request, if any, is withdrawn, and the subchannel becomes no longer status-pending. The subchannel remains no longer status pending until performance of the halt function has ended. The subchannel then becomes status pending with intermediate status indicated (possibly together with any combination of primary, secondary, and alert status).

When TEST SUBCHANNEL is executed and stores an SCSW with the intermediate-status bit and the status-pending bit as ones in the IRB, the intermediate interruption condition is cleared at the subchannel. The intermediate interruption condi-


LzLabs GmbH. Ex. 1008-725

tion is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

***Primary Status (Bit 29):*** When one (and when the status-pending bit is also one), bit 29 indicates a primary interruption condition exists. In such a case, the subchannel is said to be status pending with primary status. A primary interruption condition is a solicited interruption condition that indicates the completion of the start function at the subchannel. The primary interruption condition is described by the SCSW stored. When an I/O operation is terminated by HALT SUBCHANNEL but the halt signal is not issued to the device because the device appeared not operational, the subchannel is made status pending with primary status (and secondary status) with both the subchannel-status field and the device-status field set to zero.

When TEST SUBCHANNEL is executed and stores an SCSW with the primary-status bit and the status-pending bit as ones in the IRB, the primary interruption condition is cleared at the subchannel. The primary interruption condition is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

***Secondary Status (Bit 30):*** When one (and when the status-pending bit is also one), bit 30 indicates a secondary interruption condition exists. In such a case, the subchannel is said to be status pending with secondary status. A secondary interruption condition is a solicited interruption condition that normally indicates the completion of the I/O operation at the device. The secondary interruption condition is described by the SCSW stored.

When an I/O operation is terminated by HALT SUBCHANNEL but the halt signal is not issued to the device because the device appeared not operational, the subchannel is made status pending with secondary status (and primary status if the subchannel is also subchannel active) with zeros for subchannel and device status.

When TEST SUBCHANNEL is executed and stores an SCSW with the secondary-status bit as one in the IRB, the secondary interruption condition is cleared at the subchannel. The secondary interruption condition is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

***Status-Pending (Bit 31):*** When one, bit 31 indicates that the subchannel is status pending and that information describing the cause of the interruption condition is available to the program. The subchannel becomes status pending whenever intermediate, primary, secondary, or alert status is generated. When HALT SUBCHANNEL is executed, designating a subchannel that is idle, the subchannel becomes status pending subsequent to performance of the halt function to notify the program that the halt function has been completed. When TEST SUBCHANNEL is executed, thus storing an SCSW with the status-pending bit as one in the IRB, the status-pending condition is cleared at the subchannel. The status-pending condition is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL. When CLEAR SUBCHANNEL is executed and the designated subchannel is operational, the subchannel becomes status pending subsequent to performance of the clear function to notify the program that the clear function has been completed.

**Note:** The status-pending bit, in conjunction with the remaining bits of the status-control field, indicates the type of status condition. For example, if bits 29 and 31 are ones, the subchannel is status pending with primary status. Alternatively, if only bit 31 is one, then the subchannel is said to be status pending or status pending alone. If only bit 31 is one in the status-control field, the settings of all bits in the subchannel-status and device-status fields are unpredictable. If bit 31 is not one, then the remaining bits of the status-control field are not meaningful.

## CCW-Address Field

Bits 1-31 of word 1 form an absolute address. The address indicated is a function of the subchannel state when the SCSW is stored, as indicated in Figure 16-4 on page 16-19. When the subchannel-status field indicates channel-control check, channel-data check, or interface-control check, the CCW-address field is usable for recovery purposes if the CCW-address field-validity flag in the ESW is one.

**Programming Note:** When a CCW address, either detected in the channel-program address (see “Channel-Program Address” on page 15-26) or generated during chaining, would cause the channel subsystem to fetch a CCW from a



LzLabs GmbH. Ex. 1008-726

location greater than $2^{24} - 1$ while format-0 CCWs are specified for the operation, the invalid address is stored in the CCW-address field of the SCSW without truncation. If the invalid address causes the channel subsystem, while chaining, to fetch a CCW from a location greater than $2^{31} - 1$ while format-1 CCWs are specified for the operation, the rightmost 31 bits of the invalid address are stored in the CCW-address field.

| Subchannel State[1] | CCW Address[2] |
|---|---|
| Start pending (UUUU0/AIPSX)[3] | Unpredictable |
| Start pending and device active (UUUU0/AIPSX)[3] | Unpredictable |
| Subchannel-and-device active (UUUU0/AIPSX)[3] | Unpredictable |
| Device active only (UUUU0/AIPSX) | Unpredictable |
| Suspended (YYYYY/AIPSX)[3] | See note 1 |
| Status pending (10001/AIPSX) because of unsolicited alert status from the device while the subchannel was start pending[3] | Channel-program address + 8 |
| Status pending (0Y111/AIPSX) because the device appeared not operational on all paths[3] | Channel-program address + 8 |
| Status pending (10011/AIPSX) because of solicited alert status from the device while the subchannel was start pending and device active[3] | Channel-program address + 8 |
| Status pending (10111/AIPSX) because of solicited alert status generated by the channel subsystem while the subchannel was start pending[3] or start pending and device active[3] | See note 2 |
| Status pending (01001/AIPSX) for the program-controlled-interruption condition while the subchannel was subchannel-and-device active[3] | CCW + 8 of the CCW that contained the last recognized PCI, or 8 higher than a CCW that has subsequently become current |
| Status pending (01001/AIPSX) for the initial-status-interruption condition while the subchannel was subchannel-and-device active[3] | CCW + 8 of the CCW causing the intermediate interruption condition, or a CCW that has subsequently become current |
| Status pending (1Y1Y1/AIPSX); termination occurred because of program check caused by one of the following conditions:[3] | |
| Bit 24 of word 1 of the ORB set to one; incorrect-length-indication-suppression facility not installed | Channel-program address + 8 |
| Unused bits in ORB not set to zeros | Channel-program address + 8 |
| Invalid CCW-address specification in transfer in channel (TIC) | Address of TIC + 8 |
| Invalid CCW-address specification in the channel-program address in the ORB | Channel-program address + 8[4] |

*Figure 16-4 (Part 1 of 4). CCW Address as Function of Subchannel State*


LzLabs GmbH. Ex. 1008-727

| Subchannel State[1] | CCW Address[2] |
|---|---|
| Invalid CCW address in TIC | Address of TIC + 8 |
| Invalid CCW address in the channel-program address in the ORB | Channel-program address + 8[4] |
| Invalid CCW address while chaining | Invalid CCW address + 8 |
| Invalid command code | Address of invalid CCW + 8[5] |
| Invalid count | Address of invalid CCW + 8[5] |
| Invalid IDAW-address specification | Address of invalid CCW + 8[5] |
| Invalid IDAW address in a CCW | Address of invalid CCW + 8[5] |
| Invalid IDAW address while sequentially fetching IDAWs | Address of current CCW + 8 |
| Invalid data-address specification, format 1 | Address of invalid CCW + 8[5] |
| Invalid data address in a CCW | Address of invalid CCW + 8[5] |
| Invalid data address while sequentially accessing storage | Address of current CCW + 8 |
| Invalid data address in IDAW | Address of current CCW + 8 |
| Invalid IDAW specification | Address of current CCW + 8 |
| Invalid CCW, format 0 or 1, for a CCW other than a TIC | Address of invalid CCW + 8[5] |
| Invalid suspend flag — CCW fetched during data chaining has suspend flag set to one | Address of invalid CCW + 8 |
| Invalid suspend flag — CCW has suspend flag set to one, but suspend control was not specified in the ORB | Address of invalid CCW + 8 |
| Invalid CCW, format 1, for a TIC | Address of TIC + 8 |
| Invalid sequence — two TICs | Address of second TIC + 8 |
| Invalid sequence — 256 or more CCWs without data transfer | Address of 256th CCW + 8 |
| Status pending (1Y1Y1/AIPSX); termination occurred because of protection check detected as follows:[3] | |
| On a CCW access | Address of the protected CCW + 8[5] |
| On data or an IDAW access | Address of current CCW + 8 |

*Figure 16-4 (Part 2 of 4). CCW Address as Function of Subchannel State*



LzLabs GmbH. Ex. 1008-728

| Subchannel State[1] | CCW Address[2] |
|---|---|
| Status pending (1Y1Y1/AIPSX); termination occurred because of chaining check[3] | Address of current CCW + 8 |
| Status pending (YY1Y1/AIPSX); termination occurred under count control[3] | Address of current CCW + 8[6] |
| Status pending (1Y1Y1/AIPSX); operation prematurely terminated by the device because of alert status[3] | Address of current CCW + 8[6] |
| Status pending (YYYY1/AIPSX) after termination by HALT SUBCHANNEL and the activity-control-field bits indicated below set to ones: | |
| Status pending alone | Unpredictable |
| Start pending[3] | Unpredictable |
| Device active and start pending[3] | Unpredictable |
| Device active | Unpredictable |
| Subchannel active and device active[3] | CCW + 8 of the last-executed CCW |
| Suspended | CCW + 8 of CCW causing suspension |
| Suspended and resume pending | Unpredictable |
| Status pending (00001/AIPSX) after termination by CLEAR SUBCHANNEL | Unpredictable |
| Status pending (YY1Y1/AIPSX); operation completed normally at the subchannel[3] | CCW + 8 of the last-executed CCW[6] |
| Status pending (00011/AIPSX) | Unpredictable |
| Status pending (10001/AIPSX) | Unpredictable |
| Status pending (00001/AIPSX) | Unpredictable |
| Status pending (1Y111/AIPSX); command chaining suppressed because of alert status other than channel-control check or interface-control check[3] | Address of current CCW + 8[6] |
| Status pending (1YYY1/AIPSX) because of alert status for channel-control check or interface-control check[3] | See note 3[6] |
| Status pending (1Y1Y1/AIPSX) because of channel-data check[3] | Address of current CCW + 8[6] |

*Figure 16-4 (Part 3 of 4). CCW Address as Function of Subchannel State*


LzLabs GmbH. Ex. 1008-729

**Explanation:**

[1] The meaning of the notation used in this column is as follows:
- A Alert status
- I Intermediate status
- P Primary status
- S Secondary status
- X Status pending

The possible combination of status-control-bit settings is shown to the left of the "/" symbol by the use of these symbols:

- 0 Corresponding condition is not indicated.
- 1 Corresponding condition is indicated.
- U Unpredictable. The corresponding condition is not meaningful when the subchannel is not status pending.
- Y The corresponding condition is not significant and is indicated as a function of the subchannel state.

[2] A CCW becomes current when (1) it is the first CCW of a channel program and has been fetched, (2) while command chaining, the previous CCW is no longer current and the new CCW has been fetched, or (3) in the case of data chaining, the new CCW takes over control of the I/O operation (see the section "Data Chaining" in Chapter 15, "Basic I/O Functions"). If chaining is not specified or is suppressed, a CCW is no longer current and becomes the last-executed CCW when secondary status has been accepted by the channel subsystem. During command chaining, a CCW is no longer current when device-end status has been accepted or, in the case of data chaining, when the last byte of data for that CCW has been accepted.

[3] The subchannel may also be resume pending.

[4] The stored address is the channel-program address (in the ORB) + 8 even though it is either invalid or protected.

[5] The stored address is the address of the current CCW + 8 even though it is either invalid or protected.

[6] Incorrect length is indicated as a function of the setting of the suppress-length-indication flag in the current CCW (see the section "Channel-Command Word" in Chapter 15, "Basic I/O Functions").

Notes:

1. Unless the subchannel is also resume pending, the address stored is the address of the CCW that caused suspension, plus 8. Otherwise, the address stored is unpredictable.
2. The address of the CCW is given as a function of the alert status indicated. For example, if a program-check or protection-check condition is recognized, the CCW address stored is the same as for the entry for program check or protection check, respectively, in this table. Alternatively, if alert status for interface-control check or channel-control check is indicated, the CCW address stored is either the channel-program address (in the ORB) + 8 or invalid as specified by the field-validity flags in the subchannel logout.
3. Bit 21 of the subchannel-logout information, when stored as one, indicates that the address is CCW + 8 of the last-fetched CCW if the command for the CCW has not been accepted by the device. If the command has been accepted by the device at the time the error condition is recognized, then the address stored is the address of the CCW + 8 of the last-executed CCW.

*Figure 16-4 (Part 4 of 4). CCW Address as Function of Subchannel State*



LzLabs GmbH. Ex. 1008-730

## Device-Status Field

Device-status conditions are generated by the I/O device and are presented to the channel subsystem over the channel path. The timing and causes of these conditions for each type of device are specified in the System Library publication for the device. The device-status field is meaningful whenever the subchannel is status pending with any combination of primary, secondary, intermediate, or alert status. Whenever the subchannel is status pending with intermediate status alone, the device-status field is zero. When the subchannel-status field indicates channel-control check, channel-data check, or interface-control check, the device-status field is usable for recovery purposes if the device-status field-validity flag in the ESW is one. When the subchannel is status pending with deferred condition code 3 indicated, the contents of the device-status field are not meaningful.

If, within a system, the I/O device is accessible from more than one channel path, status related to channel-subsystem-initiated operations in the single-path mode (solicited status) is signaled over the initiating channel path. Devices operating in the multipath mode may signal solicited status over any channel path that belongs to the same path group as the initiating channel path. The handling of conditions not associated with I/O operations (unsolicited alert status), such as attention, unit exception, and device end due to transition from the not-ready to the ready state, depends on the type of device and condition and is specified in the System Library publication for the device.

The channel subsystem does not modify the status bits received from the I/O device. These bits appear in the SCSW as received over the channel path. For more information on the status bits received from the I/O device, see the publication *ESA/390 Common I/O-Device Commands*, SA22-7204.

## Subchannel-Status Field

Subchannel-status conditions are detected and indicated in the SCSW by the channel subsystem. Except for the conditions caused by equipment malfunctioning, they can occur only while the channel subsystem is involved with the performance of a halt, resume, or start function. The subchannel-status field is meaningful whenever the subchannel is status pending with any combination of primary, secondary, intermediate, or alert status. Individual bits contained in the subchannel-status field may be unpredictable even when the subchannel-status field is meaningful. When the subchannel is status pending with deferred condition code 3 indicated, the contents of the subchannel-status field are not meaningful.

### Program-Controlled Interruption

An intermediate interruption condition is generated after a CCW with the program-controlled-interruption (PCI) flag set to one becomes the current CCW. The I/O interruption due to the PCI flag may be delayed an unpredictable amount of time because of masking of the interruption request or other activity in the system. (See "Program-Controlled Interruption" on page 15-35.) When the channel subsystem recognizes an alert interruption condition due to either a channel-control-check condition or an interface-control-check condition, then any previously existing intermediate interruption condition caused by a PCI flag in a CCW may or may not be recognized by the channel subsystem.

Detection of the PCI condition does not affect the progress of the I/O operation.

### Incorrect Length

Incorrect length occurs when the number of bytes contained in the storage areas assigned for the I/O operation is not equal to the number of bytes requested or offered by the I/O device. Incorrect length is indicated for one of the following reasons:

***Long Block on Input:*** During a read, read-backward, or sense operation, the device attempted to transfer one or more bytes to main storage after the assigned main-storage areas were filled, or the device indicated that more data could have been transferred if the count had been larger. The extra bytes have not been placed in main storage. The count in the SCSW is zero.

***Long Block on Output:*** During a write or control operation, the device requested one or more bytes from the channel subsystem after the assigned main-storage areas were exhausted, or the device indicated that more data could have been transferred if the count had been larger. The count in the SCSW is zero.


LzLabs GmbH. Ex. 1008-731

***Short Block on Input:*** The number of bytes transferred during a read, read-backward, or sense operation is insufficient to fill the main-storage areas assigned to the operation. The count in the SCSW is not zero.

***Short Block on Output:*** The device terminated a write or control operation before all information contained in the assigned main-storage areas was transferred to the device. The count in the SCSW is not zero.

The incorrect-length indication is suppressed when the current CCW has the SLI flag set to one and the CD flag set to zero. The indication does not occur for operations rejected during the initiation sequence. The indication also does not occur for immediate operations when the count field is nonzero and the subchannel is in the incorrect-length-suppression mode. The incorrect-length indication is not meaningful when the count field of the SCSW is not meaningful.

Presence of the incorrect-length condition suppresses command chaining unless the SLI flag in the CCW is one or unless the condition occurs in an immediate operation when the subchannel is in the incorrect-length-suppression mode.

## Program Check

Program check occurs when programming errors are detected by the channel subsystem. The condition can be due to the following causes:

***Invalid CCW-Address Specification:*** The channel-program address (CPA) or the transfer-in-channel command does not designate the CCW on a doubleword boundary, or bit 0 of the CPA or bit 32 of a format-1 CCW specifying the transfer-in-channel command is not zero.

***Invalid CCW Address:*** The channel subsystem has attempted to fetch a CCW from a main-storage location that is not available. An invalid CCW address can occur because the program has designated an invalid address in the channel-program-address field of the ORB or in the transfer-in-channel command or because, on chaining, the channel subsystem attempts to fetch a CCW from an unavailable location. A main-storage location is unavailable when any of the following conditions is detected:

1. The absolute CCW address does not correspond to a physical location.
2. Format-0 CCWs are specified in the ORB, and the absolute CCW address is greater than $2^{24} - 1$.
3. Format-1 CCWs are specified in the ORB, and the absolute CCW address is greater than $2^{31} - 1$.

***Invalid Command Code:*** There are zeros in the four rightmost bit positions of the command code in the CCW designated by the CPA or in a CCW fetched on command chaining. The command code is not tested for validity during data chaining.

***Invalid Count, Format 0:*** A CCW, which is other than a CCW specifying transfer in channel, contains zeros in bit positions 48-63.

***Invalid Count, Format 1:*** A CCW that specifies data chaining or a CCW fetched while data chaining contains zeros in bit positions 16-31.

***Invalid IDAW-Address Specification:*** Indirect data addressing is specified, and either of the following conditions is detected:

1. The ORB specifies format-1 IDAWs, and the contents of the data-address field in the CCW do not designate the first IDAW on a word boundary; that is, bits 30 and 31 (format-0 CCW) or 62 and 63 (format-1 CCW) are not zeros.
2. The ORB specifies format-2 IDAWs, and the contents of the data-address field in the CCW do not designate the first IDAW on a doubleword boundary; that is, bits 29-31 (format-0 CCW) or 61-63 (format-1 CCW) are not zeros.

***Invalid IDAW Address:*** The channel subsystem has attempted to fetch an IDAW from a main-storage location that is not available. An invalid IDAW address can occur because the program has designated an invalid address in a CCW that specifies indirect data addressing or because the channel subsystem, on sequentially fetching IDAWs, attempts to fetch from an unavailable location. A main-storage location is unavailable when any of the following conditions is detected:

1. The absolute IDAW address does not correspond to a physical location.



LzLabs GmbH. Ex. 1008-732

2. Format-0 CCWs are specified in the ORB, and the absolute IDAW address is greater than $2^{24} - 1$.
3. Format-1 CCWs are specified in the ORB, and the absolute IDAW address is greater than $2^{31} - 1$.

***Invalid Data-Address Specification:*** Bit 32 of a format-1 CCW is not zero.

***Invalid Data Address:*** When any of the following conditions is detected, an invalid data address is recognized by the channel subsystem.

1. Use of the data address has caused the channel subsystem to attempt to wrap from the maximum storage address to zero.
2. Use of the data address has caused the channel subsystem to attempt to wrap from zero to the maximum storage address during a read-backward operation.
3. The channel subsystem has attempted to transfer data to a storage location that is unavailable.

An invalid data address can occur because the program has designated an unavailable location in a CCW or in an IDAW, or because the channel subsystem, on sequentially accessing storage, attempted to access an unavailable location. A main-storage location is unavailable when any of the following conditions is detected:

1. The absolute address of the location does not correspond to a physical location.
2. Format-0 CCWs are specified in the ORB, indirect data addressing is not specified in the CCW, and the absolute address is greater than $2^{24} - 1$.
3. Format-1 CCWs are specified in the ORB, indirect data addressing is not specified in the CCW, and the absolute address is greater than $2^{31} - 1$.
4. Format-1 IDAWs are specified in the ORB, indirect data addressing is specified in the CCW, and the absolute address is greater than $2^{31} - 1$.
5. The absolute address is outside the addressing range specified by SET ADDRESS LIMIT, and the limit mode at the subchannel is active.

**Note:** The maximum storage address is determined as a function of the CCW and IDAW formats used. When an IDAW is not used, the maximum storage address is a function of the CCW format specified, as follows:

1. When 24-bit (format 0) CCWs are specified, the maximum storage address recognized by the channel subsystem is $2^{24} - 1$.
2. When 31-bit (format 1) CCWs are specified, the maximum storage address recognized by the channel subsystem is $2^{31} - 1$.

When an IDAW is used, the maximum storage address is a function of the IDAW format specified, as follows:

1. When 31-bit (format 1) IDAWs are specified, the maximum storage address recognized by the channel subsystem is $2^{31} - 1$.
2. When 64-bit (format 2) IDAWs are specified, the maximum storage address recognized by the channel subsystem is $2^{64} - 1$

***Invalid IDAW Specification:*** When any of the following conditions is detected, an invalid IDAW specification is recognized by the channel subsystem:

1. Bit 0 of a format-1 IDAW is not zero.
2. A second or subsequent format-1 IDAW does not designate the location of the beginning byte of a 2K-byte block or, for read-backward operations, the location of the ending byte of a 2K-byte block.
3. A second or subsequent format-2 IDAW does not designate the location of the beginning byte of a 2K-byte or 4K-byte block, as required by the 2K-IDAW control in the ORB, or, for read-backward operations, the location of the ending byte of a 2K-byte or 4K-byte block.

***Invalid CCW, Format 0:*** A CCW other than a CCW specifying transfer in channel does not contain a zero in bit position 39.

***Invalid CCW, Format 1:*** A CCW other than a CCW specifying transfer in channel does not contain a zero in bit position 15, or a CCW specifying transfer in channel does not contain zeros in bit positions 0-3 and 8-31.


LzLabs GmbH. Ex. 1008-733

***Invalid Suspend Flag:*** A format-0 or format-1 CCW fetched during data chaining, other than a CCW specifying transfer in channel, does not contain a zero in bit position 38 or 14, respectively. A CCW other than a CCW specifying transfer in channel does not contain a zero in bit position 38 for a format-0 CCW or bit position 14 for a format-1 CCW, and suspend control was not specified by bit 4 of word 1 of the ORB.

***Invalid ORB Format:*** One or more reserved bit positions in the operation-request block (ORB) is not zero. (See "Operation-Request Block" on page 15-22 for more information.) If the incorrect-length-indication-suppression facility is not installed, then bit 24 of word 1 of the ORB must also be zero.

***Invalid Sequence:*** The channel subsystem has fetched two successive CCWs both of which specify transfer in channel, or, depending on the model, a sequence of 256 or more CCWs with command chaining specified was executed by the channel subsystem and did not result in the transfer of any data to or from an I/O device.

Detection of the program-check condition during the initiation of an operation at the device causes the operation to be suppressed and the subchannel to be made status pending with primary, secondary, and alert status. When the condition is detected after the I/O operation has been initiated at the device, the device is signaled to conclude the operation the next time the device requests or offers a byte of data or status. In this situation, the subchannel is made status pending as a function of the status received from the device. The program-check condition causes command chaining and command retry to be suppressed.

## Protection Check

Protection check occurs when the channel subsystem attempts a storage access that is prohibited by the protection mechanism. Protection applies to the fetching of CCWs, IDAWs, and output data and to the storing of input data. The subchannel key provided in the ORB is used as the access key for storage accesses associated with an I/O operation.

Detection of the protection-check condition during the fetching of the first CCW or IDAW causes the operation to be suppressed and the subchannel to be made status pending with primary, secondary, and alert status. When protection check is detected after the I/O operation has been initiated at the device, the device is signaled to conclude the operation after the available data logically prior to the protection check has been transferred. However, if an access violation occurs when the channel subsystem is in the process of fetching either a new IDAW or a new CCW while data chaining, and if the device signals the channel-end condition before transferring any data designated by the new CCW or IDAW, then the status is accepted, and the subchannel becomes status pending with primary and alert status and with protection check indicated. Other indications may accompany the protection-check indication as a function of the operation specified by the CCW, the status received from the device, and the current state of the subchannel. The protection-check condition causes command chaining and command retry to be suppressed.

## Channel-Data Check

Channel-data check indicates that an uncorrected storage error has been detected in regard to data, contained in main storage, that is currently used in the performance of an I/O operation. The condition may be indicated when detected, even if the data is not used when prefetched. Channel-data check is indicated when data or the associated key has an invalid checking-block code (CBC) in main storage when that data is referenced by the channel subsystem.

On an input operation, when the channel subsystem attempts to store less than a complete checking block, and invalid CBC is detected on the checking block in storage, the contents of the location remain unchanged and with invalid CBC. On an output operation, whenever channel-data check is indicated, no bytes from the checking block with invalid CBC are transferred to the device.

During a storage access, the maximum number of bytes that can be transferred is model dependent. If a channel-data-check condition is recognized during that storage access, the number of bytes transferred to or from storage may not be detectable by the channel subsystem. Consequently, the number of bytes transferred to or from storage may not be correctly reflected by the residual count. However, the residual count that is stored in the SCSW, when used in conjunction with the



LzLabs GmbH. Ex. 1008-734

storage-access code and the CCW address, designates a byte location within the page in which the channel-data-check condition was recognized.

A condition indicated as channel-data check causes the current operation, if any, to be terminated. The subchannel becomes status pending with primary and alert status, or with primary, secondary, and alert status, as a function of the status received from the device. The count and address fields of the SCSW stored by TEST SUBCHANNEL pertain to the operation terminated. The extended-status-word-format bit is one, and subchannel-logout information is stored in the ESW, when TEST SUBCHANNEL is executed.

Whenever the channel-data-check condition pertains to prefetched data, the failing-storage-address-validity flag, bit 6 of the ERW, is one. An address of a location within the checking block for which the channel-data-check condition is generated is stored in the ESW failing-storage-address field.

Uncorrectable storage or key errors detected on prefetched data while the subchannel is start pending cause the operation to be canceled before initiation at the device. In this case, the subchannel is made status pending with primary, secondary, and alert status, with channel-data check indicated, and with the ESW failing-storage address stored.

Whenever channel-data check is indicated, no measurement data for the subchannel is stored.

## Channel-Control Check

Channel-control check is caused by any machine malfunction affecting channel-subsystem controls. The condition includes invalid CBC on a CCW, an IDAW, or the respective associated key. The condition may be indicated when an invalid CBC is detected on a prefetched CCW, IDAW, or the respective associated key, even if that CCW or IDAW is not used.

Channel-control check may also indicate that an error has been detected in the information transferred to or from main storage during an I/O operation. However, when this condition is detected, the error has occurred inboard of the channel path: in the channel subsystem or in the path between the channel subsystem and main storage.

Detection of the channel-control-check condition causes the current operation, if any, to be terminated immediately. The subchannel is made status pending with primary and alert status or with primary, secondary, and alert status as a function of the type of termination, the current subchannel state, and the device status presented, if any. When the channel subsystem recognizes a channel-control-check condition, any previously existing intermediate interruption condition caused by a PCI flag in a CCW may or may not be recognized by the channel subsystem. The count and data-address fields of the SCSW stored by TEST SUBCHANNEL pertain to the operation terminated. The extended-status-word-format bit is one, and subchannel-logout information is stored in the ESW, when TEST SUBCHANNEL is executed.

Whenever the channel-control-check condition pertains to an invalid CBC detected on a prefetched CCW, a prefetched IDAW, or the key associated with the prefetched CCW or the prefetched IDAW, an extended-report word with bit 6 set to one, and the failing-storage address, are stored in the ESW when TEST SUBCHANNEL is executed.

Channel-control-check conditions encountered while prefetching when the subchannel is start pending cause the operation to be canceled before initiation at the device. In this case, the subchannel is made status pending with primary, secondary, and alert status, with channel-control check indicated, and with a failing-storage address that will be stored in the ESW.

If a subchannel is halt pending and the channel subsystem encounters a channel-control-check condition while performing the halt function for that subchannel, the subchannel remains halt pending unless the channel subsystem can determine that the halt signal was issued. The subchannel remains halt pending even if the channel subsystem was attempting to issue the halt signal and is unable to determine if the halt signal was issued.

If a subchannel is start pending or resume pending and the channel subsystem encounters a channel-control-check condition while performing the start function for that subchannel, the subchannel remains start pending or resume pending


LzLabs GmbH. Ex. 1008-735

unless the channel subsystem can determine that the first command was accepted. The subchannel remains start pending or resume pending even if the channel subsystem was attempting to initiate the I/O operation for the first command and is unable to determine if the command was accepted. If the channel subsystem is unable to determine whether the first command was accepted, the subchannel is made status pending with at least alert and primary status.

In some situations in which a channel-subsystem malfunction exists, the channel-control-check condition may be reported as a machine-check condition.

Whenever channel-control check is indicated, no measurement data for the subchannel is stored.

**Programming Note:** If the status-control field of the SCSW indicates that the subchannel is status pending with alert status but the field-validity flags of the SCSW indicate that the device-status field is not usable for error-recovery purposes, the program should (1) assume that the channel-control-check condition occurred while the channel subsystem was accepting alert status from the device and (2) take the appropriate action for alert status, even though the status itself has been lost.

## Interface-Control Check

Interface-control check indicates that an invalid signal has occurred on the channel path. The condition is detected by the channel subsystem and usually indicates malfunctioning of an I/O device. Interface-control check can occur for any of the following reasons:

1. A data or status byte received from a device while the subchannel is subchannel-and-device active or device active has an invalid checking-block code.
2. The status byte received from a device while the subchannel is idle, start pending, suspended, or halt pending has an invalid checking-block code.
3. A device responded with an address other than the address designated by the channel subsystem during initiation of an operation.
4. During command chaining, the device appeared not operational.
5. A signal from an I/O device either did not occur or occurred at an invalid time or had an invalid duration.
6. The channel subsystem recognized the I/O-error-alert condition (see "I/O-Error Alert (A)" on page 16-35).
7. ESW bit 26, indicating device-status check, is set to one.

Detection of the interface-control-check condition causes the current operation, if any, to be terminated immediately, and the subchannel is made status pending with alert status, primary and alert status, secondary and alert status, or primary, secondary, and alert status as a function of the type of termination, the current subchannel state, and the device status presented, if any. When the channel subsystem recognizes an interface-control-check condition, any previously existing intermediate interruption condition caused by a PCI flag in a CCW may or may not be recognized by the channel subsystem. The extended-status-word-format bit is one, and subchannel-logout information is stored in the ESW, when TEST SUBCHANNEL is executed.

If a subchannel is halt pending and the channel subsystem encounters an interface-control-check condition while performing the halt function for that subchannel, the subchannel remains halt pending unless the channel subsystem can determine that the halt signal was issued. The subchannel remains halt pending even if the channel subsystem was attempting to issue the halt signal and is unable to determine if the halt signal was issued.

If a subchannel is start pending or resume pending and the channel subsystem encounters an interface-control-check condition while performing the start function for that subchannel, the subchannel remains start pending or resume pending unless the channel subsystem can determine that the first command was accepted. The subchannel remains start pending or resume pending even if the channel subsystem was attempting to initiate the I/O operation for the first command and is unable to determine if the command was accepted. If the channel subsystem is unable to determine whether the first command was accepted, the subchannel is made status pending with at least alert and primary status.



LzLabs GmbH. Ex. 1008-736

If, while initiating a signaling sequence with the channel subsystem for the purpose of presenting status or transferring data, the device presents an address with invalid parity, the error condition is not made available to the program since the identity of the device and associated subchannel are unknown.

Whenever interface-control check is indicated, no measurement data for the subchannel is stored.

**Programming Note:** If the status-control field of the SCSW indicates that the subchannel is status pending with alert status but the field-validity flags of the SCSW indicate that the device-status field is not usable for error-recovery purposes, the program should (1) assume that the interface-control-check condition occurred while the channel subsystem was accepting alert status from the device and (2) take the appropriate action for alert status, even though the status itself has been lost.

### Chaining Check

Chaining check is caused by channel-subsystem overrun during data chaining on input operations. The condition occurs when the I/O-data rate is too high for the particular resolution of data addresses. Chaining check cannot occur on output operations.

Detection of the chaining-check condition causes the I/O device to be signaled to conclude the operation. It causes command chaining to be suppressed.

## Count Field

Bit positions 16-31 of word 2 contain the residual count. The count is to be used in conjunction with the original count specified in the last CCW and, depending upon existing conditions (see Figure 16-4 on page 16-19), indicates the number of bytes transferred to or from the area designated by the CCW. The count field is meaningful whenever the subchannel is status pending with primary status that consists of either (1) device status only or (2) device status together with subchannel status of incorrect length only, PCI only, or both.

In Figure 16-5 on page 16-30, the contents of the count field are listed for all cases where the subchannel is start pending, subchannel-and-device active, device active, suspended, or status pending.


LzLabs GmbH. Ex. 1008-737

| Subchannel State[1] | Count |
|---|---|
| Start pending (UUUU0/AIPSX)[2] | Not meaningful[3] |
| Start pending and status pending (10YY1/AIPSX)[2] | Not meaningful[3] |
| Start pending and status pending (00111/AIPSX) because the device appeared not operational on all paths[2] | Not meaningful[3] |
| Start pending and device active (UUUU0/AIPSX)[2] | Not meaningful[3] |
| Suspended (YYYYY/AIPSX)[2] | Not meaningful[3] |
| Subchannel-and-device active (UUUU0/AIPSX)[2] | Not meaningful[3] |
| Device active (UUUU0/AIPSX) | Not meaningful[3] |
| Status pending (01001/AIPSX) because of program-controlled-interruption condition or initial-status interruption | Not meaningful[3] |
| Status pending (1Y1Y1/AIPSX); termination occurred because of:[2] | |
| Program check | Not meaningful[3] |
| Protection check | Not meaningful[3] |
| Chaining check | Not meaningful[3] |
| Channel-control check | See note 1 |
| Interface-control check | Not meaningful[3] |
| Channel-data check | See note 2 |
| Status pending (YY1Y1/AIPSX); termination occurred under count control[2] | Correct |
| Status pending (Y0011/AIPSX)[2] | Not meaningful[3] |
| Status pending (1Y1Y1/AIPSX)[2] | Correct; residual count of last used CCW |
| Status pending (1Y111/AIPSX); command chaining suppressed because of alert status[2] | Correct; residual count of last used CCW |
| Status pending (YYYY1/AIPSX); after termination by HALT SUBCHANNEL[2] | Unpredictable |
| Status pending (00001/AIPSX); after termination by CLEAR SUBCHANNEL | Not meaningful[3] |
| Status pending (YY1Y1/AIPSX); operation completed normally at the subchannel[2] | Correct; indicates the residual count |

*Figure 16-5 (Part 1 of 2). Contents of Count Field in the SCSW*



LzLabs GmbH. Ex. 1008-738

| Subchannel State[1] | Count |
|---|---|
| Status pending (1Y111/AIPSX); command chaining terminated because of alert status[2] | Correct; original count of CCW specifying the new I/O operation |
| Status pending (10001/AIPSX) because of alert status | Not meaningful[3] |

**Explanation:**

[1] In situations where more than a single condition exists because of, for example, alert status that is described by program check and unit check, the entry appearing first in the table takes precedence.

The meaning of the notation in this column is as follows:

- A Alert status
- I Intermediate status
- P Primary status
- S Secondary status
- X Status pending

The allowed combination of status-control-bit settings is shown to the left of the "/" symbol.

Bit settings are specified as follows:

- 0 Corresponding condition is not indicated.
- 1 Corresponding condition is indicated.
- U Unpredictable. The corresponding condition is not meaningful when the subchannel is not status pending.
- Y Corresponding condition is not significant and is indicated as a function of the subchannel state.

[2] The subchannel may also be resume pending.

[3] The contents of the count field are not meaningful because the count field is not valid when the SCSW is stored and the subchannel is in the given state.

Notes:

1. The count is unpredictable unless IDAW check is indicated, in which case the count may not correctly reflect the number of bytes transferred to or from main storage but will (when used in conjunction with the CCW address) designate a byte location within the page in which the channel-control-check condition was recognized.

2. During a storage access, the maximum number of bytes that can be stored by a channel subsystem is model dependent. If a channel-data-check condition is recognized during that access, the number of bytes transferred to or from storage may not be detectable by the channel subsystem. Consequently, the number of bytes transferred to or from storage may not be correctly reflected by the residual count. However, the residual count that is stored when used in conjunction with the storage-access code and the CCW address designates a byte location within the page in which the channel-data-check condition was recognized.

*Figure 16-5 (Part 2 of 2). Contents of Count Field in the SCSW*


LzLabs GmbH. Ex. 1008-739

## Extended-Status Word

The extended-status word (ESW) provides additional information to the program about the subchannel and its associated device. The ESW is placed in words 3-7 of the IRB designated by the second operand of TEST SUBCHANNEL when TEST SUBCHANNEL is executed and the subchannel designated is operational. If the subchannel is status pending or status pending with any combination of primary, secondary, intermediate, or alert status (except as noted in the next paragraph) when TEST SUBCHANNEL is executed, the ESW may have any of the following types of extended-status format:

*Format 0* Subchannel logout in word 0, an ERW in word 1, a failing-storage address or zeros in words 2 and 3, and a secondary-CCW address or zeros in word 4.

*Format 1* Zeros in bytes 0, 2, and 3 of word 0, the LPUM in byte 1 of word 0, an ERW in word 1, and zeros in words 2-4.

*Format 2* Zeros in byte 0, the LPUM in byte 1, and the device-connect time in bytes 2 and 3 of word 0; an ERW in word 1; zeros in words 2-4.

*Format 3* Zeros in byte 0, the LPUM in byte 1, and unpredictable values in bytes 2 and 3 of word 0; an ERW in word 1; zeros in words 2-4.

Words 0-4 of the ESW contain unpredictable values if any of the following conditions is met:

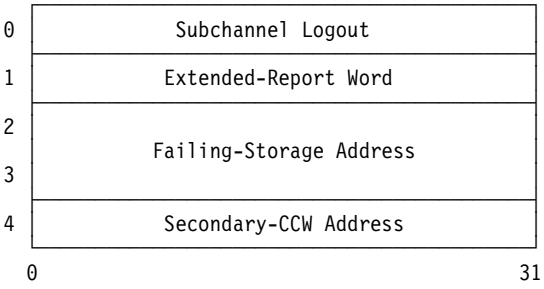
1. The subchannel is not status pending.
2. The subchannel is status pending alone, and the extended-status-word-format bit is zero.
3. The subchannel is status pending with intermediate status alone for other than the intermediate interruption condition due to suspension.

The type of extended-status format stored depends upon conditions existing at the subchannel at the time TEST SUBCHANNEL is executed. The conditions under which each of the types of formats is stored are described in the remainder of this section.

## Extended-Status Format 0

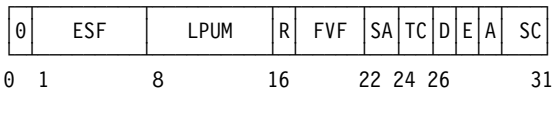
The ESW stored by TEST SUBCHANNEL is a format-0 ESW when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, is one and the subchannel is status pending with any combination of status as defined in Figure 16-6 on page 16-36. In this case, subchannel-logout information and an ERW are stored in the extended-status word. Subchannel logout provides detailed model-independent information relating to a subchannel and describing equipment errors detected by the channel subsystem. The information is provided to aid the recovery of an I/O operation, a device, or both. Whenever subchannel logout is provided, the error conditions relate only to the subchannel reporting the error. If I/O operations involving other subchannels have been affected by the error condition, those subchannels also provide similar subchannel-logout information. An extended-report word provides additional information relating to the cause of the malfunction.

A format-0 ESW has the following format:




### Subchannel Logout

The subchannel logout has the following format:




***Extended-Status Flags (ESF):*** Any of bits 1-7, when one, specifies that an error-check condition has been detected by the channel subsystem. The following indications are provided in the ESF field:

*Key Check*. Bit 1, when one, indicates that the channel subsystem, when accessing data, when attempting to update the measurement block, or when attempting to fetch either a



LzLabs GmbH. Ex. 1008-740

CCW or an IDAW, has detected an invalid checking-block code (CBC) on the associated storage key. The channel-data-check bit, bit 12 of word 2 of the SCSW, the measurement-block data-check bit, bit 3 of word 0 of the ESW, the CCW-check bit, bit 5 of word 0 of the ESW, or the IDAW-check bit, bit 6 of word 0 of the ESW, identifies the source of the key error.

**Note:** This condition may be indicated to the program when an invalid checking-block code on a key is detected but the data, CCW, or IDAW then is not used after being prefetched. In this case, the failing-storage-address-validity bit, bit 6 of the ERW, is one, indicating that the address of a location within the storage block having the key is stored in words 2 and 3 of the ESW.

*Measurement-Block Program Check*. Bit 2, when one, indicates that the channel subsystem, in attempting to update the measurement block, has either detected an invalid absolute address when combining the measurement-block origin with the measurement-block index for this subchannel or has detected an invalid measurement-block address at the subchannel.

*Measurement-Block Data Check*. Bit 3, when one, indicates that a malfunction has been detected involving the data of the measurement block in main storage. (See "Measurement Block" on page 17-3.) Measurement-block data check is indicated when the measurement block is updated and an invalid checking-block code (CBC) is detected on the storage used to contain the measurement data or on the associated key. When invalid CBC on the associated key is detected, the key-check bit, bit 1 of the ESF field, is also stored as one.

*Measurement-Block Protection Check*. Bit 4, when one, indicates that the channel subsystem, when attempting to update the measurement block, has been prohibited from accessing the measurement block because the storage key does not match the measurement-block key (see "Measurement Block" on page 17-3.) The key provided by SET CHANNEL MONITOR is used for the access of storage associated with measurement-block-update operations (see "SET CHANNEL MONITOR" on page 14-12).

**Note:** Whenever any of the measurement-check conditions is indicated by bits 2-4, the channel subsystem sets the subchannel measurement-block-update-enable bit to zero, disabling the storing of measurement data for the subchannel (see "Measurement-Mode Enable (MM)" on page 15-3).

*CCW Check*. Bit 5, when one, indicates that an invalid CBC on the contents of the CCW or its associated key has been detected. When either of these conditions is detected, the I/O operation is terminated, the subchannel becomes status pending with primary and alert status, the extended-status-word-format bit in the SCSW is stored as one, and channel-control check is indicated in the subchannel-status field. The subchannel also becomes status pending with secondary status as a function of the type of termination or status received from the device. When invalid CBC on the associated key is detected, the key-check bit, bit 1 of the ESF field, is also stored as one.

**Note:** This condition may be indicated to the program when an invalid checking-block code on a prefetched CCW is detected but the CCW is not used. In this case, the failing-storage-address-validity bit, bit 6 of the ERW, is one, indicating that the address of a location having the invalid CBC is stored in words 2 and 3 of the ESW.

*IDAW Check*. Bit 6, when one, indicates that an invalid CBC on the contents of an IDAW or its associated key has been detected. When either of these conditions is detected, the I/O operation is terminated with the device, the subchannel becomes status pending with primary and alert status, the extended-status-word-format bit in the SCSW is one, and channel-control check is indicated in the subchannel-status field. The subchannel also becomes status pending with secondary status as a function of the type of termination or status received from the device. When invalid CBC on the associated key is detected, the key-check bit, bit 1 of the ESF field, is also one.

**Note:** This condition may be indicated to the program when an invalid checking-block code on the contents of a prefetched IDAW is detected but the IDAW is not used. In this


LzLabs GmbH. Ex. 1008-741

case, the failing-storage-address-validity bit, bit 6 of the ERW, is one, indicating that the address of a location having the invalid CBC is stored in words 2 and 3 of the ESW. Detection of a channel-data-check condition does not cause the CCW-check and IDAW-check bits to be stored as ones.

*Reserved*. Bit 7 is stored as zero.

***Last-Path-Used Mask (LPUM):*** Bits 8-15 indicate the channel path that was last used for communicating or transferring information between the channel subsystem and the device. The bit corresponding to the channel path in use is set whenever any of the following occurs:

1. The first command of a start-subchannel function is accepted by the device (see "Activity Control (AC)" on page 16-13).
2. The device and channel subsystem are actively communicating when the channel subsystem performs the suspend function for the channel program in execution.
3. The channel subsystem accepts status from the device that is recognized as an interruption condition, or a condition has been recognized that suppresses command chaining (see "Interruption Conditions" on page 16-2).
4. The channel subsystem recognizes an interface-control-check condition (see "Interface-Control Check" on page 16-28), and no subchannel-logout information is currently present at the subchannel.

The LPUM field contains the most recent setting and is valid whenever the ESW contains information in one of the formats 0-3 (see "Extended-Status Word" on page 16-32) and the SCSW is stored. When subchannel-logout information is present in the ESW, a zero LPUM-field-validity flag indicates that the LPUM setting is not consistent with the other subchannel-logout indications.

***Ancillary Report (R):*** Bit 16, when one, indicates that a malfunction of a system component has occurred that has been recognized previously or that has affected the activities of multiple subchannels. When the malfunction affects the activities of multiple subchannels, an ancillary-report condition is recognized for all of the affected subchannels except one. This bit, when zero, indicates that this malfunction of a system component has not been recognized previously. This bit is meaningful only when a channel-control check, channel-data check, or an interface-control check is indicated in bit positions 12-14 of word 2 of the SCSW.

Depending on the model, recognition of an ancillary-report condition may not be provided or it may not be provided for all system malfunctions that effect subchannel activity. When ancillary-report recognition is not provided, bit 16 is set to zero.

***Field-Validity Flags (FVF):*** Bits 17-21 indicate the validity of the information stored in the corresponding fields of either the SCSW or the extended-status word. When the validity bit is one, the corresponding field has been stored and is usable for recovery purposes. When the validity bit is zero, the corresponding field is not usable.

This bit-significant field has meaning when channel-data check, channel-control check, or interface-control check is indicated in the SCSW. When these checks are not indicated, this field, as well as the termination-code and sequence-code fields, has no meaning. Furthermore, when these checks are not indicated, the last-path-used-mask, device-status, and CCW-address fields are all valid. The fields are defined as follows:

17 Last-path-used mask
18 Termination code
19 Sequence code
20 Device status
21 CCW address

***Storage-Access Code (SA):*** Bits 22-23 indicate the type of storage access that was being performed by the channel subsystem at the time of error. The SA field pertains only to the access of storage for the purpose of fetching or storing data during the performance of an I/O operation. This encoded field has meaning only when channel-data check, channel-control check, or interface-control check is indicated in the subchannel status. The access-code assignments are as follows:

00 Access type unknown
01 Read
10 Write
11 Read backward



LzLabs GmbH. Ex. 1008-742

***Termination Code (TC):*** Bits 24 and 25 indicate the type of termination that has occurred. This encoded field has meaning only when channel-data check, channel-control check, or interface-control check is indicated in the SCSW. The types of termination are as follows:

00 Halt signal issued
01 Stop, stack, or normal termination
10 Clear signal issued
11 Reserved

When at least one channel check is indicated in the SCSW but the termination-code-field-validity flag is zero, it is unpredictable which, if any, termination has been signaled to the device. If more than one channel-check condition is indicated in the SCSW, the device may have been signaled one or more termination codes that are the same or different. In this situation, if the termination-code-field-validity flag is one, the termination code indicates the most severe of the terminations signaled to the device. The termination codes, in order of increasing severity, are: stop, stack, or normal termination (01); halt signal issued (00); and clear signal issued (10).

***Device-Status Check (D):*** When the status-verification facility is installed, bit 26, when one, indicates that the subchannel logout in the ESW resulted from the channel subsystem detecting device status that had valid CBC but that contained a combination of bits that was inappropriate when the status byte was presented to the channel subsystem. When the device-status-check bit is one, the interface-control-check status bit is set to one. If, additionally, bit 20 of the subchannel-logout field has been stored as one, then the status byte in error has been stored in the device-status field of the SCSW. If the status-verification facility is not installed, bit 26 is stored as zero.

***Secondary Error (E):*** Bit 27, when one, indicates that a malfunction of a system component that may or may not have been directly related to any activity involving subchannels or I/O devices has occurred. Subsequent to this occurrence, the activity related to this subchannel and the associated I/O device was affected and caused the subchannel to be set status pending with either channel-control check or interface-control check.

***I/O-Error Alert (A):*** Bit 28, when one, indicates that subchannel logout in the ESW resulted from the signaling of I/O-error alert. The I/O-error-alert signal indicates that the control unit or device has detected a malfunction that must be reported to the channel subsystem. The channel subsystem, in response, issues a clear signal and, except as described in the next paragraph, causes interface-control check to be set and extended-status-format-0 (logout) information to be stored in the ESW.

When I/O-error alert is signaled and the subchannel has previously been set disabled or no subchannel is associated with the device, the clear signal is issued to the device, and the I/O-error-alert indication is ignored by the channel subsystem.

***Sequence Code (SC):*** Bits 29-31 identify the I/O sequence in progress at the time of error. The sequence code pertains only to I/O operations initiated by the execution of START SUBCHANNEL or RESUME SUBCHANNEL. This encoded field has meaning only when channel-data check, channel-control check, or interface-control check is indicated in the SCSW.

The sequence-code assignments are:

000 Reserved.

001 A nonzero command byte has been sent by the channel subsystem, but a response has not yet been analyzed by the channel subsystem. This code is set during the initiation sequence.

010 The command has been accepted by the device, but no data has been transferred.

011 At least one byte of data has been transferred between the channel subsystem and the device. This code may be used when the channel path is in an idle or polling state.

100 The command in the current CCW (1) has not yet been sent to the device, (2) was sent but not accepted by the device, or (3) was sent and accepted but command-retry status was presented. This code is set when any of the following conditions occurs:

1. The command address is updated during command chaining or during the initiation of a start function or resume function at the device.


LzLabs GmbH. Ex. 1008-743

2. During the initiation sequence, the status includes attention, control unit end, unit check, unit exception, busy, status modifier (without channel end and device end), or device end (without channel end).
3. Command retry is signaled.
4. The channel subsystem interrogates the device in the process of clearing an interruption condition.
5. The channel subsystem signals the conclusion of the chain of operations to the device during command chaining while performing the suspend function.

101 The command in the current CCW has been accepted, but data transfer is unpredictable. This code applies from the time a device is logically connected to a channel path until the time it is determined that a new sequence code applies. This code may also be used when the channel subsystem places a channel path in the polling or idle state and it is impossible to determine that code 010 or 011 applies. It may also be used at other times when a channel path cannot distinguish between code 010 or 011.

110 Reserved.

111 Reserved.

Figure 16-6 defines the relationship between indications provided as subchannel-logout data and the appropriate SCSW bits.

| Subchannel-Logout Condition Indicated | Logout Condition for SCSW Indication of[1] | | |
|---|---|---|---|
| | CDC | CCC | IFCC |
| Key check | V | V | - |
| Measurement-block-program check[2] | - | - | - |
| Measurement-block-data check[2] | - | - | - |
| Measurement-block-protection check[2] | - | - | - |
| CCW check | - | V | - |
| IDAW check | - | V | - |
| Last-path-used mask[3] | V | V | V |
| Field-validity flags | V | V | V |
| Termination code[3] | V | V | V |
| Device-status check | - | - | V |
| Secondary error | - | V | V |
| I/O-error alert | - | - | V |
| Sequence code[3] | V | V | V |

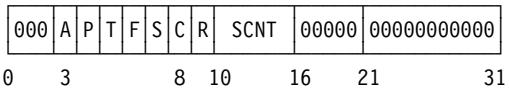
**Explanation:**

- \- No relationship.
- [1] When more than one SCSW indication is signaled, the subchannel-logout conditions that are valid are the logical OR for each of the respective SCSW indications.
- [2] Only one measurement-block check may be indicated in a specific subchannel logout.
- [3] This field has a field-validity flag.
- CCC Channel-control check.
- CDC Channel-data check.
- IFCC Interface-control check.
- V Bit setting valid.

*Figure 16-6. Relationship between Subchannel-Logout Data and SCSW Bits*

## Extended-Report Word

The extended-report word (ERW) provides information to the program describing specific conditions that may exist at the device, subchannel, or channel subsystem. The ERW is stored whenever the extended-status word is stored. When the extended-status-word-format bit, bit 5 of word 0 of the SCSW, and the extended-control bit, bit 14 of word 0 of the SCSW, are both zeros, the ERW contains all zeros. When the extended-status-word-format bit or the extended-control bit or both are ones, the ERW has the following format:




***Authorization Check (A):*** Bit 3, when one, indicates that the start or resume function was terminated because the channel subsystem has been placed in the isolated state in which pending I/O operations are not initiated and I/O operations cur-



LzLabs GmbH. Ex. 1008-744

rently being performed either are in the process of being terminated or have been terminated.

***Path Verification Required (P):*** Bit 4, when one, indicates that the program must verify the identity of the device. The LPUM, when valid, indicates the channel path for which device verification is to be performed. When a valid LPUM is not available, the identity of the device must be verified for each available channel path.

***Channel-Path Timeout (T):*** Bit 5, when one, indicates that, during a signaling sequence, an appropriate signal from the device did not occur within a predetermined time interval. Bit 5 is meaningful when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, and the interface-control-check bit, bit 14 of word 2 of the SCSW, are both ones.

***Failing-Storage-Address Validity (F):*** Bit 6, when one, and when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, is also one, indicates that the channel subsystem has detected an invalid CBC on a CCW, a data location, an IDAW, or the respective associated key and has stored, in words 2 and 3 of the ESW, the absolute address of a location associated with the invalid CBC. When an ERW is stored with bit 6 set to zero, zeros are stored in words 2 and 3 of the ESW.

***Concurrent Sense (S):*** Bit 7, when one, indicates that the concurrent-sense facility has placed sense information accepted from the device in the extended-control word and has stored a value, in bit positions 10-15 of the ERW, that specifies the number of sense bytes that have been stored in the extended-control word. When bit 7 is one, bit 14 of word 0 of the SCSW is also one.

***Concurrent-Sense Count (SCNT):*** When bit 7 is one, bit positions 10-15 contain a value, in the range 1-32, that specifies the number of sense bytes stored into the extended-control word by the concurrent-sense facility. When bit 7 is zero, bit positions 10-15 contain zeros.

***Secondary-CCW-Address Validity (C):*** Bit 8, when one, and when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, is also one, indicates that the channel subsystem has detected an error condition that precludes the continued performance of an I/O operation. When prefetching applies (bit 9 of word 1 of the ORB is one) and certain error conditions identified by channel-control check, channel-data check, or interface-control check are recognized by the channel subsystem, situations may exist where the termination point of execution of the channel program differs between the channel subsystem and the I/O device. To properly identify the termination points, bit 8 is set to one, and a second CCW address (secondary-CCW address) is provided in the ESW and designates the last CCW executed at the device. When the validity bit is zero for the previously mentioned errors, the channel subsystem was unable to determine the termination point of the control-unit execution, and the secondary-CCW-address field contains zeros.

Bit 8 is not set to one unless the program has permitted prefetching by setting the prefetch-control bit, bit 9 of word 1 of the ORB, to one for the channel program in execution.

***Failing-Storage-Address Format (R):*** Bit 9 indicates the format of the failing-storage address when the failing-storage-address validity bit, bit 6 of the ERW, is one. When bit 6 is zero, bit 9 is not meaningful and is stored as zero. When bit 6 is one and bit 9 is zero, a format-1 failing-storage address is stored in words 2 and 3 of the ESW. When bit 6 is one and bit 9 is one, a format-2 failing-storage address is stored in words 2 and 3. See “Failing-Storage Address” below for a description of format-1 and format-2 addresses.

## Failing-Storage Address

Words 2 and 3 of the extended-status contain a 24-, 31-, or 64-bit absolute address. When the failing-storage-address-validity flag, bit 6 of the ERW, is one, words 2 and 3 contain either a format-1 failing-storage address or a format-2 failing-storage address. When bit 6 is zero, words 2 and 3 are stored as zeros. When bit 6 is one, the failing-storage-address field designates a byte location within the invalid checking block associated with an invalid CBC for a CCW, data location, IDAW, or their respective associated key.

The form of the address stored in words 2 and 3 depends on the format-2-IDAW control, bit 14 of word 1 of the ORB, when an error condition is detected. When the control is zero, specifying that fullword IDAWs containing 31-bit addresses are used, a format-1 address is stored, and the failing-storage-address-format bit, bit 9 of the


LzLabs GmbH. Ex. 1008-745

ERW, is stored as zero. When the format-2-IDAW control is one, specifying that doubleword IDAWs containing 64-bit addresses are used, a format-2 address is stored, and the failing-storage-address-format bit is stored as one.

When a format-1 address is stored, bits 1-31 of word 2 form the address associated with the reported error condition, and bit 0 of word 2 and all of word 3 are stored as zeros. When a format-2 address is stored, bits 0-31 of word 2 followed by bits 0-31 of word 3 form the 64-bit address associated with the reported error condition.

### Secondary-CCW Address

When the subchannel-status field indicates channel-control check, channel-data check, or interface-control check and the secondary-CCW-address-validity flag, bit 8 of word 1, is one, bits 1-31 of word 4 form an absolute address of the last CCW executed by the I/O device at the point the reported check condition caused channel-program termination. When provided, the secondary-CCW address may be used for recovery purposes. When the secondary-CCW-address-validity flag is zero, this field contains zeros.

## Extended-Status Format 1

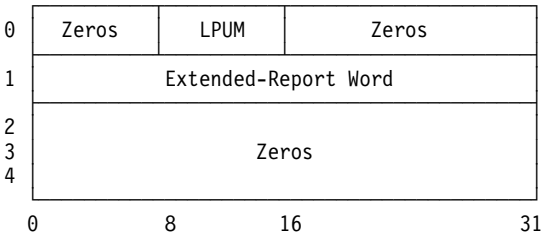
The ESW stored by TEST SUBCHANNEL is a format-1 ESW when all of the following conditions are met:

1. The extended-status-word-format bit, bit 5 of word 0 of the SCSW, is zero.
2. The subchannel status-control field has the status-pending bit, bit 31 of word 0 of the SCSW, set to one, together with:
   a. The primary-status bit, bit 29 of word 0 of the SCSW, alone,
   b. The primary-status bit and other status-control bits, or
   c. The intermediate-status bit, bit 28 of word 0 of the SCSW, and the suspended bit, bit 26 of word 0 of the SCSW.
3. At least one of the following conditions is indicated:
   a. The device-connect-time-measurement mode is inactive.
   b. The channel-subsystem-timing facility is not available for the subchannel.
   c. The subchannel is not enabled for the device-connect-time-measurement mode.

Zeros are stored in bytes 0, 2, and 3 of word 0, and the LPUM is stored in byte 1 of word 0; an ERW is stored in word 1; zeros are stored in words 2-4.

The device-connect-time-measurement mode is made inactive when SET CHANNEL MONITOR is executed and bit 31 of general register 1 is zero.

A format-1 ESW has the following format:




***Last-Path-Used Mask (LPUM):*** For a definition of the LPUM, see "Last-Path-Used Mask (LPUM)" on page 16-34.

***Extended-Report Word (ERW):*** For a definition of the ERW, see "Extended-Report Word" on page 16-36.

## Extended-Status Format 2

The ESW stored by TEST SUBCHANNEL is a format-2 ESW when all of the following conditions are met:

1. The extended-status-word-format bit, bit 5 of word 0 of the SCSW, is zero.
2. The channel-subsystem-timing facility is available for the subchannel.
3. The subchannel is enabled for the device-connect-time-measurement mode.
4. The device-connect-time-measurement mode is active.
5. The subchannel status-control field has the status-pending bit, bit 31 of word 0 of the SCSW, set to one, together with:
   a. The primary-status bit, bit 29 of word 0 of the SCSW, alone,



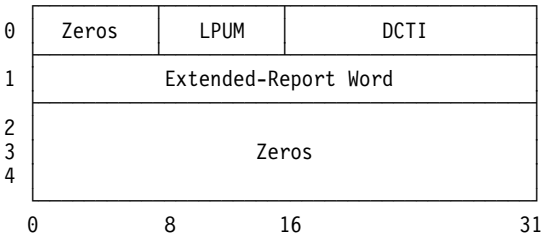
LzLabs GmbH. Ex. 1008-746

b. The primary-status bit and other status-control bits, or

c. The intermediate-status bit, bit 28 of word 0 of the SCSW, and the suspended bit, bit 26 of word 0 of the SCSW.

Zeros are stored in byte 0 of word 0, the LPUM is stored in byte 1 of word 0, and the device-connect time is stored in bytes 2 and 3 of word 0; an ERW is stored in word 1; zeros are stored in words 2-4.

A format-2 ESW has the following format:




***Last-Path-Used Mask (LPUM):*** For a definition of the LPUM, see "Last-Path-Used Mask (LPUM)" on page 16-34.

***Device-Connect-Time Interval (DCTI):*** Bit positions 16-31 contain the binary count of time increments accumulated by the channel subsystem during the time that the channel subsystem and the device were actively communicating and the subchannel was subchannel active. The time increment of the DCTI is 128 microseconds.

If the above conditions for the storing of the DCTI value in the ESW are met but the device-connect-time-measurement mode was made active by SET CHANNEL MONITOR subsequent to the execution of START SUBCHANNEL for this subchannel, the DCTI value stored is greater than or equal to zero and less than or equal to the correct DCTI value.

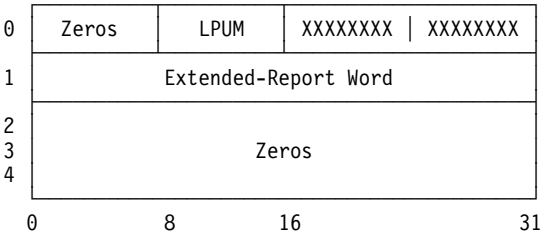
**Note:** The DCTI value stored in the ESW is the same as that used to update the corresponding measurement-block data for the subchannel if the measurement-block-update mode is in use for the subchannel. If the measurement-block-update mode for the channel subsystem is active and the subchannel is enabled for the device-connect-time-measurement mode but no DCTI value is stored in the ESW (because of the presence of subchannel-logout information), or if the DCTI is zero, then nothing is added to the corresponding measurement-block data.

***Extended-Report Word (ERW):*** For a definition of the ERW, see "Extended-Report Word" on page 16-36.

## Extended-Status Format 3

The ESW stored by TEST SUBCHANNEL is a format-3 ESW when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, is zero and the subchannel is status pending with (1) secondary status, alert status, or both when primary status is not also present, or (2) intermediate status when the subchannel is not suspended. Zeros are stored in byte 0 of word 0, and the LPUM is stored in byte 1 of word 0. Bytes 2 and 3 of word 0 contain unpredictable values; an ERW is stored in word 1; zeros are stored in words 2-4.

A format-3 ESW has the following format:




***Last-Path-Used Mask (LPUM):*** For a definition of the LPUM, see "Last-Path-Used Mask (LPUM)" on page 16-34.

An "X" in the format indicates the bit may be zero or one.

***Extended-Report Word (ERW):*** For a definition of the ERW, see "Extended-Report Word" on page 16-36.

Figure 16-7 on page 16-40 summarizes the conditions at the subchannel under which each type of information is stored in the ESW.


LzLabs GmbH. Ex. 1008-747

| Subchannel Conditions When IRB Is Stored: Subchannel-Status Word: Status-Control Field AIPSX | L Bit | Sus-pended Bit | Device-Connect-Time-Msrmnt Mode | Path-Management-Control Word: Timing-Facility Bit | Path-Management-Control Word: Device-Connect-Time-Msrmnt-Mode-Enable Bit | Extended-Status Word (ESW), Word 0: Format | Extended-Status Word (ESW), Word 0: Contents Bytes 0,1,2,3 |
|---|---|---|---|---|---|---|---|
| ----0 | //// | //// | //// | //// | //// | U | **** |
| 00001 | 0 | //// | //// | //// | //// | U | **** |
| 01001 | 0 | 0 | //// | //// | //// | U | **** |
| 01001 | 0 | 1 | Inactive | //// | //// | 1 | ZMZZ |
| 01001 | 0 | 1 | Active | 0 | //// | 1 | ZMZZ |
| 01001 | 0 | 1 | Active | 1 | 0 | 1 | ZMZZ |
| 01001 | 0 | 1 | Active | 1 | 1 | 2 | ZMDD |
| **1*1 | 0 | //// | Inactive | //// | //// | 1 | ZMZZ |
| **1*1 | 0 | //// | Active | 0 | //// | 1 | ZMZZ |
| **1*1 | 0 | //// | Active | 1 | 0 | 1 | ZMZZ |
| **1*1 | 0 | //// | Active | 1 | 1 | 2 | ZMDD |
| **011 | 0 | //// | //// | //// | //// | 3 | ZM** |
| 1*001 | 0 | //// | //// | //// | //// | 3 | ZM** |
| ****1 | 1 | //// | //// | //// | //// | 0 | RRRR |

**Explanation:**

- \- Defined to be not meaningful when X is zero.
- * Bits may be zeros or ones.
- / Information not relevant in this situation.
- A Alert status.
- D Accumulated device-connect-time-interval (DCTI) value stored in bytes 2 and 3.
- I Intermediate status.
- L Extended-status-word format.
- M Last-path-used mask (LPUM) stored in byte 1.
- P Primary status.
- R Subchannel-logout information stored in word 0.
- S Secondary status.
- U No format defined.
- X Status pending.
- Z Bits are stored as zeros.

*Figure 16-7. Information Stored in ESW*

## Extended-Control Word

The extended-control word, which is words 8-15 of an interruption-response block (see “Interruption-Response Block” on page 16-6), provides additional information to the program describing conditions that may exist at the channel subsystem, subchannel, or device. The extended-control (E) bit, bit 14 of word 0 of the SCSW, when one, indicates that model-dependent information or concurrent-sense information has been stored in the extended-control word.

The information provided in the extended-control word is as follows:

| SCSW Bits 5 | SCSW Bits 14 | ERW Bit 7 | ERW Bits 10-15 | ECW Words 0-7 |
|---|---|---|---|---|
| 0 | 0 | 0 | Zeros | Unpredictable[1] |
| 0 | 1 | 0 | ([5]) | ([5]) |
| 0 | 1 | 1 | ([3]) | Concurrent-sense information[4] |
| 1 | 0 | 0 | Zeros | Unpredictable[1] |
| 1 | 1 | 0 | Zeros | Model-dependent information[2] |
| 1 | 1 | 1 | ([3]) | Concurrent-sense information[4] |

**Explanation:**

[1] If stored, the value of these words is unpredictable.

[2] Unused bits in the model-dependent information are stored as zeros.

[3] Bit positions 10-15 contain a value equal to the number of sense bytes returned.

[4] Unused bytes in the concurrent-sense information are stored as zeros.

[5] The combination of SCSW bit 5 as 0, SCSW bit 14 as one, and ERW bit 7 as zero does not occur.

## Extended-Measurement Word

The extended-measurement word (EMW) provides I/O measurement information to the program for the most recent start or resume operation performed at the subchannel. When the extended-I/O-measurement-word facility is installed and enabled, the EMW is conditionally stored in words 16-23 of the IRB designated by the second operand of TEST SUBCHANNEL when TEST SUBCHANNEL is executed. The EMW is stored by TEST SUBCHANNEL when all of the following conditions are met:

1. The extended-status-word-format (L) bit (bit 5, word 0 of the SCSW) is zero.
2. The channel-subsystem-timing facility is available for the subchannel, as indicated by the timing facility bit (T) at the subchannel.
3. The extended-I/O-measurement-word-mode enable bit (X) at the subchannel is one.
4. The subchannel status-control field has the status-pending bit (bit 31, word 0 of the SCSW) set to one, together with:
   a. The primary-status bit (bit 29, word 0 of the SCSW) alone, or



LzLabs GmbH. Ex. 1008-748

b. The primary-status bit and other status-control bits, or

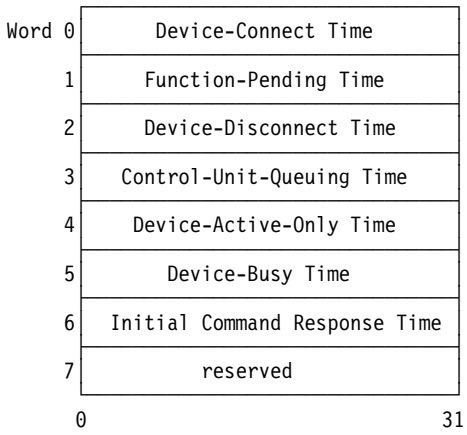
c. The secondary-status bit (bit 29, word 0 of the SCSW) alone, or

d. The intermediate-status bit (bit 28, word 0 of the SCSW) and the suspended bit (bit 26, word 0 of the SCSW).

Words 16-23 of the IRB contain unpredictable values when the extended-I/O-measurement-word facility is installed and enabled but the conditions listed above are not met.

When the extended-I/O-measurement-word facility is not installed, or the facility is installed but not enabled, words 16-23 of the IRB are not accessed by the channel subsystem.

The format of the extended-measurement word is shown below:




Each field in the EMW, when valid, contains a 32 bit binary count in which each increment of the count represents a value of 0.5 microseconds. A value of 00000000 hex represents a time period of zero seconds; the maximum value of FFFFFFFF hex represents approximately 35.79 minutes.

The accuracy of each of the measurement fields stored by the measurement facility is undefined and may vary depending on the resolution of timers implemented at the channel subsystem, the types of channels used to perform the operation, the capabilities of control units accessed during the operation, and the overall length of the I/O operation that was performed. The maximum value of FFFFFFFF hex is stored if a counter overflows; the program is not alerted when an overflow occurs.

***Device-Connect Time:*** Bit positions 0-31 of word 0 contain the measured device-connect time for the operation. The device-connect time is the sum of the time intervals measured whenever the device is logically connected to a channel path while the subchannel is subchannel active and the device is actively communicating with the channel path, as defined in the section "Device-Connect Time" on page 17-5.

***Function-Pending Time:*** Bit positions 0-31 of word 1 contain the SSCH- or RSCH-function-pending time for the operation. Function-pending time is the time interval between acceptance of the start function (or resume function if the subchannel is in the suspended state) at the subchannel and acceptance of the first command associated with the initiation or resumption of channel-program execution at the device, as defined in the section "Function-Pending Time" on page 17-5.

***Device-Disconnect Time:*** Bit positions 0-31 of word 2 contain the device-disconnect time for the operation. Device-disconnect time is the sum of the time intervals measured whenever the device is logically disconnected from the channel subsystem while the subchannel is subchannel-active, as defined in the section "Device-Disconnect Time" on page 17-5.

***Control-Unit-Queuing Time:*** Bit positions 0-31 of word 3 contain the control-unit-queuing time for the operation. Control-unit-queuing time is the sum of the time intervals measured by the control unit whenever the device is logically disconnected from the channel subsystem during an I/O operation while the device is busy with an operation initiated from a different system, as defined in the section "Control-Unit-Queuing Time" on page 17-6.

***Device-active-only time:*** Bit positions 0-31 of word 4 contain the device-active-only time for the operation. Device-active-only time is the sum of the time intervals when the subchannel is device-active but not subchannel-active at the end of an I/O operation or chain of I/O operations initiated by a start function or resume function, as defined in section "Device-Active-Only Time" on page 17-6.


LzLabs GmbH. Ex. 1008-749

***Device-busy time:*** Bit positions 0-31 of word 5 contain the device-busy time for the operation. Device-busy time is the sum of the time intervals when the device is found to be device busy during an attempt to initiate a start function or resume function at the subchannel, as defined in the section "Device-Busy Time" on page 17-6.

***Initial Command Response Time:*** Bit positions 0-31 of word 6 contain the initial-command-response time for the operation. The initial-command-response time for an operation is the time interval beginning from when the first command of the channel program is sent to the device until the device indicates it has accepted the command.



LzLabs GmbH. Ex. 1008-750

# Chapter 17. I/O Support Functions


Channel-Subsystem Monitoring . . . . . 17-1
Channel-Subsystem Timing . . . . . 17-2
Channel-Subsystem Timer . . . . . 17-2
Measurement-Block Update . . . . . 17-3
Measurement Block . . . . . 17-3
| Measurement-Block Format . . . . . 17-7
Measurement-Block Origin . . . . . 17-7
| Measurement-Block Address . . . . . 17-8
Measurement-Block Key . . . . . 17-8
Measurement-Block Index . . . . . 17-8
Measurement-Block-Update Mode . . . 17-8
| Measurement-Block-Format Control . . 17-9
Measurement-Block-Update Enable . . 17-9
Control-Unit-Queuing Measurement . . 17-9
Control-Unit-Defer Time . . . . . 17-9
Device-Active-Only Measurement . . . . 17-9
| Initial-Command-Response
| Measurement . . . . . 17-10
Time-Interval-Measurement Accuracy .17-10
Device-Connect-Time Measurement . . . 17-10
Device-Connect-Time-Measurement
Mode . . . . . 17-10
Device-Connect-Time-Measurement
Enable . . . . . 17-11
| Extended Measurement Word . . . . . 17-11
| Extended-Measurement-Word Enable 17-11
Signals and Resets . . . . . 17-12
Signals . . . . . 17-12
Halt Signal . . . . . 17-12
Clear Signal . . . . . 17-12
Reset Signal . . . . . 17-13
Resets . . . . . 17-13
Channel-Path Reset . . . . . 17-13
I/O-System Reset . . . . . 17-13
Externally Initiated Functions . . . . . 17-17
Initial Program Loading . . . . . 17-17
Reconfiguration of the I/O System . . . . 17-20
Status Verification . . . . . 17-20
Address-Limit Checking . . . . . 17-20
Configuration Alert . . . . . 17-21
Incorrect-Length-Indication Suppression . . 17-21
Concurrent Sense . . . . . 17-21
Channel-Subsystem Recovery . . . . . 17-21
Channel Report . . . . . 17-22
Channel-Report Word . . . . . 17-23
Channel-Subsystem-I/O-Priority Facility . . 17-25
Number of
Channel-Subsystem-Priority Levels 17-26


The I/O support functions are those functions of the channel subsystem that are not directly related to the initiation or control of I/O operations. The following I/O support functions are described in this chapter:

- Channel-subsystem monitoring
- Signals and resets
- Externally initiated functions
- Status verification
- Address-limit checking
- Configuration alert
- Incorrect-length-indication suppression
- Concurrent sense
- Channel-subsystem recovery
- I/O-priority facility

## Channel-Subsystem Monitoring

Monitoring facilities are provided in the channel subsystem so that the program can retrieve measured values on performance for a designated subchannel. The use of these facilities is under program control by means of the execution of the | SET CHANNEL MONITOR instruction and the | MODIFY SUBCHANNEL instruction.

The principal components of the channel-subsystem-monitoring facilities are the channel-subsystem-timing facility, measurement-block-update facility, and device-connect-time-measurement facility. The measurement-block-update facility and device-connect-time-measurement facility both use the channel-subsystem-timing facility but otherwise are logically distinct and operate independent of one another.

| Other components of the channel-
| subsystem-monitoring facilities are the control-
| unit-queuing-measurement facility, the control-

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-751

unit-defer-time facility, the device-active-only-measurement facility, the initial-command-response-measurement facility, the extended-I/O-measurement-block facility, and the extended-I/O-measurement-word facility. These enhance the measurements of the measurement-block-update facility if they are available as described in later sections, where each of the facilities that constitute the channel-subsystem-monitoring facilities is described in this chapter.

## Channel-Subsystem Timing

The channel-subsystem-timing facility provides the channel subsystem with the capability of measuring the elapsed time required for performing several different phases of the processing of a start function initiated by START SUBCHANNEL. These elapsed-time measurements are used by both the measurement-block-update facility and the device-connect-time-measurement facility to provide subchannel performance information to the program.

While every channel subsystem has a channel-subsystem-timing facility, it may or may not be provided for use with all subchannels. Subchannels for which the facility is provided have the timing-facility bit, bit 14 of word 1, stored as one in the associated subchannel-information block. (See “Timing Facility (T)” on page 15-4.) If the channel-subsystem-timing facility is not provided for the subchannel, the subchannel's timing-facility bit is stored as zero.

Subchannels that do not have the channel-subsystem-timing facility provided are those for which the characteristics of the associated device, the manner in which it is attached to the channel subsystem, or the channel-subsystem resources required to support the device are such that use of the channel-subsystem-timing facility is precluded.

The channel-subsystem-timing facility consists of at least one channel-subsystem timer and the associated logic and storage required for computing and recording the elapsed-time intervals for use by the two measurement facilities. The aspects of the channel-subsystem-timing facility that are of importance to the program are described below.

### Channel-Subsystem Timer

Each channel-subsystem timer is a binary counter that is not accessible to the program. The channel-subsystem timer provides a minimum timer resolution of 128 microseconds. A timer resolution of 1.0 microseconds is provided when FICON-I/O-interface channel paths are supported. When incrementing the channel-subsystem timer causes a carry out of the leftmost bit position, the carry is ignored, and counting continues from zero. No indications are generated as a result of the overflow.

Just as every CPU has access to a TOD clock, every channel subsystem has access to at least one channel-subsystem timer. When multiple channel-subsystem timers are provided, synchronization among these timers is also provided, creating the effect that all the timing facilities of the channel subsystem share a single timer. Synchronization among these timers may be supplied either through some TOD clock or independently by the channel subsystem.

If the TOD clocks are not synchronized, the elapsed times measured by the channel-subsystem-timing facility may have unpredictable values for some or all of the subchannels, depending on the model and on the particular channel-subsystem timer and the way the associated devices are physically attached to the system. The values are unpredictable for those devices attached to the system by separately configurable channel paths whose associated CPU TOD clocks are not synchronized.

***Synchronization:*** If either the measurement-block-update mode or the device-connect-time-measurement mode is active and any of the channel-subsystem timers is found to be out of synchronization, a channel-subsystem-timer-sync check is recognized, and a channel report is generated to alert the program (see “Channel-Subsystem Recovery” on page 17-21). If neither of these modes is active, the lack of synchronization is not recognized.



LzLabs GmbH. Ex. 1008-752

## Measurement-Block Update

The measurement-block-update facility provides the program with the capability of accumulating performance information for subchannels that are enabled for the measurement-block-update mode when the measurement-block-update mode is active. A subchannel is enabled for the measurement-block-update mode by setting bit 11 of word 1 of the SCHIB operand to one and then issuing MODIFY SUBCHANNEL. The measurement-block-update mode is made active by the execution of SET CHANNEL MONITOR when bit 30 of general register 1 is one.

When the measurement-block-update mode is active and the subchannel is enabled for the measurement-block-update mode, information is accumulated in a measurement block associated with the subchannel. A measurement block is either a 32-byte area (format-0 measurement block) or a 64-byte area (format-1 measurement block) in main storage that is associated with a subchannel for the purpose of accumulating measurement data.

For format-0 measurement blocks, the program specifies a contiguous area of absolute storage, referred to as the measurement-block area, and subdivides this area into 32-byte blocks, one block for each subchannel for which measurement data is to be accumulated. The measurement-block-update facility uses the measurement-block index contained at the subchannel in conjunction with the measurement- block origin established by the execution of SET CHANNEL MONITOR to compute the absolute address of the measurement block associated with a subchannel.

For format-1 measurement blocks, the program provides a 64-byte contiguous area of absolute storage for the subchannel. The measurement-block-update facility uses the measurement-block address provided by the MSCH instruction to access the measurement block.

Measurement data is stored in the measurement block associated with the subchannel each time an I/O operation or chain of I/O operations initiated by a START SUBCHANNEL instruction or RESUME SUBCHANNEL instruction is suspended or is completed at the device The completion of an I/O operation or chain of I/O operations at the device is normally determined when secondary status is accepted from the device.

The measurement data accumulated in the format-0 and format-1 measurement blocks by the measurement-block-update facility is described in the following section, “Measurement Block.”

### Measurement Block

A measurement block is either a 32-byte area (format-0 measurement block) at a location designated by the program by its use of a measurement-block-origin address in conjunction with the measurement-block index, or a 64-byte area (format-1 measurement block) at a location designated by the measurement-block address provided in the SCHIB during the execution of the MODIFY SUBCHANNEL instruction. The measurement block contains the accumulated values of the measurement data described below. When the measurement-block-update mode is active and the subchannel is enabled for measurement-block update, the measurement-block-update facility accumulates the values for the measurement data that accrue during the performance of an I/O operation or chain of I/O operations initiated by START SUBCHANNEL.

When the I/O operation or chain of I/O operations is suspended or completed and no error condition is encountered, the accrued values are added to the accumulated values in the measurement block for that subchannel. If an error condition is detected and subchannel-logout information is stored in the extended-status word (ESW), the accrued values are not added to the accumulated values in the measurement block for the subchannel, and the two count fields in the measurement block are not incremented.

If (1) any of the accrued time values is detected to exceed the internal storage provided for containing these values, (2) the control unit cannot provide an accurate queuing time or defer time for the current operation, or (3) the channel subsystem successfully recovers from certain error conditions, none of the accrued values is added to the measurement block for the subchannel, and the sample count in the measurement block is not incremented, but the SSCH+RSCH count in the block is incremented.

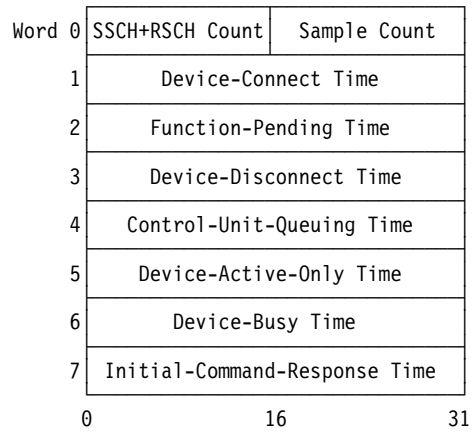

LzLabs GmbH. Ex. 1008-753

On models without z/Architecture installed, references to the measurement block by the measurement-block-update facility, in order to accumulate measurement data at the suspension or completion of an I/O function, appear block-concurrent to CPUs. CPU accesses to the block, either fetches or stores, are inhibited during the time the measurement-block update is being performed. If z/Architecture is installed, the references are single-access references and appear to be word concurrent as observed by CPUs.

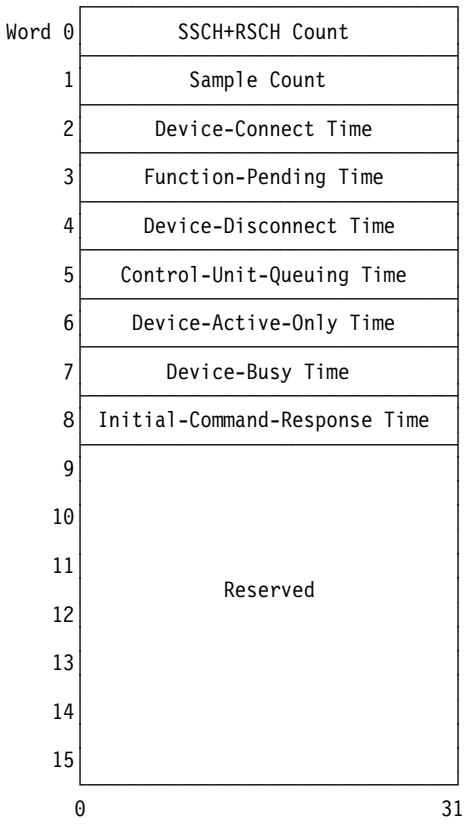
The measurement-block-update facility updates all fields in the measurement block that are required to be updated for a suspended I/O operation prior to putting the subchannel into the suspended state. The measurement-block-update facility updates all fields in the measurement block that are required to be updated for a completed I/O operation prior to making the subchannel status pending with secondary status or, if the subchannel is start pending for a subsequent operation, prior to initiating the start function.

A format-0 measurement block is stored when the measurement-block-format-control bit at the subchannel is zero; a format-1 measurement block is stored when the measurement-block-format-control bit at the subchannel is one.

The format-0 measurement block has the following format:




The format-1 measurement block has the following format:




***SSCH+RSCH Count:*** Bits 0-15 of word 0 in the format-0 measurement block and bits 0-31 of word 0 in the format-1 measurement block are used as a binary counter. During the performance of a start function for which measurement-block update is active, when (1) the secondary status condition is recognized or (2) the suspend function is performed, the counter is incremented by one, and the measurement data is stored. The counter wraps around from the maximum value to 0. The program is not alerted when counter overflow occurs.

If the measurement-block-update mode is active and the subchannel is enabled for measuring, the SSCH+RSCH count is incremented even when the lack of measured values for an individual start function precludes the updating of the remaining fields of the measurement block or when the timing-facility bit for the subchannel is zero. The SSCH+RSCH count is not incremented if the measurement-block-update mode is inactive, if the subchannel is not enabled for the measurement-block update, or if subchannel-logout information has been generated for the start function.



LzLabs GmbH. Ex. 1008-754

***Sample Count:*** Bits 16-31 of word 0 in the format-0 measurement block and bits 0-31 of word 1 in the format-1 measurement block are used as a binary counter. When the time-accumulation fields following word 0 of the measurement block are updated, the counter is incremented by one. On some models, certain conditions may prevent the measurement-block-update facility from obtaining the accrued values of the measurement data for an individual start function, even when the measurement-block-update mode is active and the subchannel is enabled for that mode. The control unit may also signal that it was not able to accumulate an accurate queuing time. In these situations, the sample-count field is not incremented.

The counter wraps around from the maximum value to 0. The program is not alerted when counter overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

The System Library publication for the system model specifies the conditions, if any, that preclude the updating of the sample count and time-accumulation fields of the measurement block.

***Device-Connect Time:*** Bit positions 0-31 of word 1 in the format-0 measurement block and bit positions 0-31 of word 2 in the format-1 measurement block contain the accumulation of measured device-connect-time intervals. The device-connect-time interval (DCTI) is the sum of the time intervals measured whenever the device is logically connected to a channel path while the subchannel is subchannel active and the device is actively communicating with the channel path. The device-connect time does not include the intervals when a device is logically connected to a channel path but is not actively communicating with the channel. The device reports the accumulation of time intervals when the device is logically connected but not actively communicating with the channel path as control-unit-defer time. The control-unit-defer time is not included in the device-connect-time measurement but, instead, is added to the accrued device-disconnect-time measurement for the operation.

The time intervals are stored using a resolution of 128 microseconds. The accumulated value is modulo approximately 152.71 hours, and the program is not alerted when an overflow occurs. This field is not updated if (1) the channel-subsystem-timing facility is not provided for the subchannel, (2) the measurement-block-update mode is inactive, or (3) any of the time values accrued for the current start function has been detected to exceed the internal storage in which it was accrued.

Accumulation of device-connect-time intervals for a subchannel and storing this data in the ESW are not affected by whether the measurement-block-update mode is active. (See "Device-Connect-Time Measurement" on page 17-10.)

***Function-Pending Time:*** Bit positions 0-31 of word 2 in the format-0 measurement block and bit positions 0-31 of word 3 in the format-1 measurement block contain the accumulated SSCH- and RSCH-function-pending time. Function-pending time is the time interval between acceptance of the start function (or resume function if the subchannel is in the suspended state) at the subchannel and acceptance of the first command associated with the initiation or resumption of channel-program execution at the device.

When channel-program execution is suspended because of a suspend flag in the first CCW of a channel program, the suspension occurs prior to transferring the first command to the device. In this case, the function-pending time accumulated up to that point is added to the value in the function-pending-time field of the measurement block. Function-pending time is not accrued while the subchannel is suspended. Function-pending time begins to be accrued again, in this case, when RESUME SUBCHANNEL is subsequently executed while the designated subchannel is in the suspended state.

The function-pending-time interval is stored using a resolution of 128 microseconds. The accumulated value is modulo approximately 152.71 hours, and the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

***Device-Disconnect Time:*** Bit positions 0-31 of word 3 in the format-0 measurement block and bit positions 0-31 of word 4 in the format-1 measurement block contain the accumulated device-disconnect time. Device-disconnect time is the sum of the time intervals measured whenever the device is logically disconnected from the channel


LzLabs GmbH. Ex. 1008-755

subsystem while the subchannel is subchannel active. The device-disconnect time also includes the sum of control-unit-defer-time intervals reported by the device during the I/O operation.

Device-disconnect time is not accrued while the subchannel is in the suspended state. Device-disconnect time begins to be accrued again, in this case, on the first device disconnection after channel-program execution has been resumed at the device (the subchannel is again subchannel active).

The device-disconnect-time interval is stored using a resolution of 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

The device-disconnect time does not include the interval between the primary status condition and the secondary status condition at the end of an I/O operation when the subchannel is no longer subchannel active but the I/O device is active. If the channel subsystem provides the device-active-only measurement facility, this time is accumulated in the device-active-only-time field of the measurement block.

***Control-Unit-Queuing Time:*** Bit positions 0-31 of word 4 in the format-0 measurement block and bit positions 0-31 of word 5 in the format-1 measurement block contain the accumulated control-unit-queuing time. Control-unit-queuing time is the sum of the time intervals measured by the control unit whenever the device is logically disconnected from the channel subsystem during an I/O operation while the device is busy with an operation initiated from a different system.

Control-unit-queuing time is not accrued while the subchannel is in the suspended state. Control-unit-queuing time may be accrued for the channel program after the subchannel becomes subchannel active following a successful resumption.

The control-unit-queuing-time field is updated such that bit 31 represents 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel, if the control-unit-queuing-measurement facility is not installed, or if the control unit does not provide a queuing time.

***Device-Active-Only Time:*** If the device-active-only-measurement facility is installed, bit positions 0-31 of word 5 in the format-0 measurement block and bit positions 0-31 of word 6 in the format-1 measurement block contain the accumulated device-active-only time. Device-active-only time is the sum of the time intervals when the subchannel is device active but not subchannel active at the end of an I/O operation or chain of I/O operations initiated by a start function or resume function.

Device-active-only time is not accumulated when the subchannel is device active during periods when the subchannel is active; such time is accumulated as device-connect time or device-disconnect time, as appropriate.

The device-active-only-time field is updated such that bit 31 represents 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

***Device-Busy Time:*** When the extended-I/O-measurement-block facility is installed, bit positions 0-31 of word 6 in the format-0 measurement block, and bit positions 0-31 of word 7 in the format-1 measurement block contain the accumulated device-busy time. Device-busy time is the sum of the time intervals when the subchannel is device busy during an attempt to initiate a start function or resume function at the subchannel.

The device-busy-time field is updated such that bit 31 represents 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

***Initial-Command-Response Time:*** Bit positions 0-31 of word 7 in the format-0 measurement block, and bit positions 0-31 of word 8 in the format-1 measurement block contain the accumulated initial-command-response time for the sub-



LzLabs GmbH. Ex. 1008-756

channel. The initial-command-response time for a start or resume function is the time interval beginning from when the first command of the channel program is sent to the device until the device indicates it has accepted the command.

The initial-command-response time is stored at a resolution of 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs.

This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel or if the initial-command-response-measurement facility is not installed.

***Control-Unit-Defer Time:*** Control-unit-defer time is the sum of the time intervals measured by the control unit whenever the device is logically connected to the channel subsystem during an I/O operation but is not actively communicating with the channel because of device-dependent delays in channel-program execution. The control-unit-defer time is not stored in the measurement block as a separate measurement field but, if the control-unit-defer-time facility is installed, is used in the calculation of device-connect-time measurement and device-disconnect-time measurement for an operation.

Control-unit-defer time, if provided by a control unit, is accrued while the device is logically connected to the channel. The time is reported to the channel when channel-end status is presented that causes a device disconnection or terminates the I/O operation. Control-unit-defer time is subtracted from the device-connect-time measurement and is added to the device-disconnect-time measurement reported for the operation.

***Reserved:*** The remaining words of the measurement block, along with any words associated with facilities that are not provided by the channel subsystem or the subchannel, are reserved for future use. They are not updated by the measurement-block-update facility.

**Programming Note:** On models that do not have the z/Architecture installed, it is possible for the program to fetch a portion of a measurement block immediately prior to a measurement block update, for the measurement-block-update facility to perform a block-concurrent update of the block, and for the program to then fetch the remainder of the now-updated measurement block. To ensure that a consistent measurement block is fetched, the program should re-fetch the sample-count field after fetching the entire measurement block and compare the re-fetched count with the sample-count value originally fetched. If the two values match, a consistent measurement block has been obtained. If they do not match, the entire measurement block should be re-fetched in the same manner.

On models that have the z/Architecture installed, it is possible for the program to fetch a portion of a measurement block immediately prior to, or during, a measurement block update by the measurement-block-update facility. To ensure that a consistent measurement block is fetched, the program should not fetch measurement data for a subchannel during the time from when a start or resume function is initiated at the subchannel until the operation is suspended or completes with secondary status.

## Measurement-Block Format

The measurement block is stored as either a format-0 measurement block or a format-1 measurement block. The format-0 measurement block is a 32-byte area (format-0 measurement block) at a location designated by the program using the measurement-block origin in conjunction with the measurement-block index. The format-1 measurement block is a 64-byte area (format-1 measurement block) at a location designated by the measurement-block address provided in the SCHIB during the execution of the MODIFY SUBCHANNEL instruction. The measurement-block-format-control bit at the subchannel indicates whether a format-0 or format-1 measurement block is stored when measurement-block-update mode is active and enabled at the subchannel.

## Measurement-Block Origin

The measurement-block origin is the beginning of the measurement-block area in main storage, used to store format-0 measurement blocks. The absolute address of the measurement-block origin, specified on a 32-byte boundary, is passed in general register 2 to the measurement-block-update facility when SET CHANNEL MONITOR is executed with bit 30 of general register 1 set to one.


LzLabs GmbH. Ex. 1008-757

## Measurement-Block Address

The measurement-block address is set at the subchannel through the execution of MODIFY SUBCHANNEL. The measurement block address specifies the absolute address of the beginning of the 64-byte area to be used for accumulating format-1 measurement-block parameters for that subchannel.

**Programming Note:** The initial value of the measurement-block address is zero. The program is responsible for setting the measurement-block address to the proper value prior to enabling the subchannel for the measurement-block-update mode for format-1 measurement blocks and making the mode active.

## Measurement-Block Key

Bits 0-3 of general register 1 form the four-bit access key to be used for subsequent measurement-block updates when SET CHANNEL MONITOR causes the measurement-block-update mode to be made active. The measurement-block key is passed to the measurement-block-update facility whenever the measurement-block-update mode is made active. The key is used for both format-0 and format-1 measurement block updates.

## Measurement-Block Index

The measurement-block index is set in the subchannel through the execution of MODIFY SUBCHANNEL. The measurement-block index specifies which 32-byte measurement block, relative to the measurement-block origin, is to be used for accumulating the format-0 measurement-block parameters for that subchannel. The location of the format-0 measurement block of a subchannel is computed by the measurement-block-update facility by appending five rightmost zeros to the measurement-block index of the subchannel and adding the result to the measurement-block origin. The result is the absolute address of the 32-byte format-0 measurement block for that subchannel. When the computed measurement-block address exceeds $2^{31}-1$, a measurement-block program-check condition is recognized, and measurement-block updating does not occur for the preceding subchannel-active period.

**Programming Note:** The initial value of the measurement-block index is zero. The program is responsible for setting the measurement-block index to the proper value prior to enabling the subchannel for the measurement-block-update mode and making the mode active. To ensure predictable results for the measured parameters in the measurement block, each subchannel for which measured parameters are to be accumulated must have a different value for its measurement-block index.

## Measurement-Block-Update Mode

The measurement-block-update mode is made active by the execution of SET CHANNEL MONITOR with bit 30 of general register 1 set to one. If bit 30 of general register 1 is zero when SET CHANNEL MONITOR is executed, the mode is made inactive. When the measurement-block-update mode is inactive, no measurement values are accumulated in main storage. When the measurement-block-update mode is made active, the contents of general register 2 are passed to the measurement-block-update facility as the absolute address of the measurement-block origin and is used to calculate the address of format-0 measurement blocks. The measurement-block origin is not used for format-1 measurement blocks. The MBK is also passed to the measurement-block-update facility as the access key to be used when updating either format-0 or format-1 measurement blocks for each subchannel. When the measurement-block-update mode is active, the measurement-block-update facility accumulates measurements in individual measurement blocks for subchannels whose measurement-block-update-enable bit is one. (See the section "Measurement Block" on page 17-3 for a description of the measured parameters.)

If the measurement-block-update mode is already active when SET CHANNEL MONITOR is executed, the values for the measurement-block origin and measurement-block key that are used for a subchannel enabled for measuring by the measurement-block-update facility are dependent upon whether SET CHANNEL MONITOR is executed prior to, during, or subsequent to the execution of START SUBCHANNEL for that subchannel. If SET CHANNEL MONITOR is executed prior to START SUBCHANNEL, the current measurement-block origin and measurement-block key are in control. If SET CHANNEL MONITOR is executed during or subsequent to execution of START SUBCHANNEL, it is unpredictable whether the measurement-block origin and



LzLabs GmbH. Ex. 1008-758

measurement-block key that are in control are old or current.

### Measurement-Block-Format Control

Bit 29, word 6, of the SCHIB is the measurement-block-format-control bit. This bit provides the capability of specifying whether a format-0 or format-1 measurement block is stored on a subchannel basis. The initial value of the bit is zero. When MODIFY SUBCHANNEL is executed with the measurement-block-format-control bit in the SCHIB operand set to one, the format-1 measurement block is specified for the subchannel. If the measurement-block-update mode is active and enabled at the subchannel, the measurement-block-update facility stores a format-1 measurement-block for the subchannel, starting with the next START SUBCHANNEL issued to that subchannel. Similarly, if MODIFY SUBCHANNEL is executed with measurement-block-format-control bit of the SCHIB operand set to zero by the program, the measurement-block-update facility stores a format-0 measurement-block for the subchannel, starting with the next START SUBCHANNEL issued to that subchannel.

### Measurement-Block-Update Enable

Bit 11 of word 1 of the SCHIB is the measurement-block-update-enable bit. This bit provides the capability of controlling the accumulation of measurement-block parameters on a subchannel basis. The initial value of the enable bit is zero. When MODIFY SUBCHANNEL is executed with the enable bit set to one in the SCHIB, the subchannel is enabled for the measurement-block-update mode. If the measurement-block-update mode is active, the measurement-block-update facility accumulates measurement-block parameters for the subchannel, starting with the next START SUBCHANNEL issued to that subchannel. Conversely, if MODIFY SUBCHANNEL is executed with bit 11 of word 1 of the SCHIB operand set to zero by the program, the subchannel is disabled for the measurement-block-update mode, and no additional measurement-block parameters are accumulated for that subchannel.

### Control-Unit-Queuing Measurement

The control-unit-queuing-measurement facility allows the channel subsystem to accept queuing times from control units and, in conjunction with the measurement-block-update facility, to accumulate those times in the measurement block.

The System Library publication for the control-unit model specifies its ability to supply queuing time. If a control-unit model is capable of supplying queuing time, the publication specifies the conditions that prevent the control unit from accumulating an accurate control-unit-queuing time.

### Control-Unit-Defer Time

The control-unit-defer-time facility allows the channel subsystem to accept defer times from control units and, in conjunction with the measurement-block-update facility, to modify the device-connect and device-disconnect times reported in the measurement block to reflect the defer time. The control-unit-defer time is subtracted from the device-connect-time measurement and is added to the device-disconnect-time measurement reported for an I/O operation.

The System Library publication for the control-unit model specifies its ability to supply defer time. If a control-unit model is capable of supplying defer time, the publication specifies the conditions that prevent the control unit from accumulating an accurate control-unit-defer time.

### Device-Active-Only Measurement

The device-active-only-measurement facility permits the channel subsystem to report the times that the device is disconnected between primary status and secondary status at the end of an I/O operation or chain of I/O operations.

When the device-active-only-measurement facility is not installed, measurement-block updates are performed when the subchannel becomes status pending for primary status. When the device-active-only-measurement facility is installed, the measurement-block updates are performed at the time that secondary status is accepted from the I/O device, in order that the device-active time between primary status and secondary status can be reported.


LzLabs GmbH. Ex. 1008-759

If the subchannel is start pending when secondary status is accepted from the I/O device and the measurement-block update is to be performed, the measurement-block update is performed prior to performing the start function. If measurement-block errors occur, they are reported to the program along with the secondary status instead of performing the start function.

## Initial-Command-Response Measurement

The initial-command-response-measurement facility allows the channel subsystem to calculate initial-command-response time for I/O operations and, in conjunction with the measurement-block-update facility, to accumulate those times in the measurement block. The initial-command-response time is accumulated in word 7 of the measurement block.

### Time-Interval-Measurement Accuracy

On some models, when time intervals are to be measured and condition code 0 is set for START SUBCHANNEL (or RESUME SUBCHANNEL in the case of a suspended subchannel), a period of latency may occur prior to the initiation of the function-pending time measurement. The System Library publication for the system model specifies the mean latency value and variance for each of the measured time intervals.

**Programming Notes:**

1. Excessive delays may be encountered by the channel subsystem when attempting to update measurement data if the program is concurrently accessing the same measurement-block area. A programming convention should ensure that the storage block designated by SET CHANNEL MONITOR is made read-only while the measurement-block-update mode is active.
2. To ensure that programs written to support measurement functions are executed properly, the program should initialize all the measurement blocks to zeros prior to making the measurement-block-update mode active. Only zeros should appear in the reserved and unused words of the measurement blocks.
3. When the incrementing of an accumulated value causes a carry to be propagated out of bit position 0, the carry is ignored, and accumulating continues from zero on.

## Device-Connect-Time Measurement

The device-connect-time-measurement facility provides the program with the capability of retrieving the length of time that a device is actively communicating with the channel subsystem while executing a channel program. The measured length of time that the device spends actively communicating on a channel path during the execution of a channel program is called the device-connect-time interval (DCTI). Control-unit-defer time is not included in the DCTI.

If timing facilities are provided for the subchannel, the DCTI value is passed to the program in the extended-status word (ESW) at the completion of the operation when the primary-status condition is cleared by TEST SUBCHANNEL and when TEST SUBCHANNEL clears an intermediate-status condition alone while the subchannel is suspended. The DCTI value passed in the ESW pertains to the previous subchannel-active period. The passing of the DCTI in the ESW is under program control by the SET CHANNEL MONITOR device-connect-time-measurement mode-control bit and the corresponding enable bit in the subchannel. However, the DCTI value is not stored in the ESW if the I/O function initiated by START SUBCHANNEL is terminated because of an error condition that is described by subchannel logout. See the section "Extended-Status Format 0" on page 16-32. In this case, the extended-status bit (L) of the SCSW is stored as one, indicating that the ESW contains logout information describing the error condition. See the section "Extended-Status Word" on page 16-32 for the description of the logout information. If the accrued DCTI value exceeded 8.388608 seconds during the previous subchannel-active period, then the maximum value (FFFF hex) is passed in the ESW.

### Device-Connect-Time-Measurement Mode

The device-connect-time-measurement mode is made active by the execution of SET CHANNEL MONITOR when bit 31 of general register 1 is one. If bit 31 of general register 1 is zero when SET CHANNEL MONITOR is executed, the mode is made inactive, and DCTIs are not passed to the



LzLabs GmbH. Ex. 1008-760

program. When timing facilities are provided for the subchannel, the device-connect-time-measurement mode is active, and the subchannel is enabled for the mode, the DCTI value is passed to the program in the ESW stored when TEST SUBCHANNEL (1) clears the primary-interruption condition with no logout information indicated in the SCSW (extended-status-word-format bit is zero) or (2) clears the intermediate-status condition alone while the subchannel is suspended.

If a start function is currently being executed with a subchannel enabled for the device-connect-time-measurement mode when SET CHANNEL MONITOR makes this mode active for the channel subsystem, the value of the DCTI stored under the appropriate conditions may be zero, a partial result, or the full and correct value, depending on the model and the progress of the start function at the time the mode was activated.

Provision of the DCTI value in the measurement-block area is not affected by whether the device-connect-time-measurement mode is active.

### Device-Connect-Time-Measurement Enable

Bit 12 of word 1 of the SCHIB is the device-connect-time measurement-mode-enable bit. This bit provides the program with the capability of selectively controlling the storing of DCTI values for a subchannel when the device-connect-time-measurement mode is active. The initial value of the enable bit is zero. When this enable bit is one in the SCHIB and MODIFY SUBCHANNEL is executed, the subchannel is enabled for the device-connect-time-measurement mode. If the device-connect-time-measurement mode is active, the device-connect-time-measurement facility begins providing DCTI values for the subchannel, starting with the next START SUBCHANNEL issued to the subchannel. In this situation, the DCTI values are provided in the ESW (see the section “Extended-Status Format 2” on page 16-38). Similarly, if MODIFY SUBCHANNEL is executed with bit 12 of word 1 of the SCHIB operand set to zero by the program, the subchannel is disabled for the device-connect-time-measurement mode, and no further DCTI values are passed to the program for that subchannel.

## Extended Measurement Word

The extended-I/O-measurement-word facility provides the program with the capability of retrieving measurement information for a channel program. The measurement information is stored into the extended-measurement word (EMW) in the Interruption Response Block when the extended-measurement-word enable bit is one at the subchannel. See the section “Extended-Measurement Word” on page 16-40 in Chapter 16, “I/O Interruptions,” for the description of the extended-measurement word.

When the extended-measurement-word is enabled for the subchannel, measurement values are passed to program in the EMW when TEST SUBCHANNEL clears a primary-status condition, secondary-status condition alone, or an intermediate-status condition alone while the subchannel is suspended. The measurement values stored in the EMW pertain to the previous subchannel-active and device-active period. Measurement values are not stored in the EMW if the I/O function initiated by START SUBCHANNEL is terminated because of an error condition that is described by subchannel logout (see the section “Extended-Status Format 0” on page 16-32). In this case, the extended-status bit (L) of the SCSW is stored as one, indicating that the ESW contains logout information describing the error condition. See the section “Extended-Status Word” on page 16-32 for the description of the logout information. If any of the accrued measurement values exceeded the maximum value capable of being measured during the previous subchannel-active and device-active period, then the maximum value is stored for that value in the EMW.

### Extended-Measurement-Word Enable

Bit 30 of word 6 of the SCHIB is the extended-measurement-word enable bit. This bit provides the program with the capability of selectively controlling the storing of measurement values for a subchannel. The initial value of the enable bit is zero. When this enable bit is one in the SCHIB and MODIFY SUBCHANNEL is executed, the subchannel is enabled for the extended-measurement-word and the extended-measurement-word facility begins providing measurement values for the subchannel starting with the next START SUBCHANNEL issued to the sub-


LzLabs GmbH. Ex. 1008-761

channel. Similarly, if MODIFY SUBCHANNEL is executed with bit 30, word 6, of the SCHIB operand set to zero by the program, the subchannel is disabled for the extended-measurement-word and no further measurement values are passed to the program for that subchannel.

## Signals and Resets

During system operation, it may become necessary to terminate an I/O operation or to reset either the I/O system or a portion of the I/O system. (The I/O system consists of the channel subsystem plus all of the attached control units and devices.) Various signals and resets are provided for this purpose. Three signals are provided for the channel subsystem to notify an I/O device to terminate an operation or perform a reset function or both. Two resets are provided to cause the channel subsystem to reinitialize certain information contained either at the I/O device or at the channel subsystem.

## Signals

The request that the channel subsystem initiate a signaling sequence is made by one of the following:

1. The program's issuance of the CLEAR SUBCHANNEL, HALT SUBCHANNEL, or RESET CHANNEL PATH instruction
2. The I/O device's signaling of I/O-error alert
3. The channel subsystem itself, upon detecting certain error conditions or equipment malfunctions

The three signals are the halt signal, the clear signal, and the reset signal.

### Halt Signal

The halt signal is provided so the channel subsystem can terminate an I/O operation. The halt signal is issued by the channel subsystem as part of the halt function performed subsequent to the execution of HALT SUBCHANNEL. The halt signal is also issued by the channel subsystem when certain error conditions are encountered.

For the parallel-I/O-interface type of channel path, the halt signal results in the channel subsystem using the interface-disconnect sequence control defined in the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974.

For the ESCON-I/O-interface type of channel path, the halt signal results in the channel subsystem using the cancel function defined in the System Library publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

For the FICON-I/O-interface type of channel path, the halt signal results in the channel subsystem using the cancel function defined in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

### Clear Signal

The clear signal is provided so the channel subsystem can terminate an I/O operation and reset status and control information contained at the device. The clear signal is issued as part of the clear function performed subsequent to the execution of CLEAR SUBCHANNEL. The clear signal is also issued by the channel subsystem when certain error conditions or equipment malfunctions are detected by the I/O device or the channel subsystem.

For the parallel-I/O-interface type of channel path, the clear signal results in the channel subsystem using the selective-reset sequence control defined in the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974.

For the ESCON-I/O-interface type of channel path, the clear signal results in the channel subsystem using the selective-reset function defined in the System Library publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

For the FICON-I/O-interface type of channel path, the clear signal results in the channel subsystem using the selective-reset function defined in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

If an I/O operation is in progress at the device and the device is actively communicating over a channel path in the performance of that I/O operation when a clear signal is received on that channel path, the device disconnects from that



LzLabs GmbH. Ex. 1008-762

channel path upon receiving the clear signal. Data transfer and any operation using the facilities of the control unit are immediately concluded, and the I/O device is not necessarily positioned at the beginning of a block. Mechanical motion not involving the use of the control unit, such as rewinding magnetic tape or positioning a disk-access mechanism, proceeds to the normal stopping point, if possible. The device may appear busy until termination of the mechanical motion or the inherent cycle of operation, if any, whereupon it becomes available. Status information in the device and control unit is reset, but an interruption condition may be generated upon the completion of any mechanical operation.

### Reset Signal

The reset signal is provided so the channel subsystem can reset all I/O devices on a channel path. The reset signal is issued by the channel subsystem as part of the channel-path-reset function performed subsequent to the execution of RESET CHANNEL PATH. The reset signal is also issued by the channel subsystem as part of the I/O-system-reset function.

For the parallel-I/O-interface type of channel path, the reset signal results in the channel subsystem using the system-reset sequence control defined in the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974.

For the ESCON-I/O-interface type of channel path, the reset signal results in the channel subsystem using the system-reset function defined in the System Library publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

For the FICON-I/O-interface type of channel path, the reset signal results in the channel subsystem using the system-reset function defined in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

## Resets

Two resets are provided so the channel subsystem can reinitialize certain information contained at either the I/O device or the channel subsystem. The request that the channel subsystem initiate one of the reset functions is made by one of the following:

1. The program's issuance of the RESET CHANNEL PATH instruction
2. The operator's activation of a system-reset-clear or system-reset-normal key or a load-clear or load-normal key
3. The channel subsystem itself upon detecting certain error conditions or equipment malfunctions

The resets are channel-path reset and I/O-system reset.

### Channel-Path Reset

The channel-path-reset facility provides a mechanism to reset certain indications that pertain to a designated channel path at all associated subchannels. Channel-path reset occurs when the channel subsystem performs the channel-path-reset function initiated by RESET CHANNEL PATH. (See “RESET CHANNEL PATH” on page 14-9.) All internal indications of dedicated allegiance, control unit busy, and device busy that pertain to the designated channel path are cleared in all subchannels, and reset is signaled on that channel path. The receipt of the reset signal by control units attached to that channel path causes all operations in progress and all status, mode settings, and allegiance, pertaining to that channel path, of the control unit and its attached devices to be reset. (See also the description of the system-reset-signal actions in “I/O-System Reset.”)

The results of the channel-path-reset function on the designated channel path are communicated to the program by means of a subsequent machine-check-interruption condition generated by the channel subsystem (see “Channel-Subsystem Recovery” on page 17-21).

### I/O-System Reset

The I/O-system-reset function is performed when the channel subsystem is powered on, when initial program loading is initiated manually (see “Initial Program Loading” on page 17-17), and when the system-reset-clear or system-reset-normal key is activated. The I/O-system-reset function cannot be initiated under program control; it must be initiated manually. I/O-system reset may fail to complete due to malfunctions detected at the channel subsystem or on a channel path. I/O-system reset is performed as part of subsystem reset, which also resets all floating interruption requests,


LzLabs GmbH. Ex. 1008-763

including pending I/O interruptions. (See "Subsystem Reset" on page 4-41.) Detailed descriptions of the effects of I/O-system reset on the various components of the I/O system appear later in this chapter.

I/O-system reset provides a means for placing the channel subsystem and its attached I/O devices in the initialized state. I/O-system reset affects only the channel-subsystem configuration in which it is performed, including all channel-subsystem components configured to that channel subsystem. I/O-system reset has no effect on any system components that are not part of the channel-subsystem configuration that is being reset. The effects of I/O-system reset on the configured components of the channel subsystem are described in the following sections.

***Channel-Subsystem State:*** I/O-system reset causes the channel subsystem to be placed in the initialized state, with all the channel-subsystem components in the states described in the following sections. All operations in progress are terminated and reset, and all indications of prior conditions are reset. These indications include status information, interruption conditions (but not pending interruptions), dedicated-allegiance conditions, pending channel reports, and all internal information regarding prior conditions and operations. In the initialized state, the channel subsystem has no activity in progress and is ready to perform the initial-program-loading (IPL) function or respond to I/O instructions, as described in Chapter 14, "I/O Instructions."

***Control Units and Devices:*** I/O-system reset causes a reset signal to be sent on all configured channel paths, including those which are not physically available (as indicated by the PAM bit being zero) because of a permanent error condition detected earlier. When the reset signal is received by a control unit, control-unit functions in progress, control-unit status, control-unit allegiance, and control-unit modes for the resetting channel path are reset. Device operations in progress, device status, device allegiance, and the device mode for the resetting channel path are also reset. Control-unit and device mode, allegiance, status, and I/O functions in progress for other channel paths are not affected.

For devices that are operating in the single-path mode, an operation can be in progress for, at most, one channel path. Therefore, if the reset signal is received on that channel path, the operation in progress is reset. Devices that have the dynamic-reconnection feature and are operating in the multipath mode, however, have the capability to establish an allegiance to a group of channel paths during an I/O operation, where all the channel paths of the path group are configured to the same channel subsystem. If an operation is in progress for a device that is operating in the multipath mode and the reset signal is received on one of the channel paths of that path group, then the operation in progress is reset for the resetting channel path only. Although the operation in progress cannot continue on the resetting channel path, it can continue on the other channel paths of the path group, subject to the following restrictions:

1. If the device is actively communicating with the channel subsystem on a channel path when it receives the reset signal on that channel path, then the operation is reset unconditionally, regardless of path groups.
2. If the operation is in progress in the multipath mode but the path group consists only of the resetting path, then the operation is reset.
3. Except as noted in item 2, if the operation in progress is currently in a disconnected state (device not actively communicating with the channel subsystem) or is active on another channel path of a path group, system reset has no effect upon the continued performance of the operation.

A control unit is completely reset after the reset signal has been received on all its channel paths, provided no new activity is initiated at the control unit between the receipt of the first and last reset signal. "Completely reset" means that the current operation, if any, at the control unit is terminated and that control-unit allegiance, control-unit status, and the control-unit mode, if any, are reset.

An I/O device is completely reset after the reset signal has been received on all channel paths of all control units by which the device is accessible, provided no new activity is initiated at the device between the receipt of the first and last reset signal. "Completely reset" means that the current operation, if any, at the device is terminated and that device allegiance, device status, and the device mode are reset.



LzLabs GmbH. Ex. 1008-764

In summary, system reset always causes an operation in progress to be reset for the channel path on which the reset signal is received. If the resetting channel path is the only channel path for which the operation is in progress, then the operation is completely reset. If a device is actively communicating on a channel path over which the reset signal is received, then the operation in progress is unconditionally and completely reset.

The reset signal is not received by control units and devices on channel paths from which the control unit has been partitioned. A control unit is partitioned from a channel path by means of an enable/disable switch on the control unit for each channel path by which it is accessible. Multitagged, unsolicited status, if any, remains pending at the control unit for such a channel path in this case. However, from the point of view of the program, the control unit and device appear to be completely reset if the reset signal is received by the control unit on all the channel paths by which it is currently accessible.

The resultant reset state of individual control units and devices is described in the System Library publication for the control unit.

***Channel Paths:*** I/O-system reset causes a reset signal to be sent on all configured channel paths and causes the channel subsystem to be placed in the reset and initialized state, as described in the previous sections. As a result of these actions, all communication between the channel subsystem and its attached control units and devices is terminated and the components reset, and all configured channel paths are made quiescent or are deconfigured.

***Subchannels:*** I/O-system reset causes all operations on all subchannels to be concluded. Status information, all interruption conditions (but not pending interruptions), dedicated-allegiance conditions, and internal indications regarding prior conditions and operations in all subchannels are reset, and all valid subchannels are placed in the initialized state.

In the initialized state, the subchannel parameters of all valid subchannels are set to their initial values. The initial values of the following subchannel parameters are zeros:

- Interruption parameter
- I/O-interruption-subclass code (ISC)
- Enabled
- Limit mode
- Measurement mode
- Multipath mode
- Path-not-operational mask
- Last-path-used mask
- Measurement-block index
- Concurrent sense

The initial values of the following subchannel parameters are assigned as part of the installation procedure for the device associated with each valid subchannel:

- Timing facility
- Device number
- Logical-path mask (same value as path-installed mask)
- Path-installed mask
- Path-available mask
- Channel-path ID 0-7

The values assigned may depend upon the particular system model and the configuration; dependencies, if any, are described in the System Library publication for the system model. Programming considerations may further constrain the values assigned.

The initial value of the path-operational mask is all ones.

The device-number-valid bit is one for all subchannels having an assigned I/O device.

The initial value of the model-dependent area of the subchannel-information block is described in the System Library publication for the system model.

The initial value of the subchannel-status word and extended-status word is all zeros.

The initialized state of the subchannel is the state specified by the initial values for the subchannel parameters described above. The description of the subchannel parameters can be found in "Subchannel-Information Block" on page 15-1, "Subchannel-Status Word" on page 16-6, and "Extended-Status Word" on page 16-32.

***Channel-Path-Reset Facility:*** I/O-system reset causes the channel-path-reset facility to be reset. A channel-path-reset function initiated by RESET CHANNEL PATH, either pending or in progress, is


LzLabs GmbH. Ex. 1008-765

overridden by I/O-system reset. The machine-check-interruption condition, which normally signals the completion of a channel-path-reset function, is not generated for a channel-path-reset function that is pending or in progress at the time I/O-system reset occurs.

***Address-Limit-Checking Facility:*** I/O-system reset causes the address-limit-checking facility to be reset. The address-limit value is initialized to all zeros and validated.

***Channel-Subsystem-Monitoring Facilities:*** I/O-system reset causes the channel-subsystem-monitoring facilities to be reset. The measurement-block-update mode and the device-connect-time-measurement mode, if active, are made inactive. The measurement-block origin and the measurement-block key are both initialized to zeros and validated.

***Pending Channel Reports:*** I/O-system reset causes pending channel reports to be reset.

***Channel-Subsystem Timer:*** I/O-system reset does not necessarily affect the contents of the channel-subsystem timer. In models that provide channel-subsystem-timer checking, I/O-system reset may cause the channel-subsystem timer to be validated.

***Pending I/O Interruptions:*** I/O-system reset does not affect pending I/O interruptions. However, during subsystem reset, I/O interruptions are cleared concurrently with the performance of I/O-system reset. (See “Subsystem Reset” on page 4-41.)



LzLabs GmbH. Ex. 1008-766

| Area Affected | Effect of I/O–System Reset[1] |
|---|---|
| Channel-subsystem state | Reset and initialized |
| Control units and devices | Reset |
| Channel paths | Quiescent |
| Subchannels | Reset and initialized |
| Interruption parameter | Zeros[2] |
| I/O-interruption–subclass code (ISC) | Zeros[2] |
| Enabled bit | Zero[2] |
| Address-limit-mode bits | Zeros[2] |
| Timing-facility bit | Installed value[2] |
| Multipath-mode bit | Zero[2] |
| Measurement-mode bits | Zeros[2] |
| Device-number-valid bit | Installed value[2] |
| Device number | Installed value[2] |
| Logical-path mask | Equal to path-installed mask value[2] |
| Path-not-operational mask | Zeros[2] |
| Last-path-used mask | Zeros[2] |
| Path-installed mask | Installed value[2] |
| Measurement-block index | Zeros[2] |
| Path-operational mask | Ones[2] |
| Path-available mask | Installed value[2] [3] |
| Channel-path ID 0-7 | Installed value[2] |
| Concurrent-sense bit | Zero[2] |
| Subchannel-status word | Zeros[2] |
| Extended-status word | Zeros[2] |
| Model-dependent area | Model dependent [2] |
| Channel-path-reset facility | Reset |
| Address-limit-checking facility | Reset and initialized |
| Address-limit value | Zeros[2] |
| Channel-subsystem-monitoring facility | Reset and initialized |
| Measurement-block-update mode | Inactive[2] |
| Device-connect-time-measurement mode | Inactive[2] |
| Measurement-block origin | Zeros[2] |
| Measurement-block key | Zeros[2] |
| Pending channel-report words | Cleared |
| Channel-subsystem timer | Unchanged/validated |

**Explanation:**

1 For a detailed description of the effect of I/O-system reset on each area, see the text.

2 Initialized value.

3 Also subject to model-dependent configuration controls, if any.

*Figure 17-1. Summary of I/O-System-Reset Actions*

## Externally Initiated Functions

I/O-system reset, which is an externally initiated function, is described in “I/O-System Reset” on page 17-13.

## Initial Program Loading

Initial program loading (IPL) provides a manual means for causing a program to be read from a designated device and for initiating execution of that program.

Some models may provide additional controls and indications relating to IPL; this additional information is specified in the System Library publication for the model.


LzLabs GmbH. Ex. 1008-767

IPL is initiated manually by setting the load-unit-address controls to a four-digit number to designate an input device and by subsequently activating the load-clear or load-normal key.

Activating the load-clear key causes a clear reset to be performed on the configuration.

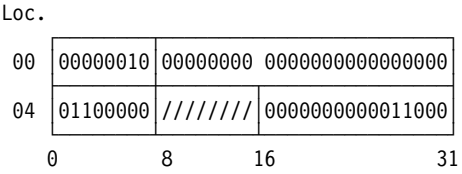
Activating the load-normal key causes an initial CPU reset to be performed on this CPU, CPU reset to be propagated to all other CPUs in the configuration, and a subsystem reset to be performed on the remainder of the configuration.

In the loading part of the operation, after the resets have been performed, this CPU enters the load state. This CPU does not necessarily enter the stopped state during performance of the reset. The load indicator is on while the CPU is in the load state.

Subsequently, if conditions allow, a read operation is initiated from the designated input device and associated subchannel. The read operation is performed as if a START SUBCHANNEL instruction were executed that designated (1) the subchannel corresponding to the device number specified by the load-unit-address controls and (2) an ORB containing all zeros, except for a byte of all ones in the logical-path-mask field. The ORB parameters are interpreted by the channel subsystem as follows:

- **Interruption parameter:** All zeros
- **Subchannel key:** All zeros
- **Suspend control:** Zero (suspension not allowed)
- **CCW format:** Zero
- **CCW prefetch:** Zero (prefetching not allowed)
- **Initial-status-interruption control:** Zero (no request)
- **Address-limit-checking control:** Zero (no checking)
- **Suppress suspended interruption:** Zero (suppression not allowed)
- **Logical-path mask:** Ones (all channel paths logically available)
- **Incorrect-length-suppression mode:** Zero (ignored because format-0 CCWs are specified)
- **Channel-program address:** Absolute address 0

The first CCW to be executed is not fetched from storage. Instead, the effect is as if an implied format-0 CCW, beginning in absolute location 0 and having the following detailed format, were executed:




In the illustration above, the CCW specifies a read command with the modifier bits zeros, a data address of 0, a byte count of 24, the chain-command flag one, the suppress-incorrect-length-indication flag one, the chain-data flag zero, the skip flag zero, the program-controlled-interruption (PCI) flag zero, the indirect-data-address (IDA) flag zero, and the suspend flag zero. The CCW fetched, as a result of command chaining, from location 8 or 16, as well as any subsequent CCW in the IPL sequence, is interpreted the same as a CCW in any I/O operation, except that any PCI flags that are specified in the IPL channel program are ignored.

At the time the subchannel is made start pending for the IPL read, it is also enabled, which ensures proper handling of subsequent status from the device by the channel subsystem and facilitates subsequent I/O operations using the IPL device. (Except for the subchannel used by the IPL I/O operation, each subchannel must first be made enabled by MODIFY SUBCHANNEL before it can accept a start function or any status from the device.)

When the IPL subchannel becomes status pending for the last operation of the IPL channel program, no I/O-interruption condition is generated. Instead, the subsystem ID is stored in absolute locations 184-187, zeros are stored in absolute locations 188-191, and the subchannel is cleared of the pending status as if TEST SUBCHANNEL had been executed but without storing information usually stored in an IRB. If the subchannel-status field that would normally have been stored is all zeros and the device-status field that would normally have been stored contains only the channel-end indication, with or without the device-end indication, the IPL I/O operation is considered to be completed successfully. If the device-end status for the IPL I/O operation is provided separately after channel-end status, it



LzLabs GmbH. Ex. 1008-768

causes an I/O-interruption condition to be generated. When the IPL I/O operation is completed successfully, a new PSW is loaded from absolute locations 0-7. If the PSW loading is successful and no malfunctions are recognized that preclude the completion of IPL, then the CPU leaves the load state, and the load indicator is turned off. If the rate control is set to the process position, the CPU enters the operating state, and CPU operation proceeds under control of the new PSW. If the rate control is set to the instruction-step position, the CPU enters the stopped state, with the manual indicator on, after the new PSW has been loaded.

If the IPL I/O operation or the PSW loading is not completed successfully, the CPU remains in the load state, and the load indicator remains on.

IPL does not complete when any of the following occurs:

- No subchannel contains a valid device number equal to the IPL device number specified by the load-unit-address controls.
- A malfunction is detected in the CPU, main storage, or channel subsystem that precludes the completion of IPL.
- Unsolicited alert status is presented by the device subsequent to the subchannel becoming start pending for the IPL read and before the IPL subchannel becomes subchannel active. The IPL read operation is not initiated in this case.
- The IPL device appeared not operational on all available channel paths to the device, or there were no available channel paths.
- The IPL device presented a status byte containing indications other than channel end, device end, status modifier, control unit end, control unit busy, device busy, or retry status during the IPL I/O operation. Whenever control-unit end, control-unit busy, or device busy is presented in the status byte, normal path-management actions are taken.
- A subchannel-status indication other than PCI was generated during the IPL I/O operation.
- The PSW loaded from absolute locations 0-7 has a PSW-format error of the type that is recognized early.

Except in the cases of no corresponding subchannel for the device number entered or a machine malfunction, the subsystem ID of the IPL device is stored in absolute locations 184-187; otherwise, the contents of these locations are unpredictable. In all cases of unsuccessful IPL, the contents of absolute locations 0-7 are unpredictable.

Subsequent to a successful IPL, the subchannel parameters contain the normal values as if an actual START SUBCHANNEL had been executed, designating the ORB as described above.

**Programming Notes:**

1. The information read and placed at absolute locations 8-15 and 16-23 may be used as CCWs for reading additional information during the IPL I/O operation: the CCW at location 8 may specify reading additional CCWs elsewhere in storage, and the CCW at location 16 may specify the transfer-in-channel command, causing transfer to these CCWs.
2. The status-modifier bit has its normal effect during the IPL I/O operation, causing the channel subsystem to fetch and chain to the CCW whose address is 16 higher than that of the current CCW. This applies also to the initial chaining that occurs after completion of the read operation specified by the implicit CCW.
3. The PSW that is loaded at the completion of the IPL operation may be provided by the first eight bytes of the IPL I/O operation or may be placed at absolute locations 0-7 by a subsequent CCW.
4. Activating the load-normal key implicitly specifies the use of the first 24 bytes of main storage and the eight bytes at absolute locations 184-191. Since the remainder of the IPL program may be placed in any part of storage, it is possible to preserve such areas of storage as may be helpful in debugging or recovery. The IPL program should not be placed in the low 512 bytes of storage since that area is reserved as described in a programming note under “Compatibility among ESA/390, ESA/370, 370-XA, and System/370” on page 1-13. When the load-clear key is activated, the IPL program starts with a cleared machine in a known state, except that


LzLabs GmbH. Ex. 1008-769

information on external storage remains unchanged.

5. When the PSW at absolute location 0 has bit 14 set to one, the CPU is placed in the wait state after the IPL operation is completed. At that point, the load and manual indicators are off, and the wait indicator is on.

## Reconfiguration of the I/O System

Reconfiguration of the I/O system is handled in a model-dependent manner. For example, changes may be made under program control, by using the model-dependent DIAGNOSE instruction; or manually, by using system-operator configuration controls; or by using a combination of DIAGNOSE and manual controls. The method used depends on the system model. The System Library publication for the system model specifies how the changes are made. The partitioning of channel paths because of reconfiguration is indicated by the setting of the PAM bits in the SCHIB stored when STORE SUBCHANNEL is executed (see "Path-Available Mask (PAM)" on page 15-7).

## Status Verification

The status-verification facility provides the channel subsystem with a means of indicating that a device has presented a device-status byte that has valid CBC but that contained a combination of bits that was inappropriate when the status byte was presented to the channel subsystem. The indication provided to the program in the ESW by the channel subsystem is called device-status check. When the channel subsystem recognizes a device-status-check condition, an interface-control-check condition is also recognized. For a summary of the status combinations considered to be appropriate or inappropriate, see the System Library publications *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202, and *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974, and the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

## Address-Limit Checking

The address-limit-checking facility provides a storage-protection mechanism for I/O data accesses to storage that augments key-controlled protection. When address-limit checking is used, absolute storage is divided into two parts by a program-controlled address-limit value. I/O data accesses can then be optionally restricted to only one of the two parts of absolute storage by the limit mode at each subchannel. The address-limit constraint applies at a higher priority than key-controlled protection, that is, I/O data accesses to the part of main storage that is protected by address-limit checking are prevented even when the subchannel key is zero or matches the key in storage. Address-limit checking does not apply to the fetching of CCWs and IDAWs.

The address-limit-checking facility consists of the following elements:

- The I/O instruction SET ADDRESS LIMIT.
- The limit mode at each subchannel.
- The address-limit-checking-control bit in the ORB.

The execution of SET ADDRESS LIMIT passes the contents of general register 1 to the address-limit-checking facility to be used as the address-limit value. Bits 0 and 16-31 of general register 1 must be zeros to designate a valid absolute address on a 64K-byte boundary; otherwise, an operand exception is recognized, and the execution of the instruction is suppressed.

The limit mode at each subchannel indicates the manner in which address-limit checking is to be performed. The limit mode is set by placing the desired value in bit positions 9 and 10 of word 1 in the SCHIB and executing MODIFY SUBCHANNEL. The settings of these bits in the SCHIB have the following meanings:

00 No limit checking (initialized value).

01 Data address must be equal to or greater than the current address limit.

10 Data address must be less than the current address limit.

11 Reserved. This combination of limit-mode bits causes an operand exception to be recognized when MODIFY SUBCHANNEL is executed.



LzLabs GmbH. Ex. 1008-770

The address-limit-checking-control bit, bit 11 of word 1 of the ORB, specifies whether address-limit checking is to be used for the start function that is accepted when the execution of START SUBCHANNEL causes the contents of the ORB to be passed to the subchannel. If the address-limit-checking-control bit is zero when the contents of the ORB are passed, address-limit checking is not specified for that start function. If the bit is one, address-limit checking is specified and is under the control of the current address limit and the current setting of the limit mode at the subchannel.

During the performance of the start function, an attempt to access an absolute storage location for data that is protected by an address limit (either high or low) is recognized as an address-limit violation, and the access is not allowed. A program-check condition is recognized, and channel-program execution is terminated, just as when an attempt is made to access an invalid address.

## Configuration Alert

The configuration-alert facility provides a detection mechanism for devices that are not associated with a subchannel in the configuration. The configuration-alert facility notifies the program, by means of a channel report, that a device which is not associated with a subchannel has attempted to communicate with the program.

Each device must be assigned to a subchannel during an installation procedure; otherwise, the channel subsystem is unable to generate an I/O-interruption condition for the device. This is because the I/O-interruption code contains the subchannel number that identifies the particular device causing the I/O-interruption condition. When a device that is not associated with a subchannel attempts to communicate with the channel subsystem, the configuration-alert facility generates a channel report in which the unassociated device is identified. For a description of the means by which the program is notified of a pending channel report and how the information in the channel report is retrieved, see "Channel Report" on page 17-22.

## Incorrect-Length-Indication Suppression

The incorrect-length-indication-suppression facility allows the indication of incorrect length for immediate operations to be suppressed in the same manner when using format-1 CCWs as when using format-0 CCWs. When the incorrect-length-indication-suppression facility is installed, bit 24 of word 1 of the ORB specifies whether the channel subsystem is to suppress the indication of incorrect length for an immediate operation when format-1 CCWs are used or whether this indication will remain under the control of the SLI flag of the current CCW (as is the case for CCWs not executed as immediate operations). This bit provides the capability for a channel program to operate in the same manner regarding the indication of incorrect length regardless of whether format-0 or format-1 CCWs are used.

## Concurrent Sense

The concurrent-sense facility provides a mechanism whereby sense information that is provided by the device can be presented by the channel subsystem to the program in the same IRB that contains the unit-check indication when the subchannel is in the concurrent-sense mode. The concurrent-sense mode is made active at a subchannel for which the concurrent-sense facility is applicable when MODIFY SUBCHANNEL is executed and bit 31 of word 6 of the SCHIB operand is set to one. The concurrent-sense facility is applicable to subchannels that are associated with channel paths by which the channel subsystem can attempt to retrieve sense information from the device without requiring program intervention.

## Channel-Subsystem Recovery

The channel subsystem provides various methods for extensive detection of malfunctions and other conditions to ensure the integrity of channel-subsystem operation and to achieve automatic recovery of some malfunctions.

The method used to report a particular malfunction or other condition is dependent upon the severity of the malfunction or other condition and the degree to which the malfunction or other condition can be isolated. A malfunction or other condition in the channel subsystem may be indicated to the


LzLabs GmbH. Ex. 1008-771

program by information being stored by one of the following methods:

1. Information is provided in the IRB describing a condition that has been recognized by either the channel subsystem or device that must be brought to the attention of the program. Generally, this information is made available to the program by the execution of TEST SUBCHANNEL, which is usually executed in response to the occurrence of an I/O interruption. (See "Interruption Action" on page 16-5, for a definition of the information stored, as well as Chapter 6, "Interruptions" on page 6-1.)
2. Information is provided in a channel report describing a machine malfunction affecting the identified facility associated with the channel-subsystem. This information is made available to the program by the execution of STORE CHANNEL REPORT WORD, which is usually executed in response to the occurrence of a machine-check interruption. (See Chapter 11, "Machine-Check Handling" on page 11-1 for a description of the machine-check-interruption mechanism and the contents of the machine-check-interruption code.)
3. Information is provided in a channel report describing a malfunction or other condition affecting a collection of channel-subsystem facilities. This information is made available to the program as indicated in item 2.
4. Information is provided in the machine-check-interruption code (MCIC) describing a malfunction affecting the continued operational integrity of the channel subsystem. (See "Channel-Subsystem Damage" on page 11-19.)
5. Information is provided in the MCIC describing a malfunction affecting the continued operational integrity of a process or of the system. (See "Instruction-Processing Damage" on page 11-17 and "System Damage" on page 11-16.)

Channel reports are used to report malfunctions or other conditions only when the use of the I/O-interruption facility is not appropriate and in preference to reporting channel-subsystem damage, instruction-processing damage, or system damage.

## Channel Report

When a malfunction or other condition affecting elements of the channel subsystem has been recognized, a channel report is generated. The performance of recovery actions by the program or by external means may be required to gain recovery from the error condition. The channel report indicates the source of the channel report and the recovery state to the extent necessary for determining the proper recovery action. A channel report consists of one or more channel-report words (CRWs) that have been generated from an analysis of the malfunction or other condition. The inclusion of two or more CRWs within a channel report is indicated by the chaining flag being stored as one in all of the CRWs of the channel report except the last one in the chain.

When a channel report is made pending by the channel subsystem for retrieval and analysis by the program (by means of the execution of STORE CHANNEL REPORT WORD), a malfunction or other condition that affects the normal operation of one or more of the channel-subsystem facilities has been recognized. If the channel report that is made pending is an initial channel report, a machine-check-interruption condition is generated that indicates one or more CRWs are pending at the channel subsystem. A channel report is initial either if it is the first channel report to be generated after the most recent I/O-system reset or if no previously generated reports are pending and the last STORE CHANNEL REPORT WORD instruction that was executed resulted in the setting of condition code 1, indicating that no channel report was pending. When the machine-check interruption occurs and bit 9 of the machine-check-interruption code (channel report pending) is one, a channel report is pending. If the program clears the first CRW of a channel report before the associated machine-check interruption has occurred, some models may reset the machine-check-interruption condition, and the associated machine-check interruption does not occur. A machine-check interruption indicating that a channel report is pending occurs only if the machine-check mask (PSW bit 13) and the channel-report-pending subclass mask, bit 3 of control register 14, are both ones.

If the channel report that is made pending is not an initial channel report, a machine-check-interruption condition is not generated. The



LzLabs GmbH. Ex. 1008-772

CRW that is presented to the program in response to the first STORE CHANNEL REPORT WORD instruction that is executed after a machine-check interruption may or may not be part of the initial channel report that caused the machine-check condition to be generated. A pending channel-report word is cleared by any CPU executing STORE CHANNEL REPORT WORD, regardless of whether a machine-check interruption has occurred in any CPU. If a CRW is not pending and STORE CHANNEL REPORT WORD is executed, condition code 1 is set, and zeros are stored at the location designated by the second-operand address. During the execution of STORE CHANNEL REPORT WORD as a result of receiving a machine-check interruption, condition code 1 may be set, and zeros may be stored because (1) the related channel report has been cleared by another CPU or (2) a malfunction occurred during the generation of a channel report. In the latter case, if, during a subsequent attempt, a valid channel report can be made pending, an additional machine-check-interruption condition is generated.

When a channel report consists of multiple chained CRWs, they are presented to the program in the same order that they are placed in the chain by the channel subsystem as a result of consecutive executions of STORE CHANNEL REPORT WORD. If, for example, the first CRW of a chain is presented to the program as a result of executing STORE CHANNEL REPORT WORD, the CRW that is presented as a result of the next execution of STORE CHANNEL REPORT WORD is the second CRW of the same chain and not a CRW that is part of another channel report.

Channel reports are not presented to the program in any special order, except for channel reports whose first or only CRW indicates the same reporting-source code and the same reporting-source ID. These channel reports are presented to the program in the same order that they are generated by the channel subsystem, but they are not necessarily presented consecutively. For example, suppose the channel subsystem generates channel reports A, B, and C, in that order. The first CRW of channel reports B and C indicates the same reporting-source code and the same reporting-source ID. Channel report B is presented to the program before channel report C is presented, but channel report A may be presented after channel report B and before channel report C.

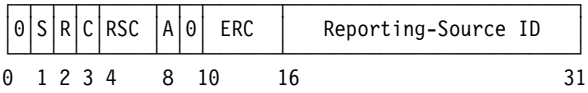
**Programming Notes:**

1. The information that is provided in a single CRW may be made obsolete by another CRW that is subsequently generated for the same channel-subsystem facility. Therefore, the information that is provided in one channel report should be interpreted in light of the information provided by all of the channel reports that are pending at a given instant.
2. A machine-check-interruption condition is not always generated when a channel report is made pending. The conditions that result in a machine-check-interruption condition being generated are described earlier in this section.
3. After a machine-check interruption has occurred with bit 9 of the machine-check-interruption code set to one, STORE CHANNEL REPORT WORD should be issued repeatedly until all of the pending channel reports have been cleared and condition code 1 has been set.
4. A CRW-overflow condition can occur if the program does not issue successive STORE CHANNEL REPORT WORD instructions in a timely manner after the machine-check interruption occurs.
5. The number of CRWs that can be pending at the same time is model dependent. During the existence of an overflow condition, CRWs that would have otherwise been made pending are lost and are never presented to the program.

## Channel-Report Word

The channel-report word (CRW) provides information to the program that can be used to facilitate the recovery of an I/O operation, a device, or some element of the channel subsystem, such as a channel path or subchannel.

The format of the CRW is as follows:





LzLabs GmbH. Ex. 1008-773

***Solicited CRW (S):*** Bit 1, when one, indicates a solicited CRW. A CRW is considered by the channel subsystem to be solicited if it is made pending as the direct result of some action that is taken by the program. When bit 1 is zero, the CRW is unsolicited and has been made pending as the result of an action taken by the channel subsystem that is independent of the program.

***Overflow (R):*** Bit 2, when one, indicates that a CRW-overflow condition has been recognized since this CRW became pending and that one or more CRWs have been lost. This bit is one in the CRW that has most recently been set pending when the overflow condition is recognized. When bit 2 is zero, a CRW-overflow condition has not been recognized.

A CRW that is part of a channel report is not made pending, even though the overflow condition does not exist, if an overflow condition prevented a previous CRW of that report from being made pending.

***Chaining (C):*** Bit 3, when one, and when the overflow flag is zero, indicates chaining of associated CRWs. Chaining of CRWs is indicated whenever a malfunction or other condition is described by more than a single CRW. The chaining flag is zero if the channel report is described by a single CRW or if the CRW is the last CRW of a channel report.

The chaining flag is not meaningful if the overflow bit, bit 2, is one.

***Reporting-Source Code (RSC):*** Bits 4-7 identify the channel-subsystem facility that is associated with the channel report. Some facilities are further identified in the reporting-source-identification field (see below). The following combinations of bits identify the facilities:

| Bits | | | | |
|---|---|---|---|---|
| **4** | **5** | **6** | **7** | **Facility** |
| 0 | 0 | 1 | 0 | Monitoring facility |
| 0 | 0 | 1 | 1 | Subchannel |
| 0 | 1 | 0 | 0 | Channel path |
| 1 | 0 | 0 | 1 | Configuration-alert facility |
| 1 | 0 | 1 | 1 | Channel subsystem |

All other bit combinations in the reporting-source-code field are reserved.

***Ancillary Report (A):*** Bit 8, when one, indicates that a malfunction of a system component has occurred that was recognized previously or which has affected the activity of multiple channel-subsystem facilities. When the malfunction affects the activity of multiple channel-subsystem facilities, an ancillary-report condition is recognized for all of the affected facilities except one. This bit, when zero, indicates that this malfunction of a system component was not recognized previously. This bit is meaningful for all channel reports.

Depending on the model, recognition of an ancillary-report condition may not be provided, or it may not be provided for all system malfunctions that affect channel-subsystem facilities. When ancillary-report recognition is not provided, bit 8 is set to zero.

***Error-Recovery Code (ERC):*** Bits 10-15, when zero, indicate that the channel subsystem has error information regarding the channel-subsystem facility identified in the reporting-source code, and that the program can now request that information. Otherwise, bit positions 10-15 contain the error-recovery code that defines the recovery state of the channel-subsystem facility identified in the reporting-source code. This field, when used in conjunction with the reporting-source code, can be used by the program to determine whether the identified facility has already been recovered and is available for use or whether recovery actions are still required. The following error-recovery codes are defined:



LzLabs GmbH. Ex. 1008-774

| Bits | | | | | | |
|---|---|---|---|---|---|---|
| **10** | **11** | **12** | **13** | **14** | **15** | **State** |
| 0 | 0 | 0 | 0 | 0 | 0 | Event-information pending |
| 0 | 0 | 0 | 0 | 0 | 1 | Available |
| 0 | 0 | 0 | 0 | 1 | 0 | Initialized |
| 0 | 0 | 0 | 0 | 1 | 1 | Temporary error |
| 0 | 0 | 0 | 1 | 0 | 0 | Installed parameters initialized |
| 0 | 0 | 0 | 1 | 0 | 1 | Terminal |
| 0 | 0 | 0 | 1 | 1 | 0 | Permanent error with facility not initialized |
| 0 | 0 | 0 | 1 | 1 | 1 | Permanent error with facility initialized |
| 0 | 0 | 1 | 0 | 0 | 0 | Installed parameters modified |

All other bit combinations in the error-recovery-code field are reserved.

The specific meaning of each error-recovery code depends on the particular reporting-source code that accompanies it in a CRW. The error-recovery codes are defined as follows:

*Event-Information Pending:* Event information for the identified facility is available for retrieval by the program. This CRW does not indicate the state of the identified facility.

*Available:* The identified facility is in the same state that the program would expect if the CRW had not been generated.

*Initialized:* The identified facility is in the same state that existed immediately following the I/O-system reset that was part of the most recent system IPL.

*Temporary Error:* The identified facility is not operating in a normal manner or has recognized the occurrence of an abnormal event. It is expected that subsequent actions either will restore the facility to normal operation or will record the appropriate information describing the abnormal event.

*Installed Parameters Initialized:* This state is the same as the initialized state, except that one or more parameters that are associated with the facility and that are not modifiable by the program may have been changed.

*Terminal:* The identified facility is in a state such that an operation that was in progress can neither be completed nor terminated in the normal manner.

*Permanent Error with Facility Not Initialized:* The identified facility is in a state of malfunction, and the channel subsystem has not caused a reset function to be performed for that facility.

*Permanent Error with Facility Initialized:* The identified facility is in a state of malfunction, and the channel subsystem has caused or may have caused a reset function to be performed for that facility.

*Installed Parameters Modified:* One or more parameters of the specified facility have been changed.

***Reporting-Source ID (RSID):*** Bit positions 16-31 contain the reporting-source ID, which may, depending upon the condition that caused the channel report and the reporting-source code, either further identify the affected channel-subsystem facility or provide additional information describing the condition that caused the channel report. The RSID field has the following format as a function of the bit settings of the reporting-source code.

| Reporting-Source Code | | | | Reporting-Source ID Bits 16-31 | | | |
|---|---|---|---|---|---|---|---|
| **4** | **5** | **6** | **7** | | | | |
| 0 | 0 | 1 | 0 | 0000 | 0000 | 0000 | 0000 |
| 0 | 0 | 1 | 1 | xxxx | xxxx | xxxx | xxxx |
| 0 | 1 | 0 | 0 | 0000 | 0000 | yyyy | yyyy |
| 1 | 0 | 0 | 1 | 0000 | 0000 | yyyy | yyyy |
| 1 | 0 | 1 | 1 | 0000 | 0000 | 0000 | 0000 |

**Note:**

| | |
|---|---|
| **xxxx xxxx xxxx xxxx** | Subchannel number |
| **yyyy yyyy** | Channel-path ID (CHPID) |

## Channel-Subsystem-I/O-Priority Facility

The channel-subsystem-I/O-priority facility, when installed, provides a means by which the program can establish a priority relationship, at the channel subsystem, among the subchannels that are placed into the start-pending state when START SUBCHANNEL is executed and condition code 0


LzLabs GmbH. Ex. 1008-775

is indicated. For I/O-subchannels that are configured to fibre-channel channel paths (FICON and FICON-converted channel paths), it also provides a means by which the program can establish a priority relationship for I/O operations at the fibre-channel-attached control units.

The program assigns the desired channel-subsystem priority and control-unit priority by specifying the desired priority numbers in the ORB extension when START SUBCHANNEL is executed.

The channel-subsystem-priority number specified in the ORB is used by the channel subsystem to determine the order in which start-pending and resume-pending subchannels are selected when the channel subsystem attempts to initiate a start function or a resume function. See the section "Start Function and Resume Function" on page 15-18 for details about these functions. In general, I/O subchannels that are in the start-pending or resume-pending state and have a higher priority number are selected for start-function or resume-function initiation by the channel subsystem before start-pending or resume-pending subchannels that have a lower priority number. The specific priority selection algorithm used by the channel subsystem for this purpose depends on the model. Additionally, the channel subsystem also applies a fairness selection algorithm in conjunction with the priority selection algorithm when selecting I/O subchannels. The specific fairness selection algorithm also depends on the model. For all models, the channel-subsystem priority and fairness selection algorithms are always applied to I/O subchannels that are either start pending or resume pending. Some models may also apply both algorithms to subchannels that are either clear pending or halt pending. See a model's System-Library publication for a description of the priority and fairness selection algorithms that the model provides and whether these algorithms are also applied to clear-pending or halt-pending subchannels.

When the channel-subsystem-I/O-priority facility and the FICON-channel facility are installed, the control-unit-priority number specified in the ORB is used by control units attached to fibre-channel channel paths in order to determine the priority of the execution of CCWs at the control unit. See "Control-Unit (CU) Priority:" on page 15-26 for additional information.

## Number of Channel-Subsystem-Priority Levels

Depending on the model, fewer than 256 channel-subsystem-priority levels may be provided by the channel subsystem. Each priority level that the model provides is designated by an eight-bit unsigned binary integer. The lowest provided channel-subsystem-priority level is designated by the integer 0, and each succeeding higher priority level is designated by the next-higher sequential integer. For example, if the model provides 16 priority levels, they are numbered 0-15, respectively, from the lowest priority level to the highest priority level.



LzLabs GmbH. Ex. 1008-776

# Chapter 18. Hexadecimal-Floating-Point Instructions


HFP Arithmetic . . . . . 18-1
HFP Number Representation . . . . . 18-1
Normalization . . . . . 18-3
HFP Data Format . . . . . 18-3
Instructions . . . . . 18-4
ADD NORMALIZED . . . . . 18-8
ADD UNNORMALIZED . . . . . 18-10
COMPARE . . . . . 18-10
CONVERT FROM FIXED . . . . . 18-11
CONVERT TO FIXED . . . . . 18-11
DIVIDE . . . . . 18-12
HALVE . . . . . 18-13
LOAD AND TEST . . . . . 18-14
LOAD COMPLEMENT . . . . . 18-15
LOAD FP INTEGER . . . . . 18-15
LOAD LENGTHENED . . . . . 18-16
LOAD NEGATIVE . . . . . 18-16
LOAD POSITIVE . . . . . 18-17
LOAD ROUNDED . . . . . 18-18
MULTIPLY . . . . . 18-18
MULTIPLY AND ADD . . . . . 18-20
MULTIPLY AND SUBTRACT . . . . . 18-20
SQUARE ROOT . . . . . 18-21
SUBTRACT NORMALIZED . . . . . 18-23
SUBTRACT UNNORMALIZED . . . . . 18-23


## HFP Arithmetic

### HFP Number Representation

A hexadecimal-floating-point (HFP) number consists of a sign bit, a hexadecimal fraction, and an unsigned seven-bit binary integer called the characteristic. The characteristic represents a signed exponent and is obtained by adding 64 to the exponent value (excess-64 notation). The range of the characteristic is 0 to 127, which corresponds to an exponent range of −64 to +63. The magnitude of an HFP number is the product of its fraction and the number 16 raised to the power of the exponent that is represented by its characteristic. The number is positive or negative depending on whether the sign bit is zero or one, respectively.

The fraction of an HFP number is treated as a hexadecimal number because it is considered to be multiplied by a number which is a power of 16. The name, fraction, indicates that the radix point is assumed to be immediately to the left of the leftmost fraction digit.

When an HFP operation would cause the result exponent to exceed 63, the characteristic wraps around from 127 to 0, and an HFP-exponent-overflow condition exists. The result characteristic is then too small by 128. When an operation would cause the exponent to be less than −64, the characteristic wraps around from 0 to 127, and an HFP-exponent-underflow condition exists. The result characteristic is then too large by 128, except that a zero characteristic is produced when a true zero is forced.

A true zero is an HFP number with a zero characteristic and zero fraction. A true zero may arise as the normal result of an arithmetic operation because of the particular magnitude of the operands. For HFP operations, the result is forced to be a positive true zero when:

1. An HFP exponent underflow occurs and the HFP-exponent-underflow mask bit in the PSW is zero.
2. The result fraction of an addition or subtraction operation is zero and the HFP-significance mask bit in the PSW is zero.
3. The operand of the CONVERT FROM FIXED instruction is zero.
4. The dividend in the DIVIDE instruction has a zero fraction.
5. The operand of the HALVE, LOAD FP INTEGER, or SQUARE ROOT instruction has a zero fraction.
6. One or both operands of a multiplication operation has a zero fraction.

Item 2, above, applies to normalized and unnormalized instructions.

When a program interruption for HFP exponent underflow occurs, a true zero is not forced; instead, the fraction and sign remain correct, and the characteristic is too large by 128. When a

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-777

| Instruction | Nonzero Result Normalized | Zero Result Forced to True Zero | | Zero Result Made Positive | |
|---|---|---|---|---|---|
| | | Short and Long | Extended | Short and Long | Extended |
| ADD NORMALIZED | Yes | Y/N | Y/N | Yes | Yes |
| ADD UNNORMALIZED | No | Y/N | - | Yes | - |
| CONVERT BFP TO HFP[1] | Yes | Yes | - | No | - |
| CONVERT FROM FIXED | Yes | Yes | Yes | Yes | Yes |
| DIVIDE | Yes | Yes | Yes | Yes | Yes |
| HALVE | Yes | Yes | - | Yes | - |
| LOAD[1] | No | No | No | No | No |
| LOAD AND TEST | No | No | Yes | No | No |
| LOAD COMPLEMENT | No | No | Yes | No | No |
| LOAD FP INTEGER | Yes | Yes | Yes | Yes | Yes |
| LOAD LENGTHENED | No | No | Yes | No | No |
| LOAD NEGATIVE | No | No | Yes | No | No |
| LOAD POSITIVE | No | No | Yes | Yes | Yes |
| LOAD ROUNDED | No | No | - | No | - |
| LOAD ZERO[1] | - | Yes | Yes | Yes | Yes |
| MULTIPLY | Yes | Yes | Yes | Yes | Yes |
| SQUARE ROOT | Yes | Yes | Yes | Yes | Yes |
| STORE[1] | No | No | - | No | - |
| SUBTRACT NORMALIZED | Yes | Y/N | Y/N | Yes | Yes |
| SUBTRACT UNNORMALIZED | No | Y/N | - | Yes | - |

**Explanation:**

- \- Not applicable.
- [1] Floating-point-support instruction.
- Y/N When the HFP-significance mask bit (PSW bit 23) is zero, a true zero is forced. When the HFP-significance mask bit is one, the characteristic remains unchanged, and a program interruption for HFP significance occurs.

*Figure 18-1. Normalization and Zero Handling for Instructions with HFP Results*

program interruption for HFP significance occurs, the fraction remains zero, the sign is positive, and the characteristic remains correct.

The sign of a sum, difference, product, quotient, square root, the result of CONVERT FROM FIXED, or the result of LOAD FP INTEGER with a zero fraction is positive. The sign for a zero frac-



LzLabs GmbH. Ex. 1008-778

tion resulting from other HFP operations is established from the operand sign, the same as for nonzero fractions.

## Normalization

A quantity can be represented with the greatest precision by an HFP number of a given fraction length when that number is normalized. A normalized HFP number has a nonzero leftmost hexadecimal fraction digit. If one or more leftmost fraction digits are zeros, the number is said to be unnormalized.

Unnormalized numbers are normalized by shifting the fraction left, one digit at a time, until the leftmost hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A number with a zero fraction cannot be normalized; either its characteristic remains unchanged or its characteristic is made zero when the result is forced to be a true zero.

Addition and subtraction with extended operands, as well as the MULTIPLY, DIVIDE, CONVERT FROM FIXED, HALVE, LOAD FP INTEGER, and SQUARE ROOT operations, are performed only with normalization. Addition and subtraction with short or long operands may be specified as either normalized or unnormalized. For all other operations, the result is produced without normalization.

With unnormalized operations, leftmost zeros in the result fraction are not eliminated. The result may or may not be in normalized form, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. The operands for multiply, divide, and square-root operations are normalized before the arithmetic process. For other normalized operations, normalization takes place when the intermediate arithmetic result is changed to the final result.

When the intermediate result of addition, subtraction, or rounding causes the fraction to overflow, the fraction is shifted right by one hexadecimal-digit position, and the value one is supplied to the vacated leftmost digit position. The fraction is then truncated to the final result length, while the characteristic is increased by one. This adjustment is made for both normalized and unnormalized operations.

Figure 18-1 on page 18-2 summarizes, for all instructions producing HFP results, the handling of zero results and whether normalization occurs for nonzero results.

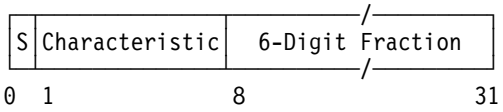
**Programming Note:** Up to three leftmost bits of the fraction of a normalized number may be zeros, since the nonzero test applies to the entire leftmost hexadecimal digit.

## HFP Data Format

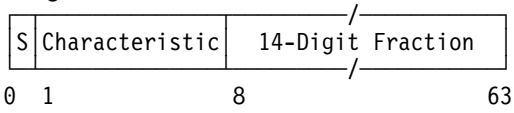
HFP numbers have a 32-bit (short) format, a 64-bit (long) format, or a 128-bit (extended) format. Numbers in the short and long formats may be designated as operands both in storage and in the floating-point registers, whereas operands having the extended format can be designated only in the floating-point registers.

In all formats, the first bit (bit 0) is the sign bit (S). The next seven bits are the characteristic. In the short and long formats, the remaining bits constitute the fraction, which consists of six or 14 hexadecimal digits, respectively.

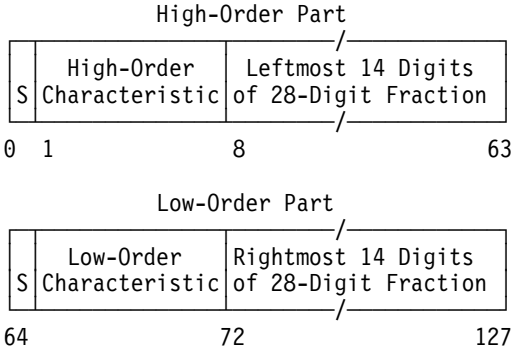
Short HFP Number




Long HFP Number




Extended HFP Number






LzLabs GmbH. Ex. 1008-779

An extended HFP number has a 28-digit fraction and consists of two long HFP numbers that are called the high-order and low-order parts. The high-order part may be any long HFP number. The fraction of the high-order part contains the leftmost 14 hexadecimal digits of the 28-digit fraction. The characteristic and sign of the high-order part are the characteristic and sign of the extended HFP number. If the high-order part is normalized, the extended number is considered normalized. The fraction of the low-order part contains the rightmost 14 digits of the 28-digit fraction. The sign and characteristic of the low-order part of an extended operand are ignored.

When a result is generated in the extended format and placed in a register pair, the sign of the low-order part is made the same as that of the high-order part, and, unless the result is a true zero, the low-order characteristic is made 14 less than the high-order characteristic. When the subtraction of 14 would cause the low-order characteristic to become less than zero, the characteristic is made 128 greater than its correct value. (Thus, the subtraction is performed modulo 128.) HFP exponent underflow is indicated only when the high-order characteristic underflows.

When an extended result is made a true zero, both the high-order and low-order parts are made a true zero.

The range covered by the magnitude (M) of a normalized HFP number depends on the format.

In the short format:

$$16^{-65} \le M \le (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \le M \le (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \le M \le (1 - 16^{-28}) \times 16^{63}$$

In all formats, approximately:

$$5.4 \times 10^{-79} \le M \le 7.2 \times 10^{75}$$

Although the final result of an HFP operation has six hexadecimal fraction digits in the short format, 14 fraction digits in the long format, and 28 fraction digits in the extended format, intermediate results have one additional hexadecimal digit on the right. This digit is called the guard digit. The guard digit may increase the precision of the final result because it participates in addition, subtraction, and comparison operations and in the left shift that occurs during normalization.

The entire set of HFP operations with normalized results is available for short, long, and extended operands in register-register versions; and for short and long operands in register-storage versions. Most instructions generate a result that has the same format as the source operands, except that there are multiplication operations which can generate a long product from short operands or an extended product from long operands. Other exceptions are instructions which convert operands from one floating-point format to another or between floating-point and fixed-point (binary-integer) formats.

**Programming Notes:**

1. In the absence of an HFP exponent overflow or HFP exponent underflow, the long HFP number constituting the low-order part of an extended result correctly expresses the value of the low-order part of the extended result when the characteristic of the high-order part is 14 or higher. This applies also when the result is a true zero. When the high-order characteristic is less than 14 but the number is not a true zero, the low-order part, when considered as a long HFP number, does not express the correct characteristic value.
2. The entire fraction of an extended result participates in normalization. The low-order part alone may or may not appear to be a normalized long HFP number, depending on whether the 15th digit of the normalized 28-digit fraction is nonzero or zero.

## Instructions

The HFP instructions and their mnemonics and operation codes are listed in the figure “Summary of HFP Instructions.” The figure indicates, in the column labeled “Characteristics,” the instruction format, when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

All HFP instructions are subject to the AFP-register-control bit, bit 13 of control register 0. The AFP-register-control bit must be one when an AFP register is specified as an operand location;



LzLabs GmbH. Ex. 1008-780

otherwise, an AFP-register data exception, DXC 1, is recognized. An operation exception is recognized when the CPU attempts to execute an instruction which is part of the HFP-extensions facility or square-root facility when the facility is not installed.

Mnemonics for the HFP instructions have an R as the last letter when the instruction is in the RR, RRE, or RRF format. Certain letters are used for HFP instructions to represent operand-format length and normalization, as follows:

F Thirty-two-bit fixed point
D Long normalized
E Short normalized
U Short unnormalized
W Long unnormalized
X Extended normalized

**Note:** In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For a register-to-register operation using COMPARE (short), for example, CER is the mnemonic and $R_1$,$R_2$ the operand designation.

**Programming Notes:**

1. The HFP instruction SQUARE ROOT (SQDR, SQER) is available when the square-root facility is installed.
2. The following additional HFP instructions are available when the HFP-extensions facility is installed:
   - COMPARE (CXR)
   - CONVERT FROM FIXED
   - CONVERT TO FIXED
   - LOAD AND TEST (LTXR)
   - LOAD COMPLEMENT (LCXR)
   - LOAD FP INTEGER
   - LOAD LENGTHENED
   - LOAD NEGATIVE (LNXR)
   - LOAD POSITIVE (LPXR)
   - LOAD ROUNDED (LEXR)
   - MULTIPLY (MEER, MEE)
   - SQUARE ROOT (SQXR, SQD, SQE)
3. The following additional HFP instructions are available when the HFP-multiply-and-add/subtract facility is installed.
   - MULTIPLY AND ADD (MAD, MADR, MAE, MAER)
   - MULTIPLY AND SUBTRACT (MSD, MSDR, MSE, MSER)


LzLabs GmbH. Ex. 1008-781

| Name | Mne-monic | Characteristics | | | | | | Op Code |
|---|---|---|---|---|---|---|---|---|
| ADD NORMALIZED (extended HFP) | AXR | RR C | | SP | Da EU EO | LS | | 36 |
| ADD NORMALIZED (long HFP) | ADR | RR C | | SP | Da EU EO | LS | | 2A |
| ADD NORMALIZED (long HFP) | AD | RX C | A | SP | Da EU EO | LS | $B_2$ | 6A |
| ADD NORMALIZED (short HFP) | AER | RR C | | SP | Da EU EO | LS | | 3A |
| ADD NORMALIZED (short HFP) | AE | RX C | A | SP | Da EU EO | LS | $B_2$ | 7A |
| ADD UNNORMALIZED (long HFP) | AWR | RR C | | SP | Da EO | LS | | 2E |
| ADD UNNORMALIZED (long HFP) | AW | RX C | A | SP | Da EO | LS | $B_2$ | 6E |
| ADD UNNORMALIZED (short HFP) | AUR | RR C | | SP | Da EO | LS | | 3E |
| ADD UNNORMALIZED (short HFP) | AU | RX C | A | SP | Da EO | LS | $B_2$ | 7E |
| COMPARE (extended HFP) | CXR | RRE C HX | | SP | Da | | | B369 |
| COMPARE (long HFP) | CDR | RR C | | SP | Da | | | 29 |
| COMPARE (long HFP) | CD | RX C | A | SP | Da | | $B_2$ | 69 |
| COMPARE (short HFP) | CER | RR C | | SP | Da | | | 39 |
| COMPARE (short HFP) | CE | RX C | A | SP | Da | | $B_2$ | 79 |
| CONVERT FROM FIXED (32 to ext. HFP) | CXFR | RRE HX | | SP | Da | | | B3B6 |
| CONVERT FROM FIXED (32 to long HFP) | CDFR | RRE HX | | | Da | | | B3B5 |
| CONVERT FROM FIXED (32 to short HFP) | CEFR | RRE HX | | | Da | | | B3B4 |
| CONVERT TO FIXED (ext. HFP to 32) | CFXR | RRF C HX | | SP | Da | R | | B3BA |
| CONVERT TO FIXED (long HFP to 32) | CFDR | RRF C HX | | SP | Da | R | | B3B9 |
| CONVERT TO FIXED (short HFP to 32) | CFER | RRF C HX | | SP | Da | R | | B3B8 |
| DIVIDE (extended HFP) | DXR | RRE | | SP | Da EU EO FK | | | B22D |
| DIVIDE (long HFP) | DDR | RR | | SP | Da EU EO FK | | | 2D |
| DIVIDE (long HFP) | DD | RX | A | SP | Da EU EO FK | | $B_2$ | 6D |
| DIVIDE (short HFP) | DER | RR | | SP | Da EU EO FK | | | 3D |
| DIVIDE (short HFP) | DE | RX | A | SP | Da EU EO FK | | $B_2$ | 7D |
| HALVE (long HFP) | HDR | RR | | SP | Da EU | | | 24 |
| HALVE (short HFP) | HER | RR | | SP | Da EU | | | 34 |
| LOAD AND TEST (extended HFP) | LTXR | RRE C HX | | SP | Da | | | B362 |
| LOAD AND TEST (long HFP) | LTDR | RR C | | SP | Da | | | 22 |
| LOAD AND TEST (short HFP) | LTER | RR C | | SP | Da | | | 32 |
| LOAD COMPLEMENT (extended HFP) | LCXR | RRE C HX | | SP | Da | | | B363 |
| LOAD COMPLEMENT (long HFP) | LCDR | RR C | | SP | Da | | | 23 |
| LOAD COMPLEMENT (short HFP) | LCER | RR C | | SP | Da | | | 33 |
| LOAD FP INTEGER (extended HFP) | FIXR | RRE HX | | SP | Da | | | B367 |
| LOAD FP INTEGER (long HFP) | FIDR | RRE HX | | | Da | | | B37F |
| LOAD FP INTEGER (short HFP) | FIER | RRE HX | | | Da | | | B377 |
| LOAD LENGTHENED (long to ext. HFP) | LXDR | RRE HX | | SP | Da | | | B325 |
| LOAD LENGTHENED (long to ext. HFP) | LXD | RXE HX | A | SP | Da | | $B_2$ | ED25 |
| LOAD LENGTHENED (short to ext. HFP) | LXER | RRE HX | | SP | Da | | | B326 |
| LOAD LENGTHENED (short to ext. HFP) | LXE | RXE HX | A | SP | Da | | $B_2$ | ED26 |

*Figure 18-2 (Part 1 of 3). Summary of HFP Instructions*



LzLabs GmbH. Ex. 1008-782

| Name | Mne-monic | Characteristics | | | | | | Op Code |
|---|---|---|---|---|---|---|---|---|
| LOAD LENGTHENED (short to long HFP) | LDER | RRE | HX | | Da | | | B324 |
| LOAD LENGTHENED (short to long HFP) | LDE | RXE | HX | A | Da | | $B_2$ | ED24 |
| LOAD NEGATIVE (extended HFP) | LNXR | RRE C | HX | SP | Da | | | B361 |
| LOAD NEGATIVE (long HFP) | LNDR | RR C | | SP | Da | | | 21 |
| LOAD NEGATIVE (short HFP) | LNER | RR C | | SP | Da | | | 31 |
| LOAD POSITIVE (extended HFP) | LPXR | RRE C | HX | SP | Da | | | B360 |
| LOAD POSITIVE (long HFP) | LPDR | RR C | | SP | Da | | | 20 |
| LOAD POSITIVE (short HFP) | LPER | RR C | | SP | Da | | | 30 |
| LOAD ROUNDED (extended to long HFP) | LDXR | RR | | SP | Da EO | | | 25 |
| LOAD ROUNDED (extended to long HFP) | LRDR | RR | | SP | Da EO | | | 25 |
| LOAD ROUNDED (extended to short HFP) | LEXR | RRE | HX | SP | Da EO | | | B366 |
| LOAD ROUNDED (long to short HFP) | LEDR | RR | | SP | Da EO | | | 35 |
| LOAD ROUNDED (long to short HFP) | LRER | RR | | SP | Da EO | | | 35 |
| MULTIPLY (extended HFP) | MXR | RR | | SP | Da EU EO | | | 26 |
| MULTIPLY (long HFP) | MDR | RR | | SP | Da EU EO | | | 2C |
| MULTIPLY (long HFP) | MD | RX | | A SP | Da EU EO | | $B_2$ | 6C |
| MULTIPLY (long to extended HFP) | MXDR | RR | | SP | Da EU EO | | | 27 |
| MULTIPLY (long to extended HFP) | MXD | RX | | A SP | Da EU EO | | $B_2$ | 67 |
| MULTIPLY (short HFP) | MEER | RRE | HX | | Da EU EO | | | B337 |
| MULTIPLY (short HFP) | MEE | RXE | HX | A | Da EU EO | | $B_2$ | ED37 |
| MULTIPLY (short to long HFP) | MDER | RR | | SP | Da EU EO | | | 3C |
| MULTIPLY (short to long HFP) | MER | RR | | SP | Da EU EO | | | 3C |
| MULTIPLY (short to long HFP) | MDE | RX | | A SP | Da EU EO | | $B_2$ | 7C |
| MULTIPLY (short to long HFP) | ME | RX | | A SP | Da EU EO | | $B_2$ | 7C |
| MULTIPLY AND ADD (long HFP) | MADR | RRF | HM | | Da EU EO | | | B33E |
| MULTIPLY AND ADD (long HFP) | MAD | RXF | HM | A | Da EU EO | | $B_2$ | ED3E |
| MULTIPLY AND ADD (short HFP) | MAER | RRF | HM | | Da EU EO | | | B32E |
| MULTIPLY AND ADD (short HFP) | MAE | RXF | HM | A | Da EU EO | | $B_2$ | ED2E |
| MULTIPLY AND SUBTRACT (long HFP) | MSDR | RRF | HM | | Da EU EO | | | B33F |
| MULTIPLY AND SUBTRACT (long HFP) | MSD | RXF | HM | A | Da EU EO | | $B_2$ | ED3F |
| MULTIPLY AND SUBTRACT (short HFP) | MSER | RRF | HM | | Da EU EO | | | B32F |
| MULTIPLY AND SUBTRACT (short HFP) | MSE | RXF | HM | A | Da EU EO | | $B_2$ | ED2F |
| SQUARE ROOT (extended HFP) | SQXR | RRE | HX | SP | Da SQ | | | B336 |
| SQUARE ROOT (long HFP) | SQDR | RRE | QR | SP | Da SQ | | | B244 |
| SQUARE ROOT (long HFP) | SQD | RXE | HX | A | Da SQ | | $B_2$ | ED35 |
| SQUARE ROOT (short HFP) | SQER | RRE | QR | SP | Da SQ | | | B245 |
| SQUARE ROOT (short HFP) | SQE | RXE | HX | A | Da SQ | | $B_2$ | ED34 |
| SUBTRACT NORMALIZED (extended HFP) | SXR | RR C | | SP | Da EU EO | LS | | 37 |
| SUBTRACT NORMALIZED (long HFP) | SDR | RR C | | SP | Da EU EO | LS | | 2B |
| SUBTRACT NORMALIZED (long HFP) | SD | RX C | | A SP | Da EU EO | LS | $B_2$ | 6B |

*Figure 18-2 (Part 2 of 3). Summary of HFP Instructions*


LzLabs GmbH. Ex. 1008-783

| Name | Mne-monic | Characteristics | | | | | | | Op Code |
|---|---|---|---|---|---|---|---|---|---|
| SUBTRACT NORMALIZED (short HFP) | SER | RR | C | | SP | Da EU EO | LS | | | | 3B |
| SUBTRACT NORMALIZED (short HFP) | SE | RX | C | A | SP | Da EU EO | LS | | | $B_2$ | 7B |
| SUBTRACT UNNORMALIZED (long HFP) | SWR | RR | C | | SP | Da EO | LS | | | | 2F |
| SUBTRACT UNNORMALIZED (long HFP) | SW | RX | C | A | SP | Da EO | LS | | | $B_2$ | 6F |
| SUBTRACT UNNORMALIZED (short HFP) | SUR | RR | C | | SP | Da EO | LS | | | | 3F |
| SUBTRACT UNNORMALIZED (short HFP) | SU | RX | C | A | SP | Da EO | LS | | | $B_2$ | 7F |

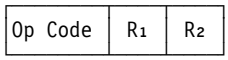
**Explanation:**

A Access exceptions for logical addresses.
$B_2$ $B_2$ field designates an access register in the access-register mode.
C Condition code is set.
Da AFP-register data exception.
EO HFP-exponent-overflow exception.
EU HFP-exponent-underflow exception.
FK HFP-divide exception.
HM HFP-multiply-and add/subtract facility.
HX HFP-extensions facility.
LS HFP-significance exception.
QR Square-root facility.
R PER general-register-alteration event.
RR RR instruction format.
RRE RRE instruction format.
RRF RRF instruction format.
RX RX instruction format.
RXE RXE instruction format.
SP Specification exception.
SQ HFP-square-root exception.

*Figure 18-2 (Part 3 of 3). Summary of HFP Instructions*

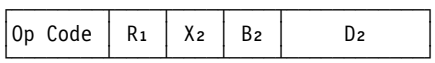
## ADD NORMALIZED

Mnemonic1 $R_1$,$R_2$ [RR]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| AER | '3A' | Short HFP |
| ADR | '2A' | Long HFP |
| AXR | '36' | Extended HFP |

Mnemonic2 $R_1$,$D_2$($X_2$,$B_2$) [RX]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| AE | '7A' | Short HFP |
| AD | '6A' | Long HFP |

The second operand is added to the first operand, and the normalized sum is placed at the first-operand location.

Addition of two HFP numbers consists in characteristic comparison, fraction alignment, and signed fraction addition. The characteristics of the two operands are compared, and the fraction accompanying the smaller characteristic is aligned with the other fraction by a right shift, with its characteristic increased by one for each hexadecimal digit of shift until the two characteristics agree.

When a fraction is shifted right during alignment, the leftmost hexadecimal digit shifted out is retained as a guard digit. The fraction that is not shifted is considered to be extended with a zero in the guard-digit position. When no alignment shift occurs, both operands are considered to be extended with zeros in the guard-digit position. The fractions with signs are then added algebraically to form a signed intermediate sum.

The intermediate-sum fraction consists of seven (short format), 15 (long format), or 29 (extended format) hexadecimal digits, including the guard



LzLabs GmbH. Ex. 1008-784

digit, and a possible carry. If a carry is present, the sum is shifted right one digit position so that the carry becomes the leftmost digit of the fraction, and the characteristic is increased by one.

If the addition produces no carry, the intermediate-sum fraction is shifted left as necessary to eliminate any leading hexadecimal zero digits resulting from the addition, provided the fraction is not zero. Zeros are supplied to the vacated rightmost digits, and the characteristic is reduced by the number of hexadecimal digits of shift. The fraction thus normalized is then truncated on the right to six (short format), 14 (long format), or 28 (extended format) hexadecimal digits. In the extended format, a characteristic is generated for the low-order part, which is 14 less than the high-order characteristic.

The sign of the sum is determined by the rules of algebra, unless all digits of the intermediate-sum fraction are zero, in which case the result is made a positive true zero.

An HFP-exponent-overflow exception exists when a carry from the leftmost position of the intermediate-sum fraction would cause the characteristic of the normalized sum to exceed 127. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct. For extended results, the characteristic of the low-order part remains correct.

An HFP-exponent-underflow exception exists when the characteristic of the normalized sum would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero. For extended results, HFP exponent underflow is not recognized when the low-order characteristic is less than zero but the high-order characteristic is equal to or greater than zero.

The result fraction is zero when the intermediate-sum fraction, including the guard digit, is zero. With a zero result fraction, the action depends on the setting of the HFP-significance mask bit in the PSW. If the HFP-significance mask bit in the PSW is one, no normalization occurs, the intermediate and final result characteristics are the same, and a program interruption for HFP significance occurs. If the HFP-significance mask bit in the PSW is zero, the program interruption does not occur; instead, the result is made a positive true zero.

For AXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --

***Program Exceptions:***

- Access (fetch, operand 2 of AE and AD only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP exponent underflow
- HFP significance
- Specification

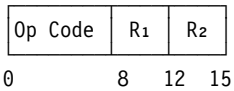
**Programming Notes:**

1. An example of the use of the ADD NORMALIZED instruction (AE) is given in Appendix A.
2. Interchanging the two operands in an HFP addition does not affect the value of the sum.
3. The ADD NORMALIZED instruction normalizes the sum but not the operands. Thus, if one or both operands are unnormalized, precision may be lost during fraction alignment.


LzLabs GmbH. Ex. 1008-785

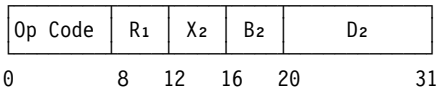
## ADD UNNORMALIZED

```
Mnemonic1    R1,R2                [RR]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| AUR | '3E' | Short HFP |
| AWR | '2E' | Long HFP |

```
Mnemonic2    R1,D2(X2,B2)         [RX]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| AU | '7E' | Short HFP |
| AW | '6E' | Long HFP |

The second operand is added to the first operand, and the unnormalized sum is placed at the first-operand location.

The execution of ADD UNNORMALIZED is identical to that of ADD NORMALIZED, except that:

1. When no carry is present after the addition, the intermediate-sum fraction is truncated to the proper result-fraction length without a left shift to eliminate leading hexadecimal zeros and without the corresponding reduction of the characteristic.
2. HFP exponent underflow cannot occur.
3. The guard digit does not participate in the recognition of a zero result fraction. A zero result fraction is recognized when the fraction (that is, the intermediate-sum fraction, excluding the guard digit) is zero.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result fraction zero
1 Result less than zero
2 Result greater than zero
3 --

***Program Exceptions:***

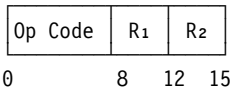
- Access (fetch, operand 2 of AU and AW only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP significance
- Specification

**Programming Notes:**

1. An example of the use of the ADD UNNORMALIZED instruction (AU) is given in Appendix A.
2. Except when the result is made a true zero, the characteristic of the result of ADD UNNORMALIZED is equal to the greater of the two operand characteristics, increased by one if the fraction addition produced a carry, or set to zero if HFP exponent overflow occurred.

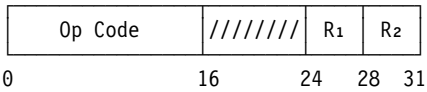
## COMPARE

```
Mnemonic1    R1,R2                [RR]
```



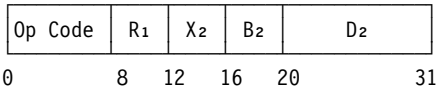

| Mnemonic1 | Op Code | Operands |
|---|---|---|
| CER | '39' | Short HFP |
| CDR | '29' | Long HFP |

```
Mnemonic2    R1,R2                [RRE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| CXR | 'B369' | Extended HFP |

```
Mnemonic3    R1,D2(X2,B2)         [RX]
```




| Mnemonic3 | Op Code | Operands |
|---|---|---|
| CE | '79' | Short HFP |
| CD | '69' | Long HFP |

The first operand is compared with the second operand, and the condition code is set to indicate the result.

The comparison is algebraic and follows the procedure for normalized subtraction, except that the



LzLabs GmbH. Ex. 1008-786

difference is discarded after setting the condition code and both operands remain unchanged. When the difference, including the guard digit, is zero, the operands are equal. When a nonzero difference is positive or negative, the first operand is high or low, respectively.

An HFP-exponent-overflow, HFP-exponent-underflow, or HFP-significance exception cannot occur.

For CXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Operands equal
1 First operand low
2 First operand high
3 --

***Program Exceptions:***

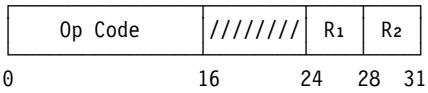
- Access (fetch, operand 2 of CE and CD only)
- Data with DXC 1, AFP register
- Operation (for CXR, if the HFP-extensions facility is not installed)
- Specification

**Programming Notes:**

1. Examples of the use of the COMPARE instruction (CDR) are given in Appendix A.
2. An exponent inequality alone is not sufficient to determine the inequality of two operands with the same sign, because the fractions may have different numbers of leading hexadecimal zeros.
3. Numbers with zero fractions compare equal even when they differ in sign or characteristic.

# CONVERT FROM FIXED

Mnemonic $R_1,R_2$ [RRE]




| Mnemonic | Op Code | Operands |
|---|---|---|
| CEFR | 'B3B4' | 32-bit binary-integer operand, short HFP result |
| CDFR | 'B3B5' | 32-bit binary-integer operand, long HFP result |
| CXFR | 'B3B6' | 32-bit binary-integer operand, extended HFP result |

The fixed-point second operand is converted to the HFP format, and the normalized result is placed at the first-operand location.

A nonzero result is normalized. A zero result is made a positive true zero.

The second operand is a 32-bit signed binary integer that is located in the general register designated by $R_2$.

The result is normalized and rounded toward zero (truncated) before it is placed at the first-operand location.

For CXFR, the $R_1$ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

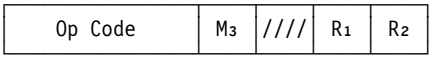
***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (if the HFP-extensions facility is not installed)
- Specification (CXFR only)

# CONVERT TO FIXED


LzLabs GmbH. Ex. 1008-787

```
Mnemonic     R1,M3,R2          [RRF]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| CFER | 'B3B8' | Short HFP operand, 32-bit binary-integer result |
| CFDR | 'B3B9' | Long HFP operand, 32-bit binary-integer result |
| CFXR | 'B3BA' | Extended HFP operand, 32-bit binary-integer result |

The HFP second operand is rounded to an integer value and then converted to the fixed-point format. The result is placed at the first-operand location.

The result is a 32-bit signed binary integer that is placed in the general register designated by $R_1$.

The second operand is rounded to an integer value by rounding as specified by the modifier in the $M_3$ field:

| $M_3$ | Rounding Method |
|---|---|
| 0 | Round toward 0 |
| 1 | Biased round to nearest |
| 4 | Round to nearest |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |
| 7 | Round toward $-\infty$ |

A modifier other than 0, 1, or 4-7 is invalid.

The sign of the result is the sign of the second operand, except that a zero result has a plus sign.

If the rounded result would have a value exceeding the range that can be represented in the result format, the largest (in magnitude) representable number of the same sign as the source is placed at the target location, and condition code 3 is set.

HFP exponent underflow is not recognized because small values are rounded to one (with the appropriate sign) or to zero, depending on the rounding mode.

The $M_3$ field must designate a valid modifier; otherwise, a specification exception is recognized. For CFXR, the $R_2$ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

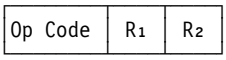
0 Source was zero
1 Source was less than zero
2 Source was greater than zero
3 Special case

***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (if the HFP-extensions facility is not installed)
- Specification

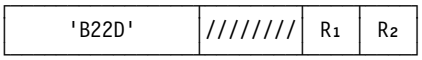
# DIVIDE

```
Mnemonic1    R1,R2             [RR]
```



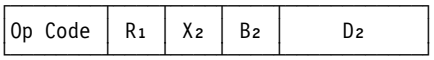

| Mnemonic1 | Op Code | Operands |
|---|---|---|
| DER | '3D' | Short HFP |
| DDR | '2D' | Long HFP |

```
Mnemonic2    R1,R2             [RRE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| DXR | 'B22D' | Extended HFP |

```
Mnemonic3    R1,D2(X2,B2)      [RX]
```




| Mnemonic3 | Op Code | Operands |
|---|---|---|
| DE | '7D' | Short HFP |
| DD | '6D' | Long HFP |

The first operand (the dividend) is divided by the second operand (the divisor), and the normalized quotient is placed at the first-operand location. No remainder is preserved.

HFP division consists in characteristic subtraction and fraction division. The operands are first normalized to eliminate leading hexadecimal zeros. The difference between the dividend and divisor characteristics of the normalized operands, plus 64, is used as the characteristic of an intermediate quotient.



LzLabs GmbH. Ex. 1008-788

All dividend and divisor fraction digits participate in forming the fraction of the intermediate quotient. The intermediate-quotient fraction can have no leading hexadecimal zeros, but a right shift of one digit position may be necessary, with this causing an increase of the characteristic by one. The fraction is then truncated to the proper result-fraction length.

An HFP-exponent-overflow exception exists when the characteristic of the final quotient would exceed 127 and the fraction is not zero. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct. If, for extended results, the low-order characteristic would also exceed 127, it too is decreased by 128.

An HFP-exponent-underflow exception exists when the characteristic of the final quotient would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero. For extended results, HFP exponent underflow is not recognized when the low-order characteristic is less than zero but the high-order characteristic is equal to or greater than zero.

HFP exponent underflow does not occur when the characteristic of an operand becomes less than zero during normalization of the operands or when the intermediate-quotient characteristic is less than zero, as long as the final quotient can be represented with the correct characteristic.

When the divisor fraction is zero, an HFP-divide exception is recognized. This includes the case of division of zero by zero.

When the dividend fraction is zero but the divisor fraction is nonzero, the quotient is made a positive true zero. No HFP exponent overflow or HFP exponent underflow occurs.

The sign of the quotient is the exclusive or of the operand signs, except that the sign is always plus when the quotient is made a positive true zero.

For DXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

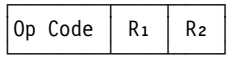
***Program Exceptions:***

- Access (fetch, operand 2 of DE and DD only)
- Data with DXC 1, AFP register
- HFP divide
- HFP exponent overflow
- HFP exponent underflow
- Specification

**Programming Note:** Examples of the use of the DIVIDE instruction (DER) are given in Appendix A.

## HALVE

```
Mnemonic    R1,R2                [RR]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| HER | '34' | Short HFP |
| HDR | '24' | Long HFP |

The second operand is divided by 2, and the normalized quotient is placed at the first-operand location.

The fraction of the second operand is shifted right one bit position, placing the contents of the rightmost bit position in the leftmost bit position of the guard digit, and a zero is supplied to the leftmost bit position of the fraction. The intermediate result, including the guard digit, is then normalized, and the final result is truncated to the proper length.

An HFP-exponent-underflow exception exists when the characteristic of the final result would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is


LzLabs GmbH. Ex. 1008-789

one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero.

When the fraction of the second operand is zero, the result is made a positive true zero, and no HFP exponent underflow occurs.

The sign of the result is the same as that of the second operand, except that the sign is always plus when the quotient is made a positive true zero.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

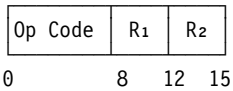
- Data with DXC 1, AFP register
- HFP exponent underflow
- Specification

**Programming Notes:**

1. An example of the use of the HALVE instruction (HDR) is given in Appendix A.
2. With short and long operands, the halve operation is identical to a divide operation with the number 2 as divisor. Similarly, the result of HDR is identical to that of MD or MDR with one-half as a multiplier, and the result of HER is identical to that of MEE or MEER with one-half as a multiplier.

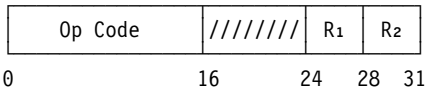
## LOAD AND TEST

Mnemonic1 $R_1$,$R_2$ [RR]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| LTER | '32' | Short HFP |
| LTDR | '22' | Long HFP |

Mnemonic2 $R_1$,$R_2$ [RRE]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| LTXR | 'B362' | Extended HFP |

The second operand is placed at the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code. The condition code is set the same as for a comparison of the second operand with zero.

For short and long operands, the second operand is placed unchanged in the first-operand location.

For extended operands, the high-order sign and the entire fraction of the source are placed unchanged in the result, and the low-order sign is set equal to the high-order sign. If the extended-operand fraction is nonzero, the high-order characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the extended-operand fraction is zero, the result is made a true zero with the same sign as the source (the high-order and low-order sign bits of the result are the same as the high-order sign bit of the source).

For LTXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.



LzLabs GmbH. Ex. 1008-790

***Resulting Condition Code:***

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --

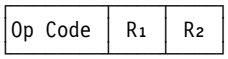
***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (LTXR if the HFP-extensions facility is not installed)
- Specification

**Programming Note:** When, for LTER and LTDR, the same register is designated as the first-operand and second-operand location, the operation is equivalent to a test without data movement.

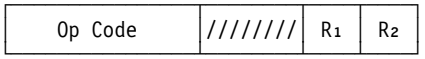
# LOAD COMPLEMENT

```
Mnemonic1    R1,R2                [RR]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| LCER | '33' | Short HFP |
| LCDR | '23' | Long HFP |

```
Mnemonic2    R1,R2              [RRE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| LCXR | 'B363' | Extended HFP |

The second operand is placed at the first-operand location with the sign bit inverted.

The sign bit is inverted even if the operand is zero. For all operand lengths, the source fraction is placed unchanged in the result.

For short and long operands, the source characteristic is placed unchanged in the result.

For extended operands, the low-order sign is set equal to the high-order sign. If the extended-operand fraction is nonzero, the high-order characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the extended-operand fraction is zero, the result is made a true zero with the sign inverted from the source (the high-order and low-order sign bits of the result are inverted from the high-order sign bit of the source).

For LCXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

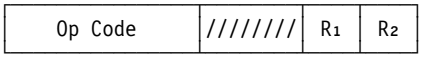
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --

***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (LCXR if the HFP-extensions facility is not installed)
- Specification

# LOAD FP INTEGER

```
Mnemonic     R1,R2              [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| FIER | 'B377' | Short HFP |
| FIDR | 'B37F' | Long HFP |
| FIXR | 'B367' | Extended HFP |

The second operand is truncated (rounded toward zero) to an integer value in the same floating-point format, and the normalized result is placed at the first-operand location.

A nonzero result is normalized. A zero result is made a positive true zero.

For FIXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.


LzLabs GmbH. Ex. 1008-791

***Program Exceptions:***

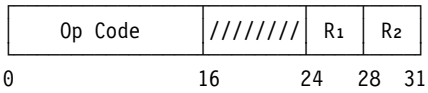
- Data with DXC 1, AFP register
- Operation (if the HFP-extensions facility is not installed)
- Specification (FIXR only)

**Programming Notes:**

1. LOAD FP INTEGER truncates (rounds toward zero) an HFP number to an integer value. These integers, which remain in the HFP format, should not be confused with binary integers, which use a fixed-point format.
2. If the HFP operand is numeric with a large enough exponent so that it is already an integer, the result value remains the same, except that an unnormalized operand is normalized, and an operand with a zero fraction is changed to a positive true zero.

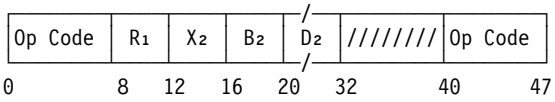
## LOAD LENGTHENED

```
Mnemonic1    R1,R2              [RRE]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| LDER | 'B324' | Short HFP operand 2, long HFP operand 1 |
| LXDR | 'B325' | Long HFP operand 2, extended HFP operand 1 |
| LXER | 'B326' | Short HFP operand 2, extended HFP operand 1 |

```
Mnemonic2    R1,D2(X2,B2)       [RXE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| LDE | 'ED24' | Short HFP operand 2, long HFP operand 1 |
| LXD | 'ED25' | Long HFP operand 2, extended HFP operand 1 |
| LXE | 'ED26' | Short HFP operand 2, extended HFP operand 1 |

The second operand is extended to a longer format, and the result is placed at the first-operand location.

For all operand lengths, the source fraction is extended with zeros and placed in the result. The sign bit of the result is set the same as the sign of the source even when the result is made a true zero.

For long results, the source characteristic is placed unchanged in the result.

For extended results, the low-order sign is set equal to the high-order sign. If the fraction is nonzero, the source characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the fraction is zero, the result is made a true zero with the same sign as the source (the high-order and low-order sign bits of the result are the same as the sign bit of the source).

For LXD, LXDR, LXE, and LXER, the $R_1$ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

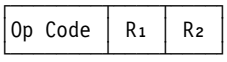
***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2 of LDE, LXE, and LXD only)
- Data with DXC 1, AFP register
- Operation (if the HFP-extensions facility is not installed)
- Specification (LXE, LXER, LXD, LXDR)

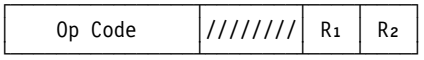
## LOAD NEGATIVE

```
Mnemonic1    R1,R2              [RR]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| LNER | '31' | Short HFP |
| LNDR | '21' | Long HFP |

```
Mnemonic2    R1,R2              [RRE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| LNXR | 'B361' | Extended HFP |

The second operand is placed at the first-operand location with the sign bit made one.



LzLabs GmbH. Ex. 1008-792

The sign bit is made one even if the operand is zero. For all operand lengths, the source fraction is placed unchanged in the result.

For short and long operands, the source characteristic is placed unchanged in the result.

For extended operands, the low-order sign is set equal to the high-order sign. If the extended-operand fraction is nonzero, the high-order characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the extended-operand fraction is zero, the result is made a negative true zero (the high-order and low-order sign bits of the result are set to one).

For LNXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

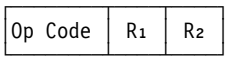
0 Result is zero
1 Result is less than zero
2 --
3 --

***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (LNXR if the HFP-extensions facility is not installed)
- Specification

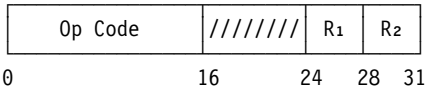
# LOAD POSITIVE

```
Mnemonic1    R1,R2                [RR]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| LPER | '30' | Short HFP |
| LPDR | '20' | Long HFP |

```
Mnemonic2    R1,R2                [RRE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| LPXR | 'B360' | Extended HFP |

The second operand is placed at the first-operand location with the sign bit made zero.

For all operand lengths, the sign bit is made zero, and the source fraction is placed unchanged in the result.

For short and long operands, the source characteristic is placed unchanged in the result.

For extended operands, the low-order sign is set equal to the high-order sign. If the extended-operand fraction is nonzero, the high-order characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the extended-operand fraction is zero, the result is made a positive true zero (the high-order and low-order sign bits of the result are set to zero).

For LPXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result is zero
1 --
2 Result is greater than zero
3 --

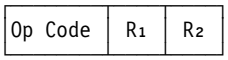
***Program Exceptions:***

- Data with DXC 1, AFP register
- Operation (LPXR if the HFP-extensions facility is not installed)
- Specification


LzLabs GmbH. Ex. 1008-793

## LOAD ROUNDED

```
Mnemonic1    R1,R2                [RR]
```

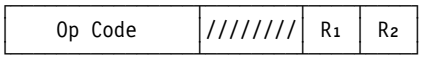



| Mnemonic1 | Op Code | Operands |
|---|---|---|
| LEDR | '35' | Long HFP operand 2, short HFP operand 1 |
| LDXR | '25' | Extended HFP operand 2, long HFP operand 1 |

The above mnemonics are alternatives to the following older mnemonics that are less descriptive of operand lengths:

| | | |
|---|---|---|
| LRER | '35' | Long HFP operand 2, short HFP operand 1 |
| LRDR | '25' | Extended HFP operand 2, long HFP operand 1 |

```
Mnemonic2    R1,R2                [RRE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| LEXR | 'B366' | Extended HFP operand 2, short HFP operand 1 |

The second operand is rounded to a shorter format, and the result is placed at the first-operand location.

Rounding consists in adding a one to the leftmost bit position of the second operand that is to be dropped and propagating any carry through the fraction. The sign of the second operand is ignored, and addition is performed as if the fraction were positive.

If rounding causes a carry out of the leftmost hexadecimal digit position of the fraction, the fraction is shifted right one digit position so that the carry becomes the leftmost digit of the fraction, and the characteristic is increased by one.

The intermediate fraction is then truncated to the proper result-fraction length. For LEDR and LEXR, the result replaces the leftmost 32 bits of the target register, and the rightmost 32 bit positions of the target register remain unchanged. For LDXR, the 64-bit result is placed in a floating-point register, not a floating-point register pair.

The sign of the result is the same as the sign of the second operand. There is no normalization to eliminate leading zeros.

An HFP-exponent-overflow exception exists when shifting the fraction right would cause the characteristic to exceed 127. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct.

HFP-exponent-underflow and HFP-significance exceptions cannot occur.

For LDXR and LEXR, the $R_2$ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

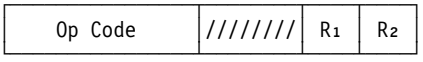
***Program Exceptions:***

- Data with DXC 1, AFP register
- HFP exponent overflow
- Operation (LEXR if the HFP-extensions facility is not installed)
- Specification

**Programming Note:** The sign of the rounded result is the same as the sign of the operand, even when the result is zero.

## MULTIPLY

```
Mnemonic1    R1,R2                [RRE]
```

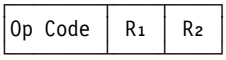



| Mnemonic1 | Op Code | Operands |
|---|---|---|
| MEER | 'B337' | Short HFP |



LzLabs GmbH. Ex. 1008-794

```
Mnemonic2    R1,R2                [RR]
```

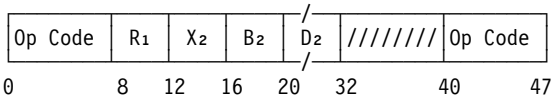



| Mnemonic2 | Op Code | Operands |
|---|---|---|
| MDR | '2C' | Long HFP |
| MXR | '26' | Extended HFP |
| MDER | '3C' | Short HFP multiplier and multiplicand, long HFP product |
| MXDR | '27' | Long HFP multiplier and multiplicand, extended HFP product |

The above mnemonic MDER is an alternative to the following older mnemonic that is less descriptive of operand lengths:

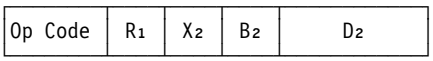
```
MER          '3C'     Short HFP multiplier and
                      multiplicand, long HFP
                      product
```

```
Mnemonic3    R1,D2(X2,B2)         [RXE]
```




| Mnemonic3 | Op Code | Operands |
|---|---|---|
| MEE | 'ED37' | Short HFP |

```
Mnemonic4    R1,D2(X2,B2)         [RX]
```




| Mnemonic4 | Op Code | Operands |
|---|---|---|
| MD | '6C' | Long HFP |
| MDE | '7C' | Short HFP multiplier and multiplicand, long HFP product |
| MXD | '67' | Long HFP multiplier and multiplicand, extended HFP product |

```
The above mnemonic MDE is an alternative to
the following older mnemonic that is less
descriptive of operand lengths:

ME           '7C'     Short HFP multiplier and
                      multiplicand, long HFP
                      product
```

The normalized product of the second operand (the multiplier) and the first operand (the multiplicand) is placed at the first-operand location.

Multiplication of two HFP numbers consists in exponent addition and fraction multiplication. The operands are first normalized to eliminate leading hexadecimal zeros. The sum of the characteristics of the normalized operands, less 64, is used as the characteristic of the intermediate product.

The fraction of the intermediate product is the exact product of the normalized operand fractions. If the intermediate-product fraction has one leading hexadecimal zero digit, the fraction is shifted left one digit position, bringing the contents of the guard-digit position into the rightmost position of the result fraction, and the intermediate-product characteristic is reduced by one. The fraction is then truncated to the proper result-fraction length.

For MDE and MDER, the multiplier and multiplicand fractions have six hexadecimal digits; the product fraction has the full 14 digits of the long format, with the two rightmost fraction digits always zeros. For MEE and MEER, the multiplier and multiplicand fractions have six digits, and the final product fraction is truncated to six digits; the result, as for all short-format results, replaces the leftmost 32 bits of the target register, and the rightmost 32 bit positions of the target register remain unchanged.

For MD and MDR, the multiplier and multiplicand fractions have 14 digits, and the final product fraction is truncated to 14 digits. For MXD and MXDR, the multiplier and multiplicand fractions have 14 digits, with the multiplicand occupying the high-order part of the first operand; the final product fraction contains 28 digits and is an exact product of the operand fractions. For MXR, the multiplier and multiplicand fractions have 28 digits, and the final product fraction is truncated to 28 digits.

An HFP-exponent-overflow exception exists when the characteristic of the final product would exceed 127 and the fraction is not zero. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct. If, for extended results, the low-order characteristic would also exceed 127, it too is decreased by 128.

HFP exponent overflow is not recognized when the intermediate-product characteristic is initially 128 but is brought back within range by normalization.


LzLabs GmbH. Ex. 1008-795

An HFP-exponent-underflow exception exists when the characteristic of the final product would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero. For extended results, HFP exponent underflow is not recognized when the low-order characteristic is less than zero but the high-order characteristic is equal to or greater than zero.

HFP exponent underflow does not occur when the characteristic of an operand becomes less than zero during normalization of the operands, as long as the final product can be represented with the correct characteristic.

If either or both operand fractions are zero, the result is made a positive true zero, and no HFP exponent overflow or HFP exponent underflow occurs.

The sign of the product is the exclusive or of the operand signs, except that the sign is always plus when the result is made a true zero.

The $R_1$ field for MXD, MXDR, and MXR, and the $R_2$ field for MXR must designate valid floating-point-register pairs. Otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

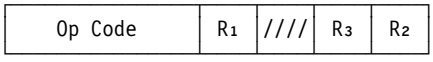
- Access (fetch, operand 2 of MDE, MEE, MD, and MXD only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP exponent underflow
- Operation (MEE and MEER if the HFP-extensions facility is not installed)
- Specification

**Programming Notes:**

1. An example of the use of the MULTIPLY instruction (MDR) is given in Appendix A.
2. Interchanging the two operands in an HFP multiplication does not affect the value of the product.

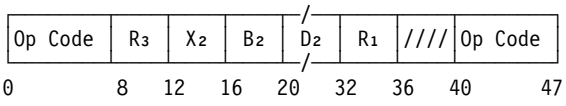
## MULTIPLY AND ADD

Mnemonic1 $R_1,R_3,R_2$ [RRF]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| MAER | 'B32E' | Short HFP |
| MADR | 'B33E' | Long HFP |

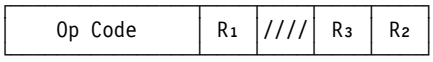
Mnemonic2 $R_1,R_3,D_2(X_2,B_2)$ [RXF]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| MAE | 'ED2E' | Short HFP |
| MAD | 'ED3E' | Long HFP |

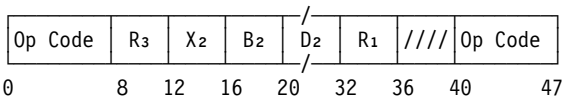
## MULTIPLY AND SUBTRACT

Mnemonic1 $R_1,R_3,R_2$ [RRF]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| MSER | 'B32F' | Short HFP |
| MSDR | 'B33F' | Long HFP |

Mnemonic2 $R_1,R_3,D_2(X_2,B_2)$ [RXF]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| MSE | 'ED2F' | Short HFP |
| MSD | 'ED3F' | Long HFP |

The third operand is multiplied by the second operand, and then the first operand is added to or subtracted from the product. The sum or difference is placed at the first-operand location. The MULTIPLY AND ADD and MULTIPLY AND SUBTRACT operations may be summarized as:



LzLabs GmbH. Ex. 1008-796

$op_1 = op_3 \cdot op_2 \pm op_1$

The third and second HFP operands are multiplied, forming an intermediate product, and the first operand is then added (or subtracted) algebraically to (or from) the intermediate product, forming an intermediate result. The exponent and fraction of the intermediate product and intermediate result are maintained exactly. The intermediate result, if nonzero, is normalized and truncated to the operand format and then placed at the first-operand location.

The sign of the result is determined by the rules of algebra, unless the intermediate-result fraction is zero, in which case the result is made a positive true zero.

An HFP-exponent-overflow exception exists when the characteristic of the normalized result would exceed 127 and the fraction is not zero. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct.

HFP exponent overflow is not recognized on intermediate values, provided the normalized result can be represented with the correct characteristic.

An HFP-exponent-underflow exception exists when the characteristic of the normalized result would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero.

HFP exponent underflow is not recognized on input operands and intermediate values, provided the normalized result can be represented with the correct characteristic.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

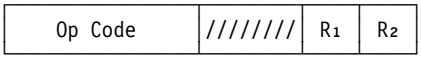
- Access (fetch, operand 2 of MAE, MAD, MSE, MSD)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP exponent underflow
- Operation (if the HFP multiply-add/subtract facility is not installed)

**Programming Note:** MULTIPLY AND ADD (SUBTRACT) differs from MULTIPLY followed by ADD (SUBTRACT) NORMALIZED in the following ways:

1. The product is maintained to full precision, and overflow and underflow are not recognized on the product.
2. The HFP-significance exception is not recognized for MULTIPLY AND ADD (SUBTRACT).
3. ADD (SUBTRACT) NORMALIZED maintains only a single guard digit and does not prenormalize input operands; thus, in some cases, an unnormalized input operand may cause loss of precision in the result. MULTIPLY AND ADD (SUBTRACT) maintains the entire intermediate sum (difference), which is normalized before the truncation operation is performed; thus, unnormalized operands do not cause any additional loss of precision.
4. On most models, the execution time of MULTIPLY AND ADD (SUBTRACT) is less than the combined execution time of MULTIPLY followed by ADD (SUBTRACT) NORMALIZED. The performance of MULTIPLY AND ADD (SUBTRACT) may be severely degraded in the case of unnormalized input operands.

## SQUARE ROOT

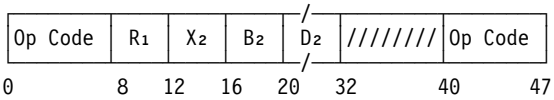
```
Mnemonic1    R1,R2                [RRE]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| SQER | 'B245' | Short HFP |
| SQDR | 'B244' | Long HFP |
| SQXR | 'B336' | Extended HFP |


LzLabs GmbH. Ex. 1008-797

```
Mnemonic2     R1,D2(X2,B2)          [RXE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| SQE | 'ED34' | Short HFP |
| SQD | 'ED35' | Long HFP |

The normalized and rounded square root of the second operand is placed at the first-operand location.

When the fraction of the second operand is zero, the sign and characteristic of the second operand are ignored, and the operation is completed by placing a positive true zero at the first-operand location.

If the second operand is less than zero, an HFP-square-root exception is recognized.

If the second operand is normalized and greater than zero, the characteristic, fraction, and sign of the result are produced as follows:

- The result characteristic is one-half of the sum of the operand characteristic and either 64, if the operand characteristic is even, or 65, if it is odd.
- If the operand characteristic is odd, the operand fraction is shifted right one digit position, the rightmost digit entering the guard-digit position.
- An intermediate-result fraction is produced by computing without rounding the square root of the operand fraction, after any right shift as described. The intermediate-result fraction consists of the 29 most significant hexadecimal digits of the square-root result in the extended format, 15 in the long format, or seven in the short format, where all three formats include a guard digit on the right.
- A one is added to the leftmost bit of the guard digit of the intermediate result, any carry is propagated to the left, and the guard digit is dropped to produce the result fraction.
- The result sign is made plus.

If the second operand is unnormalized and greater than zero, the operand is first normalized. The operation then proceeds as for normalized operands.

For SQXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***Program Exceptions:***

- Access (fetch, operand 2 of SQE and SQD only)
- Data with DXC 1, AFP register
- HFP square root
- Operation (SQER and SQDR if the square-root facility is not installed; SQE, SQD, and SQXR if the HFP-extensions facility is not installed)
- Specification

**Programming Notes:**

1. The use of the SQUARE ROOT instruction with short operands (SQER) is illustrated by the examples in the following table:

| Operand (hex) | Decimal Value | Result (hex) | Decimal Value |
|---|---|---|---|
| 42 190000 | 25.0 | 41 500000 | 5.0 |
| 40 400000 | 0.250 | 40 800000 | 0.50 |
| 40 800000 | 0.50 | 40 B504F3 | 0.7071... |
| 41 800000 | 8.0 | 41 2D413D | 2.8284... |

2. The result fraction is correctly normalized without any further left or right shifts of the intermediate-result fraction and without any further exponent adjustment. Rounding cannot cause a carry out of the leftmost digit.
3. Although a characteristic greater than 127 or less than zero may temporarily be generated during the operation, the result characteristic is always within the representable range, and no HFP exponent overflow or underflow occurs.

   Specifically, the smallest nonzero operand in the long format consists of a one bit, preceded on the left by 63 zeros. This operand is an unnormalized number with a value of $16^{-78}$, and its square root is $16^{-39}$. The normalized representation of this result has a characteristic of 26 (decimal). Similarly, the square root

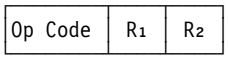


LzLabs GmbH. Ex. 1008-798

of the largest representable operand has a characteristic of 96 (decimal). The instruction, therefore, cannot produce a nonzero result with a characteristic outside the range of 26 to 96.

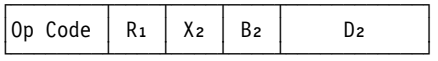
## SUBTRACT NORMALIZED

```
Mnemonic1    R1,R2                [RR]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| SER | '3B' | Short HFP |
| SDR | '2B' | Long HFP |
| SXR | '37' | Extended HFP |

```
Mnemonic2    R1,D2(X2,B2)         [RX]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| SE | '7B' | Short HFP |
| SD | '6B' | Long HFP |

The second operand is subtracted from the first operand, and the normalized difference is placed at the first-operand location.

The execution of SUBTRACT NORMALIZED is identical to that of ADD NORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

For SXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

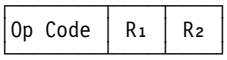
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --

***Program Exceptions:***

- Access (fetch, operand 2 of SE and SD only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP exponent underflow
- HFP significance
- Specification

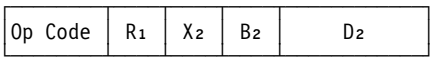
## SUBTRACT UNNORMALIZED

```
Mnemonic1    R1,R2                [RR]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| SUR | '3F' | Short HFP |
| SWR | '2F' | Long HFP |

```
Mnemonic2    R1,D2(X2,B2)         [RX]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| SU | '7F' | Short HFP |
| SW | '6F' | Long HFP |

The second operand is subtracted from the first operand, and the unnormalized difference is placed at the first-operand location.

The execution of SUBTRACT UNNORMALIZED is identical to that of ADD UNNORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

The R fields may designate the additional floating-point registers only when the basic-floating-point-extensions facility is installed; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result fraction zero
1 Result less than zero
2 Result greater than zero
3 --

***Program Exceptions:***

- Access (fetch, operand 2 of SU and SW only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP significance


LzLabs GmbH. Ex. 1008-799

- Specification



LzLabs GmbH. Ex. 1008-800

# Chapter 19. Binary-Floating-Point Instructions


Binary-Floating-Point Facility . . . . . 19-1
Floating-Point-Control (FPC) Register . . . 19-2
IEEE Masks and Flags . . . . . 19-3
FPC DXC Byte . . . . . 19-3
Operations on the FPC Register . . . . 19-3
BFP Arithmetic . . . . . 19-4
BFP Data Formats . . . . . 19-4
BFP Short Format . . . . . 19-4
BFP Long Format . . . . . 19-4
BFP Extended Format . . . . . 19-4
Biased Exponent . . . . . 19-4
Significand . . . . . 19-4
Values of Nonzero Numbers . . . . . 19-4
Classes of BFP Data . . . . . 19-5
Zeros . . . . . 19-6
Denormalized Numbers . . . . . 19-6
Normalized Numbers . . . . . 19-6
Infinities . . . . . 19-6
Signaling and Quiet NaNs . . . . . 19-6
BFP-Format Conversion . . . . . 19-7
BFP Rounding . . . . . 19-7
Rounding Mode . . . . . 19-7
Normalization and Denormalization . . . . 19-8
BFP Comparison . . . . . 19-8
Condition Codes for BFP Instructions . . . 19-9
Remainder . . . . . 19-9
IEEE Exception Conditions . . . . . 19-10
IEEE Invalid Operation . . . . . 19-10
IEEE Division-By-Zero . . . . . 19-11
IEEE Overflow . . . . . 19-11
IEEE Underflow . . . . . 19-12
IEEE Inexact . . . . . 19-12
Result Figures . . . . . 19-13
Data-Exception Codes (DXC) and Abbreviations . . . . . 19-14
Instructions . . . . . 19-15
ADD . . . . . 19-18
COMPARE . . . . . 19-23
COMPARE AND SIGNAL . . . . . 19-24
CONVERT FROM FIXED . . . . . 19-26
CONVERT TO FIXED . . . . . 19-27
DIVIDE . . . . . 19-29
DIVIDE TO INTEGER . . . . . 19-30
EXTRACT FPC . . . . . 19-34
LOAD AND TEST . . . . . 19-35
LOAD COMPLEMENT . . . . . 19-35
LOAD FP INTEGER . . . . . 19-36
LOAD FPC . . . . . 19-37
LOAD LENGTHENED . . . . . 19-38
LOAD NEGATIVE . . . . . 19-38
LOAD POSITIVE . . . . . 19-39
LOAD ROUNDED . . . . . 19-39
MULTIPLY . . . . . 19-40
MULTIPLY AND ADD . . . . . 19-42
MULTIPLY AND SUBTRACT . . . . . 19-42
SET FPC . . . . . 19-44
SET ROUNDING MODE . . . . . 19-44
SQUARE ROOT . . . . . 19-45
STORE FPC . . . . . 19-45
SUBTRACT . . . . . 19-45
TEST DATA CLASS . . . . . 19-46


## Binary-Floating-Point Facility

The binary-floating-point (BFP) facility provides instructions to operate on binary (radix-2) floating-point data.

BFP provides a number of important advantages over hexadecimal floating point (HFP):

- Greater precision and exponent range (except for numbers in the short format where HFP has the greater range).
- Automatic rounding to the nearest value for all arithmetic operations. There are directed-rounding options that may be used instead.
- Special entities of "infinity" and "Not-a-Number" (NaN), which are accepted and handled by arithmetic operations in a reasonable fashion. They provide better defaults for exponent overflow and invalid operations (such as division of zero by zero). This allows most programs to continue running without hiding such errors and without using specialized exception handlers.
- Exponent underflow gives "denormalized" numbers as the default, which provides more consistent results than the abrupt result of zero produced by the HFP instructions.
- The greater exponent range makes exponent overflow and underflow in correctly written

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-801

programs very unlikely, so that programmers may often be able to ignore these conditions.

- Both mask and flag bits are provided for all arithmetic exception conditions. The mask bits enable or disable interruptions. When interruptions are disabled, the flag bits keep track of exception conditions during execution so that warning messages may be issued.
- Programs can be migrated from and to workstations and other systems using different architectures and still give consistent results, provided that floating-point operations on the other systems also conform to the IEEE standard. This does not mean, however, that bit-wise compatible results can be guaranteed, because the standard allows implementation flexibility, especially in the presence of exceptions.

**Programming Note:** The bit representation of the BFP data formats in storage is defined to be left-to-right in a manner that is uniform for all numeric operands in the ESA/390 architecture. Although the format diagrams in the IEEE floating-point standard appear to use the same left-to-right bit sequence, the standard only defines the meaning of the bits without specifying how they appear in storage; the storage arrangement is left to the implementation. Several implementations in fact use other sequences; this may affect programs which are dependent on the bit representation of floating-point data in storage.

## Floating-Point-Control (FPC) Register

The floating-point-control (FPC) register is a 32-bit register that contains mask bits, flag bits, a data-exception code, and rounding-mode bits. An overview of the FPC register is shown in Figure 19-1. Details are shown in Figure 19-2 on page 19-3 and in Figure 19-3 on page 19-3. (In Figure 19-2, the abbreviations “IM” and “SF” are based on the terms “interruption mask” and “status flag,” respectively.)

The bits of the FPC register are often referred to as, for example, FPC 1.0, meaning bit 0 of byte 1 of the register.

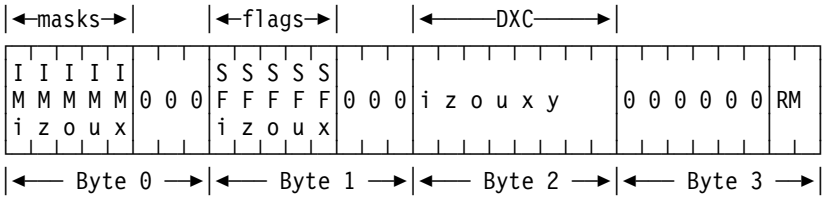



*Figure 19-1. FPC Register Overview*



LzLabs GmbH. Ex. 1008-802

| Byte | Bit(s) | Name | Abbr. |
|---|---|---|---|
| 0 | 0 | IEEE-invalid-operation mask | IMi |
| 0 | 1 | IEEE-division-by-zero mask | IMz |
| 0 | 2 | IEEE-overflow mask | IMo |
| 0 | 3 | IEEE-underflow mask | IMu |
| 0 | 4 | IEEE-inexact mask | IMx |
| 0 | 5-7 | (Reserved) | 0 |
| 1 | 0 | IEEE-invalid-operation flag | SFi |
| 1 | 1 | IEEE-division-by-zero flag | SFz |
| 1 | 2 | IEEE-overflow flag | SFo |
| 1 | 3 | IEEE-underflow flag | SFu |
| 1 | 4 | IEEE-inexact flag | SFx |
| 1 | 5-7 | (Reserved) | 0 |
| 2 | 0-7 | Data-exception code | DXC |
| 3 | 0-5 | (Reserved) | 0 |
| 3 | 6-7 | Rounding mode | RM |

*Figure 19-2. FPC-Register Bit Assignments*

| FPC Byte 3 Bits 6-7 | Rounding Mode |
|---|---|
| 00 | Round to nearest |
| 01 | Round toward 0 |
| 10 | Round toward +∞ |
| 11 | Round toward −∞ |

*Figure 19-3. Rounding Mode*

## IEEE Masks and Flags

The FPC register contains five IEEE mask bits and five IEEE flag bits that each correspond to one of the five arithmetic exception conditions that may occur when a BFP instruction is executed. The masks bits, when one, cause an interruption to occur if an exception condition is recognized. If the mask bit for an exception condition is zero, the recognition of the condition causes the corresponding flag bit to be set to one. Thus, a flag bit indicates whether the corresponding exception condition has been recognized at least once since the program last set the flag bit to zero. The mask bits are ignored, and the flag bits remain unchanged, when arithmetic exceptions are recognized for floating-point-support (FPS) and HFP instructions.

The IEEE flag bits in the FPC register are set to zero only by explicit program action, clear reset, or power-on reset.

## FPC DXC Byte

Byte 2 of the FPC register contains the data-exception code (DXC), which is an eight-bit code indicating the specific cause of a data exception. When the AFP-register-control bit, bit 13 of control register 0, is one and a program interruption causes the DXC to be placed at real location 147, the DXC is also placed in the DXC field of the FPC register. The DXC field in the FPC register remains unchanged when the AFP-register-control bit is zero or when any other program exception is reported. The DXC is described in "Data-Exception Code (DXC)" on page 6-15.

The DXC is a code, meaning it should be treated as an integer rather than as individual bits. However, when bits 6 and 7 are zero, bits 0-5 are bit significant; bits 0-4 (i,z,o,u,x) are trap flags and correspond to the same bits in bytes 0 and 1 of the FPC register (IEEE masks and IEEE flags), and bit 5 (y) is used in conjunction with bit 4, inexact (x), to indicate that the result has been incremented in magnitude. The trap flag for an exception, instead of the IEEE flag, is set to one when an interruption for the exception is enabled by the corresponding IEEE mask bit.

## Operations on the FPC Register

The following unprivileged BFP instructions allow problem-state programs to operate on the FPC register:

EXTRACT FPC
LOAD FPC
SET FPC
SET ROUNDING MODE
STORE FPC

These instructions are subject to the AFP-register-control bit, bit 13 of control register 0. An attempt to execute any of the above instructions when the AFP-register-control bit is zero results in a BFP-instruction data exception, DXC 2.


LzLabs GmbH. Ex. 1008-803

# BFP Arithmetic

# BFP Data Formats

Binary-floating-point numbers and NaNs may be represented in any of three formats: short, long, or extended.

## BFP Short Format

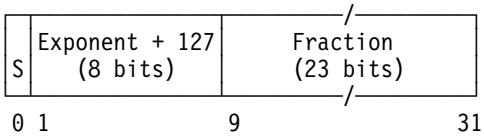



*Figure 19-4. BFP Short Format (4 bytes)*

When a number or NaN in the BFP short format is loaded into a floating-point register, it occupies the left half of the register, and the right half remains unchanged.

## BFP Long Format

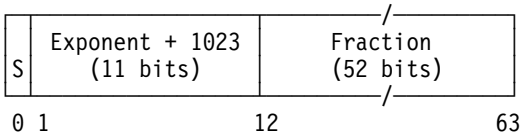



*Figure 19-5. BFP Long Format (8 bytes)*

When a number or NaN in the BFP long format is loaded into a floating-point register, it occupies the entire register.

## BFP Extended Format

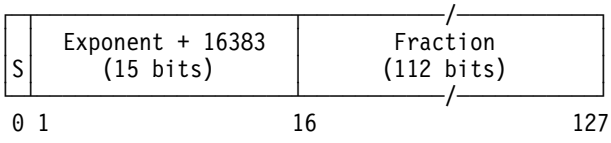



*Figure 19-6. BFP Extended Format (16 bytes)*

A number or NaN in the BFP extended format occupies a register pair. The sign and biased exponent are in the leftmost 16 bits of the left register and are followed by the leftmost 48 bits of the fraction. The rightmost 64 bits of the fraction are in the right register of the pair.

The properties of the three formats are tabulated in Figure 19-7 on page 19-5.

### Biased Exponent

For each format, the bias that is used to allow all exponents to be expressed as unsigned numbers is shown in the Figure 19-7 on page 19-5. Biased exponents are similar to the characteristics of the HFP format, except that special meanings are attached to biased exponents of all zeros and all ones, which are discussed in the section "Classes of BFP Data" on page 19-5.

### Significand

In each format, the binary point of a BFP number is considered to be to the left of the leftmost fraction bit. To the left of the binary point there is an implied unit bit, which is considered to be one for normalized numbers and zero for zeros and denormalized numbers. The fraction with the implied unit bit appended on the left is the *significand* of the number.

The value of a normalized BFP number is the significand multiplied by the radix 2 raised to the power of the unbiased exponent. The value of a denormalized BFP number is the significand multiplied by the radix 2 raised to the power of the minimum exponent.

A value of one in the rightmost bit position of the significand in each format is sometimes referred to as one ulp (unit in the last place).

### Values of Nonzero Numbers

The values of nonzero numbers in the various formats are shown in Figure 19-8.

| Number Class | Format | Value |
|---|---|---|
| Normalized | Short | $\pm 2^{e-127} \times (1.f)$ |
| | Long | $\pm 2^{e-1023} \times (1.f)$ |
| | Extended | $\pm 2^{e-16383} \times (1.f)$ |
| Denormalized | Short | $\pm 2^{-126} \times (0.f)$ |
| | Long | $\pm 2^{-1022} \times (0.f)$ |
| | Extended | $\pm 2^{-16382} \times (0.f)$ |
| **Explanation:**<br>e Biased exponent (shown in decimal).<br>f Fraction (in binary). | | |

*Figure 19-8. Values of Nonzero Numbers*

**Programming Note:** The IEEE standard specifies minimum requirements for the extended



LzLabs GmbH. Ex. 1008-804

| Property | Format | | |
|---|---|---|---|
| | Short | Long | Extended |
| Format length (bits) | 32 | 64 | 128 |
| Biased-exponent length (bits) | 8 | 11 | 15 |
| Fraction length (bits) | 23 | 52 | 112 |
| Precision (p) | 24 | 53 | 113 |
| Maximum exponent (Emax) | 127 | 1023 | 16383 |
| Minimum exponent (Emin) | -126 | -1022 | -16382 |
| Exponent bias | 127 | 1023 | 16383 |
| Nmax | $(1-2^{-24})\times2^{128}$ $\simeq3.4\times10^{38}$ | $(1-2^{-53})\times2^{1024}$ $\simeq1.8\times10^{308}$ | $(1-2^{-113})\times2^{16384}$ $\simeq1.2\times10^{4932}$ |
| Nmin | $1.0\times2^{-126}$ $\simeq1.2\times10^{-38}$ | $1.0\times2^{-1022}$ $\simeq2.2\times10^{-308}$ | $1.0\times2^{-16382}$ $\simeq3.4\times10^{-4932}$ |
| Dmin | $1.0\times2^{-149}$ $\simeq1.4\times10^{-45}$ | $1.0\times2^{-1074}$ $\simeq4.9\times10^{-324}$ | $1.0\times2^{-16494}$ $\simeq6.5\times10^{-4966}$ |

**Explanation:**

≃ Value is approximate.
Dmin Smallest (in magnitude) representable denormalized number.
Nmax Largest (in magnitude) representable number.
Nmin Smallest (in magnitude) representable normalized number.

*Figure 19-7. Summary of BFP Data Formats*

format but does not include details. The BFP extended format meets these requirements, far exceeding them in the area of precision.

## Classes of BFP Data

There are six classes of BFP data, which include numeric and related nonnumeric entities. Each data item consists of a sign, an exponent, and a significand. The exponent is biased such that all biased exponents are nonnegative unsigned numbers and the minimum biased exponent is zero. The significand consists of an explicit fraction and an implicit unit bit to the left of the binary point. The sign bit is zero for plus and one for minus.

All finite nonzero numbers within the normalized range permitted by a given format have a unique BFP representation. There are no unnormalized numbers, which numbers might allow multiple representations for the same values, and there are no unnormalized arithmetic operations. Tiny numbers of a magnitude below the minimum normalized number in a given format are represented as *denormalized* numbers, but those values are also represented uniquely. The implied unit bit of a normalized number is one, and that of a a denormalized number or a zero is zero.

The six classes of BFP data are summarized in Figure 19-9 on page 19-6.


LzLabs GmbH. Ex. 1008-805

| Data Class | Sign | Biased Exponent | Unit Bit* | Fraction |
|---|---|---|---|---|
| Zero | ± | 0 | 0 | 0 |
| Denormalized numbers | ± | 0** | 0 | Not 0 |
| Normalized numbers | ± | Not 0, not all ones | 1 | Any |
| Infinity | ± | All ones | — | 0 |
| Quiet NaN | ± | All ones | — | F0=1, Fr=any |
| Signaling NaN | ± | All ones | — | F0=0, Fr≠0 |

**Explanation:**

— Does not apply.
* The unit bit is implied.
** The biased exponent is treated arithmetically as if it had the value one.
F0 Leftmost bit of fraction.
Fr Remaining bits of fraction.
NaN Not-a-number.

*Figure 19-9. Classes of BFP Data*

The instruction TEST DATA CLASS may be used to determine the class of a BFP operand.

## Zeros

Zeros have a biased exponent of zero and a zero fraction. The implied unit bit is zero. A +0 is distinct from −0, except that comparison treats them as equal.

## Denormalized Numbers

Denormalized numbers are numbers which are smaller than the smallest normalized number and greater than zero in magnitude. They have a biased exponent of zero and a nonzero fraction. The biased exponent is treated arithmetically as if it were one, which causes the exponent to be the minimum exponent. The implied unit bit is zero.

## Normalized Numbers

Normalized numbers have a biased exponent greater than zero but less than all ones. The implied unit bit is one, and the fraction may have any value.

## Infinities

An infinity is represented by a biased exponent of all ones and a zero fraction. Infinities can participate in most arithmetic operations and give a consistent result, usually infinity. In comparisons, $+\infty$ compares greater than any finite number, and $-\infty$ compares less than any finite number.

## Signaling and Quiet NaNs

A NaN (not-a-number) entity is represented by a biased exponent of all ones and a nonzero fraction. NaNs are produced in place of a numeric result after an invalid operation when there is no interruption. NaNs may also be used by the program to flag special operands, such as the contents of an uninitialized storage area.

There are two types of NaNs, signaling and quiet. A signaling NaN (SNaN) is distinguished from the corresponding quiet NaN (QNaN) by the leftmost fraction bit: zero for the SNaN and one for the QNaN. A special QNaN is supplied as the default result for an IEEE-invalid-operation condition; it has a plus sign and a leftmost fraction bit of one, with the remaining fraction bits being set to zeros.

Normally, QNaNs are just propagated during computations so that they will remain visible at the end. An SNaN operand causes an IEEE-invalid-operation exception. If the IEEE-invalid-operation mask (FPC 0.0) is zero, the result is the corresponding QNaN, which is produced by setting the leftmost fraction bit to one, and the IEEE-invalid-operation flag (FPC 1.0) is set to one. If the IEEE-invalid-operation mask (FPC 0.0) is one, the operation is suppressed, and a data exception for IEEE-invalid operation occurs.

**Programming Notes:**

1. The program can generate and assign meanings to any nonzero fraction values of a NaN. The CPU propagates those values unchanged, except that an SNaN is changed to the corresponding QNaN if the IEEE-invalid-operation mask bit is zero, and conversion to a narrower format may truncate significant bits on the right.
2. The standard requires SNaNs to signal the invalid-operation exception for the arithmetic, comparison, and conversion operations that are part of the standard, but it makes it an implementation option whether copying an SNaN without a change of format signals the



LzLabs GmbH. Ex. 1008-806

exception. In the appendix, the standard also makes it an implementation option whether SNaNs should signal the invalid-operation exception for the recommended functions of copying the sign, taking the absolute value, reversing the sign, and testing the data class of a number.

The above functions generally correspond to the instructions LOAD, LOAD COMPLEMENT, LOAD NEGATIVE, LOAD POSITIVE, and TEST DATA CLASS. These instructions do not signal the invalid-operation exception but, instead, treat SNaNs like any other data; giving an exception would be disruptive when the intention is to include SNaNs. TEST DATA CLASS does not give an exception since it is the instruction with which to test for the presence of SNaNs.

3. LOAD AND TEST signals the invalid-operation exception when the operand is an SNaN. This instruction, in conjunction with the above instructions, gives the program the choice of either option permitted by the standard.
4. Load-type instructions which change the precision signal the invalid-operation exception when the operand is an SNaN, as this is required by the standard.

## BFP-Format Conversion

The instructions LOAD LENGTHENED and LOAD ROUNDED perform conversions of numbers between the short, long, and extended formats. For BFP formats, conversion involves adjustments to both the fraction and the exponent. When converting a normalized number to a wider format (short to long, long to extended, or short to extended), the fraction is adjusted by appending sufficient zeros on the right. Conversion to a narrower format requires rounding of the fraction before dropping excess bits on the right, and an IEEE-inexact condition may result.

The exponent is adjusted by adding or subtracting the difference in the biases of the two formats. When converting to a narrower format, this adjustment causes IEEE underflow if the resultant biased exponent would be less than one, or IEEE overflow if the resultant exponent would be equal to or greater than the maximum exponent for the new format.

When a denormalized number is converted to a wider format, the biased exponent of the source operand is treated as if it had the value one. The result is normalized.

**Programming Notes:**

1. When a NaN is converted to a narrower format, the appropriate number of fraction bits on the right are simply dropped with no indication. This is unlike the conversion of nonzero numbers, where the loss of nonzero fraction bits causes an IEEE-inexact condition. Thus, programs which encode NaN fraction bits for specific purposes must ensure that the distinguishing bits are placed in the left part of the fraction.
2. Converting a NaN from a wide format to a narrower format cannot turn the NaN into an infinity because an SNaN either causes an interruption or turns into a QNaN, and all QNaNs have a leftmost fraction bit of one.

## BFP Rounding

Arithmetic and conversion operations are performed as if they first produced an intermediate result correct to infinite precision and with unbounded range. If this intermediate result can be represented exactly in the target format, then it is given exactly. Otherwise, the intermediate result is replaced by one of the two closest values that can be represented, the choice depending on the rounding mode.

Rounding is performed automatically as part of every arithmetic and conversion operation. The precision of the target (short, long, or extended) is specified by the operation code.

### Rounding Mode

There are four rounding modes. The current rounding mode is specified by the value of two rounding-mode bits in the FPC register, as follows:

**00** Round to nearest (default). Round the intermediate result up or down to the nearest representable value; that is, add, ignoring the sign, a one to the bit just beyond the last result bit to be retained, propagate the carry, and discard the bits beyond the last one to be retained. If the difference was exactly one-half ulp (a one in the bit position just beyond the last place, with all zeros beyond


LzLabs GmbH. Ex. 1008-807

that), the nearest even number is chosen; that is, after the rounding addition, the last result bit retained is set to zero.

If the absolute value of the intermediate result is equal to or greater than the largest representable number plus one-half ulp, that is, if the absolute value is equal to or greater than $2^{Emax} \times (2 - 2^{-p})$, the result is rounded to infinity with the same sign as the intermediate result.

**01** Round toward 0. Discard all bits to the right of the last intermediate-result bit to be retained.

**10** Round toward +∞. If the intermediate result is positive and there are any ones to the right of the last result bit to be retained, add one to that bit. Then, for either sign, discard the bits beyond the last one to be retained.

**11** Round toward −∞. If the intermediate result is negative and there are any ones to the right of the last result bit to be retained, subtract one from that bit (that is, add one to the magnitude). Then, for either sign, discard the bits beyond the last one to be retained.

**Programming Notes:**

1. Rounding a finite result toward zero cannot give infinity.
2. Rounding a result toward +∞ can give +∞ but not −∞.
3. Rounding a result toward −∞ can give −∞ but not +∞.

## Normalization and Denormalization

Every arithmetic or conversion operation is considered to produce an intermediate result as if the precision and exponent range were unbounded, unless the result is defined to be zero, infinity, or NaN. The final result is produced by normalizing and then rounding this intermediate result. When there is exponent underflow, that is, the biased exponent of the normalized intermediate result is less than one, then the intermediate result is denormalized to produce the final result, as described below.

Denormalization consists in shifting the significand, including the units bit, to the right while introducing zero bits on the left, and in increasing the exponent by one for each bit of shift. When the biased exponent reaches +1, the significand is rounded according to the current rounding mode. If all bits of the rounded significand are zeros, the result is made zero. If rounding produces a carry into the units bit position of the significand, the biased exponent remains +1, since this result is a normalized number ($\pm 2^{Emin}$). Otherwise, the units bit remains zero, the biased exponent is set to zero, and the result is considered denormalized.

Arithmetic operations on denormalized operands are performed as if the operands had first been normalized.

Intermediate results are first normalized or denormalized, as required, and then rounded. This avoids double rounding of a single operation, which might increase the rounding error. (Any right shift required after a carry from rounding to renormalize the result does not require a second rounding, because the bit shifted off on the right is always zero.)

## BFP Comparison

Comparisons are always exact and cannot cause an IEEE-inexact condition.

Comparison ignores the sign of zero, that is, +0 equals −0.

Infinities with like sign compare equal, that is, +∞ equals +∞, and −∞ equals −∞.

A NaN compares as *unordered* with any other operand, whether a finite number, an infinity, or another NaN, including itself.

Two sets of instructions are provided: COMPARE and COMPARE AND SIGNAL. In the absence of QNaNs, these instructions work the same. These instructions work differently only when both of the following are true:

- Neither operand of the instruction is an SNaN
- At least one operand of the instruction is a QNaN

In this case, COMPARE simply sets condition code 3, but COMPARE AND SIGNAL recognizes the IEEE-invalid-operation condition. If any



LzLabs GmbH. Ex. 1008-808

operand is an SNaN, both instructions recognize the IEEE-invalid-operation condition.

The action when the IEEE-invalid-operation condition is recognized depends on the IEEE-invalid-operation mask bit in the FPC register. If the mask bit is zero, then the instruction execution is completed by setting condition code 3, and the IEEE-invalid-operation flag in the FPC register is set to one. If the mask bit is one, then the condition is reported as a program interruption for a data exception with DXC 80 hex (IEEE invalid operation).

**Programming Note:** A compiler can select either COMPARE or COMPARE AND SIGNAL for a comparison, depending on whether the IEEE standard or a relevant language standard requires a QNaN to be recognized as an exception condition.

## Condition Codes for BFP Instructions

For arithmetic operations with finite or infinite numeric results, condition codes 0, 1, and 2 are set to indicate that the result is a zero of either sign, less than zero, or greater than zero, respectively. The condition-code setting depends only on an inspection of the rounded result. For comparison operations, condition codes 0, 1, and 2 indicate equal, low, or high, respectively. These settings are the same as for the HFP instructions.

Condition code 3 can also be set. After an arithmetic operation, condition code 3 indicates a NaN result of either sign. After a comparison, it indicates that a NaN was involved in the comparison (the unordered condition). See Figure 19-10.

| CC | Arithmetic | Comparison |
|---|---|---|
| 0 | ±0 | Equal |
| 1 | <0 | Low |
| 2 | >0 | High |
| 3 | ±NaN | Unordered |

*Figure 19-10. Condition Codes*

## Remainder

The instruction DIVIDE TO INTEGER produces two floating-point results, an exact integer quotient and the corresponding remainder. The remainder is defined as follows:

Let

```
a = Dividend
b = Divisor
q = Exact quotient (a÷b)
r = Remainder
```

in the selected floating-point format. Then

```
r = a−b·n
```

where n is an integer. If q is an integer, then n equals q. Otherwise, n is obtained by rounding q according to a specified quotient rounding mode.

When the specified quotient rounding mode is round to nearest or round toward zero, the remainder is exact for any finite dividend and any nonzero divisor. The remainder cannot overflow.

If the integer quotient has a value that lies outside the range of the operand format, a wrapped result is provided.

In certain cases where the number of bits in the integer quotient exceeds or may exceed the maximum number of bits provided in the precision of the operand format, partial results are produced, and more than one execution of the instruction is required to obtain the final result; this may be done with a simple instruction loop.

Partial results are produced when the precise quotient is not an integer and the two integers closest to this precise quotient cannot both be represented exactly in the precision of the quotient. This situation exists when the precise quotient is greater than $2^P$, where P is the precision of the operand format, and the remainder is not zero. When the remainder is zero, then the quotient is an integer, and the number of bits required to represent the quotient is never more than the precision of the target.

**Programming Note:** The remainder result of DIVIDE TO INTEGER with a specified quotient rounding mode of round to nearest corresponds to the *Remainder* function in the IEEE standard. This function is similar to the *MOD* function found in some languages and to the mathematical


LzLabs GmbH. Ex. 1008-809

*modulo* function, but they are not the same. They differ in the definition of n:

| | |
|---|---|
| Remainder | n is q rounded to nearest. |
| modulo | n is q rounded toward $-\infty$. |
| MOD | n is q rounded toward 0. |

Another important difference is that implementations of modulo and MOD may put range restrictions on the result because they may simply use the DIVIDE instruction and accept its range restrictions.

The MOD definition provides an exact result, as does Remainder, but the modulo definition may result in rounding errors.

The differences between the various methods may be illustrated by the simple example of computing a divided by b to obtain an integer quotient n, where a is a series of integers, and b is +4 or –4. Figure 19-11 on page 19-11 shows the results for the three definitions.

The result of Remainder lies in the range of zero to one-half the divisor, inclusive, in magnitude. A zero result is defined to have the sign of the dividend. A zero divisor is invalid.

The modulo and MOD results can both be computed from the Remainder result; the reverse may not be true, because of rounding errors and, depending on the implementation, range restrictions.

An extreme example of the rounding error that can occur with the modulo definition is the following, where the result is restricted to two significant decimal digits:

```
modulo(0.01,-95) = -94.99, which rounds to -95

Remainder(0.01,-95) = 0.01
```

The properly rounded modulo result is completely wrong since it is equal to the divisor instead of being smaller in magnitude. The Remainder result is exact and can be used to compute the theoretical result of modulo.

Remainder is included as an arithmetic operation because of its usefulness in argument reduction when computing elementary transcendental functions. Thus, SIN(X) can be computed to full precision for any value of X in degrees by first reducing the argument to Remainder(X,360).

## IEEE Exception Conditions

The results of each of the IEEE exception conditions are controlled by a mask bit in the FPC register. When an IEEE exception condition is recognized, one of two actions is taken:

- If the corresponding mask bit in the FPC register is zero, a default action is taken, as specified for each condition, and the corresponding flag bit in the FPC register is set to one. Program execution then continues normally.
- If the corresponding mask bit in the FPC register is one, a program interruption for a data exception occurs, the operation is suppressed or completed, depending on the condition, and the data-exception code (DXC) assigned for that condition is provided.

### IEEE Invalid Operation

An IEEE-invalid-operation condition is recognized when, in the execution of a BFP instruction, any of the following occurs:

1. An SNaN is encountered in any BFP arithmetic, comparison, or conversion operation or by LOAD AND TEST.
2. A QNaN is encountered in a BFP comparison by COMPARE AND SIGNAL.
3. A BFP difference is undefined (addition of infinities of opposite sign, or subtraction of infinities of like sign).
4. A BFP product is undefined (zero times infinity).
5. A BFP quotient is undefined (DIVIDE instruction with both operands zero or both operands infinity).
6. A BFP remainder is undefined (DIVIDE TO INTEGER with a dividend of infinity or a divisor of zero).
7. A BFP square root is undefined (negative nonzero operand).

If the IEEE-invalid-operation mask bit in the FPC register is zero, the IEEE-invalid-operation flag bit in the FPC register is set to one. The completion of the operation depends on the type of operation and the operands.



LzLabs GmbH. Ex. 1008-810

| | a<br>-8 -7 -6 -5 -4 -3 -2 -1 -0 +0 +1 +2 +3 +4 +5 +6 +7 +8 |
|---|---|
| | Remainder |
| b=+4: n | -2 -2 -2 -1 -1 -1 -0 -0 -0 +0 +0 +0 +1 +1 +1 +2 +2 +2 |
| r | -0 +1 +2 -1 -0 1 -2 -1 -0 +0 +1 +2 -1 +0 +1 -2 -1 +0 |
| b=-4: n | +2 +2 +2 +1 +1 +1 +0 +0 +0 -0 -0 -0 -1 -1 -1 -2 -2 -2 |
| r | -0 +1 +2 -1 -0 +1 -2 -1 -0 +0 +1 +2 -1 +0 +1 -2 -1 +0 |
| | MOD |
| b=+4: n | -2 -1 -1 -1 -1 0 0 0 0 0 0 0 +1 +1 +1 +1 +2 |
| r | 0 -3 -2 -1 0 -3 -2 -1 0 +1 +2 +3 0 +1 +2 +3 0 |
| b=-4: n | +2 +1 +1 +1 +1 0 0 0 0 0 0 0 -1 -1 -1 -1 -2 |
| r | 0 -3 -2 -1 0 -3 -2 -1 0 +1 +2 +3 0 +1 +2 +3 0 |
| | modulo |
| b=+4: n | -2 -2 -2 -2 -1 -1 -1 -1 0 0 0 0 +1 +1 +1 +1 +2 |
| r | 0 +1 +2 +3 0 +1 +2 +3 0 +1 +2 +3 0 +1 +2 +3 0 |
| b=-4: n | +2 +1 +1 +1 +1 0 0 0 0 -1 -1 -1 -1 -2 -2 -2 -2 |
| r | 0 -3 -2 -1 0 -3 -2 -1 0 -3 -2 -1 0 -3 -2 -1 0 |

**Explanation:**

a Dividend.
b Divisor.
n Integer quotient.
r Result (Remainder, MOD, or modulo).

*Figure 19-11. Comparison of Remainder with MOD and Modulo*

If the instruction performs a comparison and no program interruption occurs, the comparison result is *unordered*.

If the instruction is one that produces a BFP result, if no program interruption occurs, and if none of the operands is a NaN, the result is the default QNaN. If one of the operands is a NaN, that operand becomes the result unchanged, except that an SNaN is first converted to the corresponding QNaN by setting the leftmost fraction bit to one.

If the IEEE-invalid-operation mask bit in the FPC register is one, the operation is suppressed, and the condition is reported as a program interruption for a data exception with DXC 80 hex.

## IEEE Division-By-Zero

An IEEE-division-by-zero condition is recognized when in BFP division the divisor is zero and the dividend is a finite nonzero number.

If the IEEE-division-by-zero mask bit in the FPC register is zero, the IEEE-division-by-zero flag bit in the FPC register is set to one. The operation is completed using as the result an infinity with a sign that is the exclusive or of the dividend and divisor signs.

If the IEEE-division-by-zero mask bit in the FPC register is one, the operation is suppressed, and the condition is reported as a program interruption for a data exception with DXC 40 hex.

## IEEE Overflow

An IEEE-overflow condition is recognized when the exponent of the rounded result of a BFP operation would be greater than the maximum exponent of the target format if the exponent range were unbounded.

If the IEEE-overflow mask bit in the FPC register is zero, the IEEE-overflow flag bit in the FPC register is set to one. The result of the operation depends on the sign of the intermediate result and on the current rounding mode:

1. When rounding to nearest, the result is infinity with the sign of the intermediate result.
2. When rounding toward 0, the result is the largest finite number of the format, with the sign of the intermediate result.


LzLabs GmbH. Ex. 1008-811

3. When rounding toward +∞, the result is +∞ if the sign is plus, or it is the finite negative number with the largest magnitude if the sign is minus.
4. When rounding toward −∞, the result is the largest finite positive number if the sign is plus or −∞ if the sign is minus.

If the IEEE-overflow mask bit in the FPC register is one, the operation is completed by producing a wrapped result, and the condition is reported as a program interruption for a data exception with DXC 20, 28, or 2C hex, depending on whether the wrapped result is exact, inexact and truncated, or inexact and incremented, respectively.

## IEEE Underflow

An IEEE-underflow condition is recognized when the exponent of the exact result of a BFP operation would be less than the minimum exponent of the target format.

If the IEEE-underflow mask bit in the FPC register is zero, then the action depends on whether the result can be represented exactly and, if not, also on the setting of the IEEE-inexact mask bit in the FPC register. If the result can be represented exactly, the operation is completed by denormalizing the intermediate result. If the result cannot be represented exactly and the IEEE-inexact mask bit in the FPC register is zero, the operation is completed by denormalizing and rounding the intermediate result, and the IEEE-underflow and IEEE-inexact flag bits in the FPC register are set to ones. If the result cannot be represented exactly and the IEEE-inexact mask bit in the FPC register is one, the IEEE-underflow flag bit in the FPC register is set to one, and the inexact condition is reported as a program interruption for a data exception with DXC 08 or 0C hex, depending on whether the result is inexact and truncated or inexact and incremented, respectively.

If the IEEE-underflow mask bit in the FPC register is one, then, regardless of whether the result could have been represented exactly, the operation is completed by producing a wrapped result, and the condition is reported as a program interruption for a data exception with DXC 10, 18, or 1C hex, depending on whether the wrapped result is exact, inexact and truncated, or inexact and incremented, respectively.

## IEEE Inexact

An IEEE-inexact condition is recognized when the rounded result of a BFP operation differs in value from the intermediate result computed as if exponent range and precision were unbounded. The condition is also recognized if rounding the result causes IEEE overflow and the IEEE-overflow mask bit is zero. The operation is completed using the rounded result or, in case of overflow or underflow, the result specified for IEEE overflow or IEEE underflow.

If the IEEE-inexact mask bit in the FPC register is zero, the IEEE-inexact flag bit in the FPC register is set to one.

If the IEEE-inexact mask bit in the FPC register is one, the operation is completed, and the condition is reported as a program interruption for a data exception with DXC 08 or 0C hex, depending on whether the result is inexact and truncated or inexact and incremented, respectively.

**Programming Notes:**

1. All IEEE traps are reported by means of a program interruption for a data exception with a data-exception code. The use of data exception provides the application program with a convenient interface since this exception is one of the original 15 exceptions in the System/360 architecture and is supported by most control programs that support the ESA/390 architecture.
2. The IEEE standard includes recommendations for the trap handler. When a system traps, the trap handler should be able to determine:
   a. Which exception(s) occurred on this operation.
   b. The kind of operation that was being performed.
   c. The destination's format.
   d. For overflow, underflow, and inexact exceptions, the correctly rounded result, including information that might not fit in the destination's format.
   e. For invalid-operation and divide-by-zero exceptions, the operand values.

   Items a and d are supplied as part of the interruption action. Items b, c, and e can be obtained starting with the instruction address



LzLabs GmbH. Ex. 1008-812

in the old PSW and from this finding the instruction (which indicates the operation and format) and then the operands.

3. The description of underflow is one of the most difficult parts of the standard to understand. This is because:

   a. The condition is described as two "correlated events" — "tininess" and "loss of accuracy."

   b. For tininess, the standard provides two options for detection: "after rounding" or "before rounding."

   c. For loss of accuracy, the standard provides two options for detection: "denormalization loss" or "inexact result."

   d. Implementation of the trap is optional.

   e. The conditions to signal underflow are different depending on whether or not the trap is taken.

   Each of the above items is discussed below.

   a. Tininess refers to a nonzero number strictly between $\pm 2^{Emin}$. (All denormalized numbers are in this range.) Loss of accuracy means that the result cannot be represented exactly.

   b. Detection of tininess after or before rounding differs only for the case when "rounding" would increase the magnitude of the result to exactly $\pm 2^{Emin}$. It must be noted, however, that the action which the standard here calls "rounding" is not the rounding to produce the delivered result but rounding to compute an intermediate value having the precision of the result but "as though the exponent range were unbounded." In fact, it is possible that the delivered result may not be tiny even though the intermediate value "after rounding" is tiny.

      The option selected in the ESA/390 BFP architecture (and the RS/6000) is to detect tininess before rounding.

   c. The difference between detection of loss of accuracy as a denormalization loss or as an inexact result can best be understood by considering two intermediate values: (1) a precise intermediate value, which has unbounded precision and unbounded exponent range, and (2) a rounded intermediate value, which is obtained by rounding the precise intermediate value to the precision of the result but with unbounded exponent range. Inexact result is said to occur when the delivered result differs from the precise intermediate value. Denormalization loss is said to occur when the delivered result differs from the rounded intermediate value. The two options differ in the case when the delivered result is equal in value to the rounded intermediate value but these are not equal to the precise intermediate value. Although the standard uses the term "denormalization loss," this condition includes a case in which the delivered result is normalized.

      The option selected in the ESA/390 BFP architecture (and the RS/6000) is to detect "loss of accuracy" as an inexact result.

   d. Although the standard does not require traps to be implemented for underflow or the other arithmetic exceptions, it does state that "with each exception should be associated a trap under user control." Since it also defines "should" as "that which is strongly recommended as being in keeping with the intent of the standard," the ESA/390 BFP architecture provides traps by means of program interruptions.

   e. When the underflow trap is enabled, underflow is to be signaled when tininess is detected regardless of loss of accuracy. When the underflow trap is not enabled, the underflow flag bit is to be set only when both tininess and loss of accuracy have been detected. Add and subtract can result in tiny or inexact results, but not both. Thus, when underflow is disabled, add and subtract never set the underflow flag bit.

## Result Figures

Concise descriptions of the results produced by many of the BFP instructions are made by means of figures which contain columns and rows representing all possible combinations of BFP data class for the source operands of an instruction. The information shown at the intersection of a row and a column is one or more symbols representing the result or results produced for that particular combi-


LzLabs GmbH. Ex. 1008-813

nation of source-operand data classes. Explanations of the symbols used are contained in each figure. In many cases, the explanation of a particular result is in the form of a cross reference to another figure. In many cases, the information shown at the intersection consists of several symbols separated by commas. All such results are produced unless one of the results is a program interruption. In the case of a program interruption, the operation is suppressed or completed as shown in Figure 19-13 on page 19-15.

## Data-Exception Codes (DXC) and Abbreviations

Figure 19-12 shows IEEE exception-condition and flag abbreviations that are used in the result figures, and it explains the symbols "Xi:" and "Xz:" that are used in the figures.

| Exception Condition | | FPC IEEE Mask Bit | IEEE Flag | |
|---|---|---|---|---|
| Name | Abbr. | | FPC Bit | Abbr. |
| IEEE invalid operation | Xi[1] | 0.0 | 1.0 | SFi |
| IEEE division by zero | Xz[2] | 0.1 | 1.1 | SFz |
| IEEE overflow | Xo | 0.2 | 1.2 | SFo |
| IEEE underflow | Xu | 0.3 | 1.3 | SFu |
| IEEE inexact | Xx | 0.4 | 1.4 | SFx |

**Explanation:**

1. The symbol "Xi:" followed by a list of results in a figure indicates that, when FPC 0.0 is zero, then instruction execution is completed by setting SFi (FPC 1.0) to one and producing the indicated results; and when FPC 0.0 is one, then instruction execution is suppressed, the data exception code (DXC) is set to 80 hex, and a program interruption for a data exception occurs.
2. The symbol "Xz:" followed by a list of results in a figure indicates that, when FPC 0.1 is zero, then instruction execution is completed by setting SFz (FPC 1.1) to one and producing the indicated results; and when FPC 0.1 is one, then instruction execution is suppressed, the data exception code (DXC) is set to 40 hex, and a program interruption for a data exception occurs.

*Figure 19-12. IEEE Exception-Condition and Flag Abbreviations*

Bits 0-4 (i,z,o,u,x) of the eight-bit data-exception code (DXC) in byte 2 of the FPC register are trap flags and correspond to the same bits in bytes 0 and 1 of the register (IEEE masks and IEEE flags). The trap flag for an exception, instead of the IEEE flag, is set to one when an interruption for the exception is enabled by the corresponding IEEE mask bit. Bit 5 of byte 2 (y) is used in conjunction with bit 4, inexact (x), to indicate that the result has been incremented in magnitude.

Figure 19-13 on page 19-15 shows the various DXCs that can be indicated, the associated instruction endings, and abbreviations that are used for the DXCs in the result figures. (The abbreviation "PID" stands for "program interruption for a data exception.")



LzLabs GmbH. Ex. 1008-814

| Abbr. | DXC (Hex) | Data-Exception-Code Name | Instruction Ending |
|---|---|---|---|
| PIDx | 08 | IEEE inexact and truncated | Complete |
| PIDy | 0C | IEEE inexact and incremented | Complete |
| PIDu | 10 | IEEE underflow, exact | Complete, wrap exponent |
| PIDux | 18 | IEEE underflow, inexact and truncated | Complete, wrap exponent |
| PIDuy | 1C | IEEE underflow, inexact and incremented | Complete, wrap exponent |
| PIDo | 20 | IEEE overflow, exact | Complete, wrap exponent |
| PIDox | 28 | IEEE overflow, inexact and truncated | Complete, wrap exponent |
| PIDoy | 2C | IEEE overflow, inexact and incremented | Complete, wrap exponent |
| PIDz | 40 | IEEE division by zero | Suppress |
| PIDi | 80 | IEEE invalid operation | Suppress |

*Figure 19-13. IEEE Data-Exception Codes (DXC) and Abbreviations*

## Instructions

The BFP instructions and their mnemonics and operation codes are listed in the figure "Summary of BFP Instructions." The figure indicates, in the column labeled "Characteristics," the instruction format, when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

All BFP instructions are subject to the AFP-register-control bit, bit 13 of control register 0. For the BFP instructions to be executed successfully, the AFP-register-control bit must be one; otherwise, a BFP-instruction data exception, DXC 2, is recognized. An operation exception is recognized when the CPU attempts to execute a BFP instruction when the BFP facility is not installed.

Mnemonics for the BFP instructions are distinguished from the corresponding HFP instructions by a B in the mnemonic. Mnemonics for the BFP instructions have an R as the last letter when the instruction is in the RRE or RRF format. Certain letters are used for BFP instructions to represent operand-format length, as follows:

F Thirty-two-bit fixed point
D Long
E Short
X Extended

**Note:** In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For a register-to-register operation using COMPARE (short), for example, CEBR is the mnemonic and $R_1$,$R_2$ the operand designation.

**Programming Note:** All of the instructions shown in Figure 19-14 on page 19-16 are available when the BFP facility is installed.


LzLabs GmbH. Ex. 1008-815

| Name | Mne-monic | Characteristics | | | | | | | | | | Op Code |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD (extended BFP) | AXBR | RRE | C | | BF | | SP | Db | Xi | Xo Xu Xx | | B34A |
| ADD (long BFP) | ADBR | RRE | C | | BF | | | Db | Xi | Xo Xu Xx | | B31A |
| ADD (long BFP) | ADB | RXE | C | | BF | A | | Db | Xi | Xo Xu Xx | $B_2$ | ED1A |
| ADD (short BFP) | AEBR | RRE | C | | BF | | | Db | Xi | Xo Xu Xx | | B30A |
| ADD (short BFP) | AEB | RXE | C | | BF | A | | Db | Xi | Xo Xu Xx | $B_2$ | ED0A |
| COMPARE (extended BFP) | CXBR | RRE | C | | BF | | SP | Db | Xi | | | B349 |
| COMPARE (long BFP) | CDBR | RRE | C | | BF | | | Db | Xi | | | B319 |
| COMPARE (long BFP) | CDB | RXE | C | | BF | A | | Db | Xi | | $B_2$ | ED19 |
| COMPARE (short BFP) | CEBR | RRE | C | | BF | | | Db | Xi | | | B309 |
| COMPARE (short BFP) | CEB | RXE | C | | BF | A | | Db | Xi | | $B_2$ | ED09 |
| COMPARE AND SIGNAL (extended BFP) | KXBR | RRE | C | | BF | | SP | Db | Xi | | | B348 |
| COMPARE AND SIGNAL (long BFP) | KDBR | RRE | C | | BF | | | Db | Xi | | | B318 |
| COMPARE AND SIGNAL (long BFP) | KDB | RXE | C | | BF | A | | Db | Xi | | $B_2$ | ED18 |
| COMPARE AND SIGNAL (short BFP) | KEBR | RRE | C | | BF | | | Db | Xi | | | B308 |
| COMPARE AND SIGNAL (short BFP) | KEB | RXE | C | | BF | A | | Db | Xi | | $B_2$ | ED08 |
| CONVERT FROM FIXED (32 to ext. BFP) | CXFBR | RRE | | | BF | | SP | Db | | | | B396 |
| CONVERT FROM FIXED (32 to long BFP) | CDFBR | RRE | | | BF | | | Db | | | | B395 |
| CONVERT FROM FIXED (32 to short BFP) | CEFBR | RRE | | | BF | | | Db | | Xx | | B394 |
| CONVERT TO FIXED (ext. BFP to 32) | CFXBR | RRF | C | | BF | | SP | Db | Xi | Xx | R | B39A |
| CONVERT TO FIXED (long BFP to 32) | CFDBR | RRF | C | | BF | | SP | Db | Xi | Xx | R | B399 |
| CONVERT TO FIXED (short BFP to 32) | CFEBR | RRF | C | | BF | | SP | Db | Xi | Xx | R | B398 |
| DIVIDE (extended BFP) | DXBR | RRE | | | BF | | SP | Db | Xi | Xz Xo Xu Xx | | B34D |
| DIVIDE (long BFP) | DDBR | RRE | | | BF | | | Db | Xi | Xz Xo Xu Xx | | B31D |
| DIVIDE (long BFP) | DDB | RXE | | | BF | A | | Db | Xi | Xz Xo Xu Xx | $B_2$ | ED1D |
| DIVIDE (short BFP) | DEBR | RRE | | | BF | | | Db | Xi | Xz Xo Xu Xx | | B30D |
| DIVIDE (short BFP) | DEB | RXE | | | BF | A | | Db | Xi | Xz Xo Xu Xx | $B_2$ | ED0D |
| DIVIDE TO INTEGER (long BFP) | DIDBR | RRF | C | | BF | | SP | Db | Xi | Xu Xx | | B35B |
| DIVIDE TO INTEGER (short BFP) | DIEBR | RRF | C | | BF | | SP | Db | Xi | Xu Xx | | B353 |
| EXTRACT FPC | EFPC | RRE | | | BF | | | Db | | | | B38C |
| LOAD AND TEST (extended BFP) | LTXBR | RRE | C | | BF | | SP | Db | Xi | | | B342 |
| LOAD AND TEST (long BFP) | LTDBR | RRE | C | | BF | | | Db | Xi | | | B312 |
| LOAD AND TEST (short BFP) | LTEBR | RRE | C | | BF | | | Db | Xi | | | B302 |
| LOAD COMPLEMENT (extended BFP) | LCXBR | RRE | C | | BF | | SP | Db | | | | B343 |
| LOAD COMPLEMENT (long BFP) | LCDBR | RRE | C | | BF | | | Db | | | | B313 |
| LOAD COMPLEMENT (short BFP) | LCEBR | RRE | C | | BF | | | Db | | | | B303 |
| LOAD FP INTEGER (extended BFP) | FIXBR | RRF | | | BF | | SP | Db | Xi | Xx | | B347 |
| LOAD FP INTEGER (long BFP) | FIDBR | RRF | | | BF | | SP | Db | Xi | Xx | | B35F |
| LOAD FP INTEGER (short BFP) | FIEBR | RRF | | | BF | | SP | Db | Xi | Xx | | B357 |
| LOAD FPC | LFPC | S | | | BF | A | SP | Db | | | $B_2$ | B29D |
| LOAD LENGTHENED (long to ext. BFP) | LXDBR | RRE | | | BF | | SP | Db | Xi | | | B305 |

*Figure 19-14 (Part 1 of 3). Summary of BFP Instructions*



LzLabs GmbH. Ex. 1008-816

| Name | Mne-monic | Characteristics | | | | | | Op Code |
|---|---|---|---|---|---|---|---|---|
| LOAD LENGTHENED (long to ext. BFP) | LXDB | RXE | BF | A SP | Db Xi | | $B_2$ | ED05 |
| LOAD LENGTHENED (short to ext. BFP) | LXEBR | RRE | BF | SP | Db Xi | | | B306 |
| LOAD LENGTHENED (short to ext. BFP) | LXEB | RXE | BF | A SP | Db Xi | | $B_2$ | ED06 |
| LOAD LENGTHENED (short to long BFP) | LDEBR | RRE | BF | | Db Xi | | | B304 |
| LOAD LENGTHENED (short to long BFP) | LDEB | RXE | BF | A | Db Xi | | $B_2$ | ED04 |
| LOAD NEGATIVE (extended BFP) | LNXBR | RRE C | BF | SP | Db | | | B341 |
| LOAD NEGATIVE (long BFP) | LNDBR | RRE C | BF | | Db | | | B311 |
| LOAD NEGATIVE (short BFP) | LNEBR | RRE C | BF | | Db | | | B301 |
| LOAD POSITIVE (extended BFP) | LPXBR | RRE C | BF | SP | Db | | | B340 |
| LOAD POSITIVE (long BFP) | LPDBR | RRE C | BF | | Db | | | B310 |
| LOAD POSITIVE (short BFP) | LPEBR | RRE C | BF | | Db | | | B300 |
| LOAD ROUNDED (extended to long BFP) | LDXBR | RRE | BF | SP | Db Xi Xo Xu Xx | | | B345 |
| LOAD ROUNDED (extended to short BFP) | LEXBR | RRE | BF | SP | Db Xi Xo Xu Xx | | | B346 |
| LOAD ROUNDED (long to short BFP) | LEDBR | RRE | BF | | Db Xi Xo Xu Xx | | | B344 |
| MULTIPLY (extended BFP) | MXBR | RRE | BF | SP | Db Xi Xo Xu Xx | | | B34C |
| MULTIPLY (long BFP) | MDBR | RRE | BF | | Db Xi Xo Xu Xx | | | B31C |
| MULTIPLY (long BFP) | MDB | RXE | BF | A | Db Xi Xo Xu Xx | | $B_2$ | ED1C |
| MULTIPLY (long to extended BFP) | MXDBR | RRE | BF | SP | Db Xi | | | B307 |
| MULTIPLY (long to extended BFP) | MXDB | RXE | BF | A SP | Db Xi | | $B_2$ | ED07 |
| MULTIPLY (short BFP) | MEEBR | RRE | BF | | Db Xi Xo Xu Xx | | | B317 |
| MULTIPLY (short BFP) | MEEB | RXE | BF | A | Db Xi Xo Xu Xx | | $B_2$ | ED17 |
| MULTIPLY (short to long BFP) | MDEBR | RRE | BF | | Db Xi | | | B30C |
| MULTIPLY (short to long BFP) | MDEB | RXE | BF | A | Db Xi | | $B_2$ | ED0C |
| MULTIPLY AND ADD (long BFP) | MADBR | RRF | BF | | Db Xi Xo Xu Xx | | | B31E |
| MULTIPLY AND ADD (long BFP) | MADB | RXF | BF | A | Db Xi Xo Xu Xx | | $B_2$ | ED1E |
| MULTIPLY AND ADD (short BFP) | MAEBR | RRF | BF | | Db Xi Xo Xu Xx | | | B30E |
| MULTIPLY AND ADD (short BFP) | MAEB | RXF | BF | A | Db Xi Xo Xu Xx | | $B_2$ | ED0E |
| MULTIPLY AND SUBTRACT (long BFP) | MSDBR | RRF | BF | | Db Xi Xo Xu Xx | | | B31F |
| MULTIPLY AND SUBTRACT (long BFP) | MSDB | RXF | BF | A | Db Xi Xo Xu Xx | | $B_2$ | ED1F |
| MULTIPLY AND SUBTRACT (short BFP) | MSEBR | RRF | BF | | Db Xi Xo Xu Xx | | | B30F |
| MULTIPLY AND SUBTRACT (short BFP) | MSEB | RXF | BF | A | Db Xi Xo Xu Xx | | $B_2$ | ED0F |
| SET FPC | SFPC | RRE | BF | SP | Db | | | B384 |
| SET ROUNDING MODE | SRNM | S | BF | | Db | | | B299 |
| SQUARE ROOT (extended BFP) | SQXBR | RRE | BF | SP | Db Xi Xx | | | B316 |
| SQUARE ROOT (long BFP) | SQDBR | RRE | BF | | Db Xi Xx | | | B315 |
| SQUARE ROOT (long BFP) | SQDB | RXE | BF | A | Db Xi Xx | | $B_2$ | ED15 |
| SQUARE ROOT (short BFP) | SQEBR | RRE | BF | | Db Xi Xx | | | B314 |
| SQUARE ROOT (short BFP) | SQEB | RXE | BF | A | Db Xi Xx | | $B_2$ | ED14 |
| STORE FPC | STFPC | S | BF | A | Db | ST | $B_2$ | B29C |
| SUBTRACT (extended BFP) | SXBR | RRE C | BF | SP | Db Xi Xo Xu Xx | | | B34B |

*Figure 19-14 (Part 2 of 3). Summary of BFP Instructions*


LzLabs GmbH. Ex. 1008-817

| Name | Mne-monic | Characteristics | | | | | Op Code |
|---|---|---|---|---|---|---|---|
| SUBTRACT (long BFP) | SDBR | RRE C BF | | Db Xi Xo Xu Xx | | | B31B |
| SUBTRACT (long BFP) | SDB | RXE C BF | A | Db Xi Xo Xu Xx | | $B_2$ | ED1B |
| SUBTRACT (short BFP) | SEBR | RRE C BF | | Db Xi Xo Xu Xx | | | B30B |
| SUBTRACT (short BFP) | SEB | RXE C BF | A | Db Xi Xo Xu Xx | | $B_2$ | ED0B |
| TEST DATA CLASS (extended BFP) | TCXB | RXE C BF | SP | Db | | | ED12 |
| TEST DATA CLASS (long BFP) | TCDB | RXE C BF | | Db | | | ED11 |
| TEST DATA CLASS (short BFP) | TCEB | RXE C BF | | Db | | | ED10 |

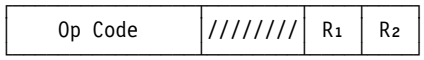
**Explanation:**

A Access exceptions for logical addresses.
$B_2$ $B_2$ field designates an access register in the access-register mode.
BF BFP facility.
C Condition code is set.
Db BFP-instruction data exception.
R PER general-register-alteration event.
RRE RRE instruction format.
RRF RRF instruction format.
RXE RXE instruction format.
RXF RXF instruction format.
SP Specification exception.
ST PER storage-alteration event.
Xi IEEE invalid-operation condition.
Xo IEEE overflow condition.
Xu IEEE underflow condition.
Xx IEEE inexact condition.
Xz IEEE division-by-zero condition.

*Figure 19-14 (Part 3 of 3). Summary of BFP Instructions*

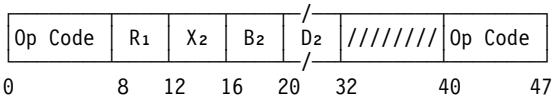
## ADD

Mnemonic1 $R_1$,$R_2$ [RRE]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| AEBR | 'B30A' | Short BFP |
| ADBR | 'B31A' | Long BFP |
| AXBR | 'B34A' | Extended BFP |

Mnemonic2 $R_1$,$D_2$($X_2$,$B_2$) [RXE]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| AEB | 'ED0A' | Short BFP |
| ADB | 'ED1A' | Long BFP |

The second operand is added to the first operand, and the sum is placed at the first-operand location.

If both operands are numeric and finite, they are added algebraically, forming an intermediate sum. The intermediate sum, if nonzero, is normalized and rounded to the operand format according to the current rounding mode. The sum is then placed at the result location.

The sign of the sum is determined by the rules of algebra. This also applies to a result of zero:

- If the result of rounding a nonzero intermediate sum is zero, the sign of the zero result is the sign of the intermediate sum.
- If the sum of two operands with opposite signs is exactly zero, the sign of the result is plus in all rounding modes except round toward $-\infty$, in which mode the sign is minus.
- The sign of the sum x plus x is the sign of x, even when x is zero.



LzLabs GmbH. Ex. 1008-818

If one operand is an infinity and the other is finite and numeric, the result is that infinity. If both operands are infinities of the same sign, the result is the same infinity. If the two operands are infinities of opposite signs, an IEEE-invalid-operation condition is recognized.

See Figure 19-16 on page 19-20 for a detailed description of the results of this instruction. (Figure 19-15 is referred to by Figure 19-16.)

For AXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Result is a NaN

***IEEE Exception Conditions:***

- Invalid operation
- Overflow
- Underflow
- Inexact

***Program Exceptions:***

- Access (fetch, operand 2 of AEB and ADB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (AXBR only)

**Programming Note:** Interchanging the two operands in a BFP addition does not affect the value of the sum when the result is numeric. This is not true, however, when both operands are QNaNs, in which case the result is the first operand; or when both operands are SNaNs and the IEEE-invalid-operation mask bit in the FPC register is zero, in which case the result is the QNaN derived from the first operand.

| Value of Result (r) | Condition Code |
|---|---|
| r=0 | cc0 |
| r<0 | cc1 |
| r>0 | cc2 |
| **Explanation:** ccn Condition code is set to n. | |

*Figure 19-15. Condition Code for Resultant Sum*


LzLabs GmbH. Ex. 1008-819

| First Operand (a) Is | Results for ADD (a+b) when Second Operand (b) Is | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | −∞ | −Nn | −Dn | −0 | +0 | +Dn | +Nn | +∞ | QNaN | SNaN |
| −∞ | T(−∞), cc1 | T(−∞), cc1 | T(−∞), cc1 | T(−∞), cc1 | T(−∞), cc1 | T(−∞), cc1 | T(−∞), cc1 | Xi: T(dNaN), cc3 | T(b), cc3 | Xi: T(b*), cc3 |
| −Nn | T(−∞), cc1 | R(a+b), cc1 | R(a+b), cc1 | T(a), cc1 | T(a), cc1 | R(a+b), cc1 | R(a+b), ccrs | T(+∞), cc2 | T(b), cc3 | Xi: T(b*), cc3 |
| −Dn | T(−∞), cc1 | R(a+b), cc1 | R(a+b), cc1 | R(a), cc1 | R(a), cc1 | R(a+b), ccrs | R(a+b), cc2 | T(+∞), cc2 | T(b), cc3 | Xi: T(b*), cc3 |
| −0 | T(−∞), cc1 | T(b), cc1 | R(b), cc1 | T(−0), cc0 | Rezd, cc0 | R(b), cc2 | T(b), cc2 | T(+∞), cc2 | T(b), cc3 | Xi: T(b*), cc3 |
| +0 | T(−∞), cc1 | T(b), cc1 | R(b), cc1 | Rezd, cc0 | T(+0), cc0 | R(b), cc2 | T(b), cc2 | T(+∞), cc2 | T(b), cc3 | Xi: T(b*), cc3 |
| +Dn | T(−∞), cc1 | R(a+b), cc1 | R(a+b), ccrs | R(a), cc2 | R(a), cc2 | R(a+b), cc2 | R(a+b), cc2 | T(+∞), cc2 | T(b), cc3 | Xi: T(b*), cc3 |
| +Nn | T(−∞), cc1 | R(a+b), ccrs | R(a+b), cc2 | T(a), cc2 | T(a), cc2 | R(a+b), cc2 | R(a+b), cc2 | T(+∞), cc2 | T(b), cc3 | Xi: T(b*), cc3 |
| +∞ | Xi: T(dNaN), cc3 | T(+∞), cc2 | T(+∞), cc2 | T(+∞), cc2 | T(+∞), cc2 | T(+∞), cc2 | T(+∞), cc2 | T(+∞), cc2 | T(b), cc3 | Xi: T(b*), cc3 |
| QNaN | T(a), cc3 | T(a), cc3 | T(a), cc3 | T(a), cc3 | T(a), cc3 | T(a), cc3 | T(a), cc3 | T(a), cc3 | T(a), cc3 | Xi: T(b*), cc3 |
| SNaN | Xi: T(a*), cc3 | Xi: T(a*), cc3 | Xi: T(a*), cc3 | Xi: T(a*), cc3 | Xi: T(a*), cc3 | Xi: T(a*), cc3 | Xi: T(a*), cc3 | Xi: T(a*), cc3 | Xi: T(a*), cc3 | Xi: T(a*), cc3 |

**Explanation:**

| | |
|---|---|
| * | The SNaN is converted to the corresponding QNaN before it is placed at the target operand location. |
| ccn | Condition code is set to n. |
| ccrs | Condition code is set according to the resultant sum. See Figure 19-15 on page 19-19. |
| dNaN | Default quiet NaN. |
| Dn | Denormalized number. |
| Nn | Normalized nonzero number. |
| R(v) | Rounding and range action is performed on the value v. See Figure 19-17 on page 19-21. |
| Rezd | Exact zero-difference result. See Figure 19-17 on page 19-21. |
| T(x) | The value x is placed at the target operand location. |
| Xi: | IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero. |

*Figure 19-16. Results: ADD*



LzLabs GmbH. Ex. 1008-820

| Range of v | Case | Normal Result (r) when Rounding Mode Is | | | |
|---|---|---|---|---|---|
| | | To Nearest | Toward 0 | Toward +∞ | Toward −∞ |
| v < −Nmax, p < −Nmax | Overflow | −∞[1] | −Nmax | −Nmax | −∞[1] |
| v < −Nmax, p = −Nmax | Normal | −Nmax | −Nmax | −Nmax | – |
| −Nmax ≤ v ≤ −Nmin | Normal | p | p | p | p |
| −Nmin < v ≤ −Dmin | Tiny | d* | d | d | d* |
| −Dmin < v < −Dmin/2 | Tiny | −Dmin | −0 | −0 | −Dmin |
| −Dmin/2 ≤ v < 0 | Tiny | −0 | −0 | −0 | −Dmin |
| v = 0 | Exact zero difference[2] | +0 | +0 | +0 | −0 |
| 0 < v ≤ +Dmin/2 | Tiny | +0 | +0 | +Dmin | +0 |
| +Dmin/2 < v < +Dmin | Tiny | +Dmin | +0 | +Dmin | +0 |
| +Dmin ≤ v < +Nmin | Tiny | d* | d | d* | d |
| +Nmin ≤ v ≤ +Nmax | Normal | p | p | p | p |
| +Nmax < v, p = +Nmax | Normal | +Nmax | +Nmax | – | +Nmax |
| +Nmax < v, +Nmax < p | Overflow | +∞[1] | +Nmax | +∞[1] | +Nmax |

**Explanation:**

– This situation cannot occur.

* The rounded value, in the extreme case, may be Nmin. In this case, the exception conditions are underflow, inexact and incremented.

[1] The normal result r is considered to have been incremented.

[2] The exact-zero-difference case applies only to ADD, SUBTRACT, MULTIPLY AND ADD, and MULTIPLY AND SUBTRACT. For all other operations, a zero result is detected by inspection of the source operands without use of the R(v) function.

d The value derived when the exact result v is rounded to the format of the target, including both precision and bounded exponent range. Except as explained in note *, this is a denormalized number.

p The value derived when the exact result v is rounded to the precision of the target, but assuming an unbounded exponent range.

v Exact result before rounding, assuming unbounded precision and an unbounded exponent range. For LOAD ROUNDED, v is the source value a.

Dmin Smallest (in magnitude) representable denormalized number in the target format.

Nmax Largest (in magnitude) representable finite number in the target format.

Nmin Smallest (in magnitude) representable normalized number in the target format.

*Figure 19-17 (Part 1 of 2). Action for R(v): Rounding and Range Function*



LzLabs GmbH. Ex. 1008-821

| Case | Is r Inexact (r≠v) | Overflow Mask (FPC 0.2) | Underflow Mask (FPC 0.3) | Inexact Mask (FPC 0.4) | Is r Incremented (\|r\|>\|v\|) | Is p Inexact (p≠v) | Is p Incremented (\|p\|>\|v\|) | Results |
|---|---|---|---|---|---|---|---|---|
| Overflow | Yes[1] | 0 | – | 0 | – | – | – | T(r), SFo←1, SFx←1 |
| Overflow | Yes[1] | 0 | – | 1 | No | – | – | T(r), SFo←1, PIDx(08) |
| Overflow | Yes[1] | 0 | – | 1 | Yes | – | – | T(r), SFo←1, PIDy(0C) |
| Overflow | Yes[1] | 1 | – | – | – | No | No[1] | Tw(p÷β), PIDo(20) |
| Overflow | Yes[1] | 1 | – | – | – | Yes | No | Tw(p÷β), PIDox(28) |
| Overflow | Yes[1] | 1 | – | – | – | Yes | Yes | Tw(p÷β), PIDoy(2C) |
| Normal | No | – | – | – | – | – | – | T(r) |
| Normal | Yes | – | – | 0 | – | – | – | T(r), SFx←1 |
| Normal | Yes | – | – | 1 | No | – | – | T(r), PIDx(08) |
| Normal | Yes | – | – | 1 | Yes | – | – | T(r), PIDy(0C) |
| Tiny | No | – | 0 | – | – | – | – | T(r) |
| Tiny | No | – | 1 | – | – | No[1] | No[1] | Tw(p·β), PIDu(10) |
| Tiny | Yes | – | 0 | 0 | – | – | – | T(r), SFu←1, SFx←1 |
| Tiny | Yes | – | 0 | 1 | No | – | – | T(r), SFu←1, PIDx(08) |
| Tiny | Yes | – | 0 | 1 | Yes | – | – | T(r), SFu←1, PIDy(0C) |
| Tiny | Yes | – | 1 | – | – | No | No[1] | Tw(p·β), PIDu(10) |
| Tiny | Yes | – | 1 | – | – | Yes | No | Tw(p·β), PIDux(18) |
| Tiny | Yes | – | 1 | – | – | Yes | Yes | Tw(p·β), PIDuy(1C) |

**Explanation:**

| | |
|---|---|
| – | The results do not depend on this condition or mask bit. |
| [1] | This condition is true by virtue of the state of some condition to the left of this column. |
| β | Wrap adjust, which depends on the type of operation and operand format. For all operations except LOAD ROUNDED, the wrap adjust depends on the target format: $\beta = 2^{\alpha}$, where $\alpha$ is 192 for short, 1536 for long, and 24576 for extended. For LOAD ROUNDED, the wrap adjust depends on the source format: $\beta = 2^{\kappa}$, where $\kappa$ is 512 for long and 8192 for extended. |
| p | The value derived when the exact result v is rounded to the precision of the target, but assuming an unbounded exponent range. |
| r | Normal result as defined in Part 1 of this figure. |
| v | Exact result before rounding, assuming unbounded precision and unbounded exponent range. |
| PIDc(h) | Program interruption for data exception, condition c, with DXC of h in hex. See Figure 19-13 on page 19-15. |
| SFo | IEEE overflow flag, FPC 1.2. |
| SFu | IEEE underflow flag, FPC 1.3. |
| SFx | IEEE inexact flag, FPC 1.4. |
| T(x) | The value x is placed at the target operand location. |
| Tw(x) | The wrapped result x is placed at the target operand location. For all operations except LOAD ROUNDED, the wrapped result is in the same format and length as normal results at the target location. For LOAD ROUNDED, the wrapped result is in the same format and length as the source, but rounded to the precision of the target. |

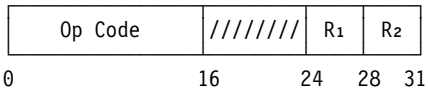
*Figure 19-17 (Part 2 of 2). Action for R(v): Rounding and Range Function*



LzLabs GmbH. Ex. 1008-822

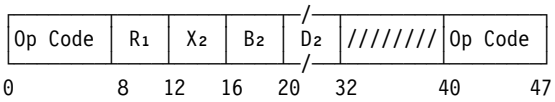
## COMPARE

Mnemonic1 $R_1,R_2$ [RRE]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| CEBR | 'B309' | Short BFP |
| CDBR | 'B319' | Long BFP |
| CXBR | 'B349' | Extended BFP |

Mnemonic2 $R_1,D_2(X_2,B_2)$ [RXE]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| CEB | 'ED09' | Short BFP |
| CDB | 'ED19' | Long BFP |

The first operand is compared with the second operand, and the condition code is set to indicate the result.

If both operands are numeric and finite, the comparison is algebraic and follows the procedure for BFP subtraction, except that the difference is discarded after setting the condition code, and both operands remain unchanged. If the difference is exactly zero with either sign, the operands are equal; this includes zero operands (so +0 equals −0). If a nonzero difference is positive or negative, the first operand is high or low, respectively.

+∞ compares greater than any finite number, and all finite numbers compare greater than −∞. Two infinity operands of like sign compare equal.

Numeric comparison is exact, and the condition code is determined for finite operands as if range and precision were unlimited. No overflow or underflow condition can occur.

If either or both operands are QNaNs and neither operand is an SNaN, the comparison result is unordered, and condition code 3 is set.

If either or both operands are SNaNs, an IEEE-invalid-operation condition is recognized. If the IEEE invalid-operation mask bit is one, a program interruption for a data exception with DXC 80 hex (IEEE invalid operation) occurs. If the IEEE-invalid-operation mask bit is zero, the IEEE-invalid-operation flag bit is set to one, and instruction execution is completed by setting condition code 3.

See Figure 19-18 on page 19-24 for a detailed description of the results of this instruction.

For CXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Operands equal
1 First operand low
2 First operand high
3 Operands unordered

***IEEE Exception Conditions:***

- Invalid operation

***Program Exceptions:***

- Access (fetch, operand 2 of CEB and CDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (CXBR only)

**Programming Notes:**

1. COMPARE may be used by a compiler to implement those comparisons which are required by the IEEE standard to not recognize an exception condition when the result is unordered due to a QNaN.
2. The IEEE standard requires that it be possible to compare BFP operands in different formats. To accomplish this, LOAD LENGTHENED may be used before COMPARE to convert the shorter operand to the same format as the longer.


LzLabs GmbH. Ex. 1008-823

| First Operand (a) Is | Results for COMPARE (a:b) when Second Operand (b) Is | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | −∞ | −Fn | −0 | +0 | +Fn | +∞ | QNaN | SNaN |
| −∞ | cc0 | cc1 | cc1 | cc1 | cc1 | cc1 | cc3 | Xi: cc3 |
| −Fn | cc2 | C(a:b) | cc1 | cc1 | cc1 | cc1 | cc3 | Xi: cc3 |
| −0 | cc2 | cc2 | cc0 | cc0 | cc1 | cc1 | cc3 | Xi: cc3 |
| +0 | cc2 | cc2 | cc0 | cc0 | cc1 | cc1 | cc3 | Xi: cc3 |
| +Fn | cc2 | cc2 | cc2 | cc2 | C(a:b) | cc1 | cc3 | Xi: cc3 |
| +∞ | cc2 | cc2 | cc2 | cc2 | cc2 | cc0 | cc3 | Xi: cc3 |
| QNaN | cc3 | cc3 | cc3 | cc3 | cc3 | cc3 | cc3 | Xi: cc3 |
| SNaN | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 |

**Explanation:**

ccn Condition code is set to n.
C(a:b) Basic compare results. See Figure 19-19.
Fn Finite nonzero number (includes both denormalized and normalized).
Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

*Figure 19-18. Results: COMPARE*

| Relation of Value (a) to Value (b) | Condition Code for C(a:b) |
|---|---|
| a=b | cc0 |
| a<b | cc1 |
| a>b | cc2 |

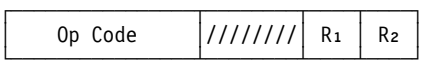
**Explanation:**

ccn Condition code is set to n.

*Figure 19-19. Basic Compare Results*

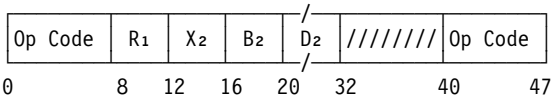
## COMPARE AND SIGNAL

```
Mnemonic1    R1,R2              [RRE]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| KEBR | 'B308' | Short BFP |
| KDBR | 'B318' | Long BFP |
| KXBR | 'B348' | Extended BFP |

```
Mnemonic2    R1,D2(X2,B2)       [RXE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| KEB | 'ED08' | Short BFP |
| KDB | 'ED18' | Long BFP |

The first operand is compared with the second operand, and the condition code is set to indicate the result. The operation is the same as for COMPARE except that QNaN operands cause an IEEE-invalid-operation condition to be recognized. Thus, QNaN operands are treated as if they were SNaNs.

See Figure 19-20 on page 19-25 for a detailed description of the results of this instruction.

For KXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Operands equal
1 First operand low
2 First operand high
3 Operands unordered



LzLabs GmbH. Ex. 1008-824

***IEEE Exception Conditions:***

- Invalid operation

***Program Exceptions:***

- Access (fetch, operand 2 of KEB and KDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (KXBR only)

**Programming Notes:**

1. COMPARE AND SIGNAL may be used by a compiler to implement those comparisons which are required by the IEEE standard to recognize an exception condition when the result is unordered due to a QNaN.
2. The IEEE standard requires that it be possible to compare BFP operands in different formats. To accomplish this, LOAD LENGTHENED may be used before COMPARE AND SIGNAL to convert the shorter operand to the same format as the longer.

| First Operand (a) Is | Results for COMPARE AND SIGNAL (a:b) when Second Operand (b) Is | | | | | | |
|---|---|---|---|---|---|---|---|
| | −∞ | −Fn | −0 | +0 | +Fn | +∞ | NaN |
| −∞ | cc0 | cc1 | cc1 | cc1 | cc1 | cc1 | Xi: cc3 |
| −Fn | cc2 | C(a:b) | cc1 | cc1 | cc1 | cc1 | Xi: cc3 |
| −0 | cc2 | cc2 | cc0 | cc0 | cc1 | cc1 | Xi: cc3 |
| +0 | cc2 | cc2 | cc0 | cc0 | cc1 | cc1 | Xi: cc3 |
| +Fn | cc2 | cc2 | cc2 | cc2 | C(a:b) | cc1 | Xi: cc3 |
| +∞ | cc2 | cc2 | cc2 | cc2 | cc2 | cc0 | Xi: cc3 |
| NaN | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 | Xi: cc3 |

**Explanation:**

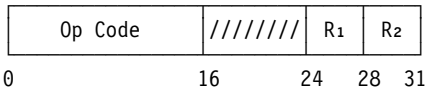
| | |
|---|---|
| ccn | Condition code is set to n. |
| C(a:b) | Basic compare results. See Figure 19-19 on page 19-24. |
| Fn | Finite nonzero number (includes both denormalized and normalized). |
| Xi: | IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero. |

*Figure 19-20. Results: COMPARE AND SIGNAL*


LzLabs GmbH. Ex. 1008-825

## CONVERT FROM FIXED

```
Mnemonic    R1,R2              [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| CEFBR | 'B394' | 32-bit binary-integer operand, short BFP result |
| CDFBR | 'B395' | 32-bit binary-integer operand, long BFP result |
| CXFBR | 'B396' | 32-bit binary-integer operand, extended BFP result |

The fixed-point second operand is converted to the BFP format, and the result is placed at the first-operand location.

The second operand is a 32-bit signed binary integer that is located in the general register designated by $R_2$.

The result is rounded according to the current rounding mode before it is placed at the first-operand location.

See Figure 19-21 for a detailed description of the results of this instruction.

For CXFBR, the $R_1$ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***IEEE Exception Conditions:***

- Inexact (CEFBR only)

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (CXFBR only)

| Instruction | Results for Instructions with a Single Operand (a) when Operand (a) Is | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | −∞ | −Fn | −0 | +0 | +Fn | +∞ | QNaN | SNaN |
| CONVERT FROM FIXED | – | Rf(a) | – | T(+0) | Rf(a) | – | – | – |
| LOAD AND TEST | T(−∞) | T(a) | T(−0) | T(+0) | T(a) | T(+∞) | T(a) | Xi: T(a*) |
| LOAD LENGTHENED | T(−∞) | T(a)[1] | T(−0) | T(+0) | T(a)[1] | T(+∞) | T(a)[1] | Xi: T(a*)[1] |
| LOAD ROUNDED | T(−∞) | R(a) | T(−0) | T(+0) | R(a) | T(+∞) | T(a)[2] | Xi: T(a*)[2] |
| SQUARE ROOT | Xi: T(dNaN) | Xi: T(dNaN) | T(−0) | T(+0) | R(√ a) | T(+∞) | T(a) | Xi: T(a*) |

**Explanation:**

| | |
|---|---|
| – | This situation cannot occur. |
| * | The SNaN is converted to the corresponding QNaN before it is placed at the target operand location. |
| 1 | The operand is extended to the longer format by appending zeros on the right before it is placed at the target operand location. |
| 2 | The NaN is shortened to the target format by truncating the rightmost bits. |
| dNaN | Default quiet NaN. |
| Fn | Finite nonzero number (includes both denormalized and normalized). |
| R(v) | Rounding and range action is performed on the value v. See Figure 19-17 on page 19-21. |
| Rf(a) | The value a is converted to the exact floating-point number v, and then action R(v) is performed. |
| T(x) | The value x is placed in the target operand location. |
| Xi: | IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero. |

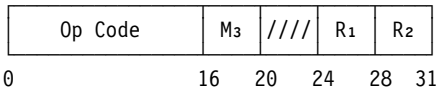
*Figure 19-21. Results: Single-Operand Instructions*



LzLabs GmbH. Ex. 1008-826

## CONVERT TO FIXED

Mnemonic R1,M3,R2 [RRF]




| Mnemonic | Op Code | Operands |
|---|---|---|
| CFEBR | 'B398' | Short BFP operand, 32-bit binary-integer result |
| CFDBR | 'B399' | Long BFP operand, 32-bit binary-integer result |
| CFXBR | 'B39A' | Extended BFP operand, 32-bit binary-integer result |

The BFP second operand is rounded to an integer value and then converted to the fixed-point format. The result is placed at the first-operand location.

The result is a 32-bit signed binary integer that is placed in the general register designated by R1.

If the second operand is numeric and finite, it is rounded to an integer value by rounding as specified by the modifier in the M3 field:

| M3 | Rounding Method |
|---|---|
| 0 | According to current rounding mode |
| 1 | Biased round to nearest |
| 4 | Round to nearest |
| 5 | Round toward 0 |
| 6 | Round toward +∞ |
| 7 | Round toward −∞ |

A modifier other than 0, 1, or 4-7 is invalid.

When the modifier field is zero, rounding is controlled by the current rounding mode specified in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current rounding mode. Rounding for modifiers 4-7 is the same as for rounding modes 0-3 (binary 00-11), respectively. Biased round to nearest (modifier 1) is the same as round to nearest (modifier 4), except when the second operand is exactly halfway between two integers, in which case the result for biased rounding is the next integer that is greater in magnitude.

The sign of the result is the sign of the second operand, except that a zero result has a plus sign.

See Figure 19-22 on page 19-28 for a detailed description of the results of this instruction.

The M3 field must designate a valid modifier; otherwise, a specification exception is recognized. For CFXBR, the R2 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Source was zero
1 Source was less than zero
2 Source was greater than zero
3 Special case

***IEEE Exception Conditions:***

- Invalid operation
- Inexact

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification


LzLabs GmbH. Ex. 1008-827

| Operand (a) | Is n Inexact (n≠a) | Inv.-Op. Mask (FPC 0.0) | Inexact Mask (FPC 0.4) | Is n Incremented (\|n\|>\|a\|) | Results |
|---|---|---|---|---|---|
| −∞ ≤ a < MN, p < MN | – | 0 | 0 | – | T(MN), SFi←1, SFx←1, cc3 |
| −∞ ≤ a < MN, p < MN | – | 0 | 1 | – | T(MN), SFi←1, cc3, PIDx(08) |
| −∞ ≤ a < MN, p < MN | – | 1 | – | – | PIDi(80) |
| −∞ < a < MN, p = MN | – | – | 0 | – | T(MN), SFx←1, cc1 |
| −∞ < a < MN, p = MN | – | – | 1 | – | T(MN), cc1, PIDx(08) |
| MN ≤ a < 0 | No | – | – | – | T(n), cc1 |
| MN ≤ a < 0 | Yes | – | 0 | – | T(n), SFx←1, cc1 |
| MN ≤ a < 0 | Yes | – | 1 | No | T(n), cc1, PIDx(08) |
| MN ≤ a < 0 | Yes | – | 1 | Yes | T(n), cc1, PIDy(0C) |
| −0 | No[1] | – | – | – | T(0), cc0 |
| +0 | No[1] | – | – | – | T(0), cc0 |
| 0 < a ≤ MP | No | – | – | – | T(n), cc2 |
| 0 < a ≤ MP | Yes | – | 0 | – | T(n), SFx←1, cc2 |
| 0 < a ≤ MP | Yes | – | 1 | No | T(n), cc2, PIDx(08) |
| 0 < a ≤ MP | Yes | – | 1 | Yes | T(n), cc2, PIDy(0C) |
| MP < a < +∞, p = MP | – | – | 0 | – | T(MP), SFx←1, cc2 |
| MP < a < +∞, p = MP | – | – | 1 | – | T(MP), cc2, PIDx(08) |
| MP < a ≤ +∞, p > MP | – | 0 | 0 | – | T(MP), SFi←1, SFx←1, cc3 |
| MP < a ≤ +∞, p > MP | – | 0 | 1 | – | T(MP), SFi←1, cc3, PIDx(08) |
| MP < a ≤ +∞, p > MP | – | 1 | – | – | PIDi(80) |
| NaN | – | 0 | 0 | – | T(MN), SFi←1, SFx←1, cc3 |
| NaN | – | 0 | 1 | – | T(MN), SFi←1, cc3, PIDx(08) |
| NaN | – | 1 | – | – | PIDi(80) |

**Explanation:**

| | |
|---|---|
| – | The results do not depend on this condition or mask bit. |
| [1] | This condition is true by virtue of the state of some condition to the left of this column. |
| ccn | Condition code is set to n. |
| n | The value p converted to a fixed-point result. |
| p | The value derived when the source value a is rounded to an integer using the specified rounding mode. |
| MN | Maximum negative number representable in the target fixed-point format. |
| MP | Maximum positive number representable in the target fixed-point format. |
| PIDc(h) | Program interruption for data exception, condition c, with DXC of h in hex. See Figure 19-13 on page 19-15. |
| SFi | IEEE invalid-operation flag, FPC 1.0. |
| SFx | IEEE inexact flag, FPC 1.4. |
| T(x) | The value x is placed at the target operand location. |

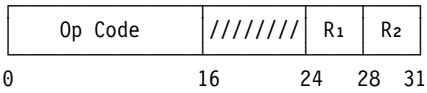
*Figure 19-22. Results: CONVERT TO FIXED*



LzLabs GmbH. Ex. 1008-828

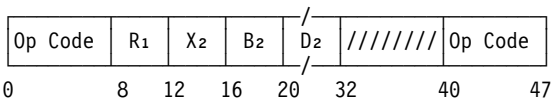
# DIVIDE

Mnemonic1 $R_1,R_2$ [RRE]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| DEBR | 'B30D' | Short BFP |
| DDBR | 'B31D' | Long BFP |
| DXBR | 'B34D' | Extended BFP |

Mnemonic2 $R_1,D_2(X_2,B_2)$ [RXE]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| DEB | 'ED0D' | Short BFP |
| DDB | 'ED1D' | Long BFP |

The first operand (the dividend) is divided by the second operand (the divisor), and the quotient is placed at the first-operand location. No remainder is preserved.

If the divisor is nonzero and both the dividend and divisor are numeric and finite, the first operand is divided by the second operand to form an intermediate quotient. The intermediate quotient, if nonzero, is normalized and rounded to the target format according to the current rounding mode.

The sign of the quotient is the exclusive or of the operand signs. This includes the sign of a zero quotient.

If the divisor is zero but the dividend is nonzero and finite, an IEEE-division-by-zero condition is recognized. If the dividend and divisor are both zero, or if both are infinite, regardless of sign, an IEEE-invalid-operation condition is recognized.

See Figure 19-23 on page 19-30 for a detailed description of the results of this instruction.

For DXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***IEEE Exception Conditions:***

- Invalid operation
- Division by zero
- Overflow
- Underflow
- Inexact

***Program Exceptions:***

- Access (fetch, operand 2 of DEB and DDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (DXBR only)


LzLabs GmbH. Ex. 1008-829

| Dividend (a) | Results for DIVIDE (a÷b) when Divisor (b) Is | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | −∞ | −Fn | −0 | +0 | +Fn | +∞ | QNaN | SNaN |
| −∞ | Xi: T(dNaN) | T(+∞) | T(+∞) | T(−∞) | T(−∞) | Xi: T(dNaN) | T(b) | Xi: T(b*) |
| −Fn | T(+0) | R(a÷b) | Xz: T(+∞) | Xz: T(−∞) | R(a÷b) | T(−0) | T(b) | Xi: T(b*) |
| −0 | T(+0) | T(+0) | Xi: T(dNaN) | Xi: T(dNaN) | T(−0) | T(−0) | T(b) | Xi: T(b*) |
| +0 | T(−0) | T(−0) | Xi: T(dNaN) | Xi: T(dNaN) | T(+0) | T(+0) | T(b) | Xi: T(b*) |
| +Fn | T(−0) | R(a÷b) | Xz: T(−∞) | Xz: T(+∞) | R(a÷b) | T(+0) | T(b) | Xi: T(b*) |
| +∞ | Xi: T(dNaN) | T(−∞) | T(−∞) | T(+∞) | T(+∞) | Xi: T(dNaN) | T(b) | Xi: T(b*) |
| QNaN | T(a) | T(a) | T(a) | T(a) | T(a) | T(a) | T(a) | Xi: T(b*) |
| SNaN | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) |

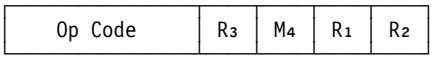
**Explanation:**

| | |
|---|---|
| * | The SNaN is converted to the corresponding QNaN before it is placed at the target operand location. |
| Fn | Finite nonzero number (includes both denormalized and normalized). |
| R(v) | Rounding and range action is performed on the value v. See Figure 19-17 on page 19-21. |
| T(x) | The value x is placed at the target operand location. |
| Xi: | IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero. |
| Xz: | IEEE division-by-zero exception. The results shown are produced only when FPC 0.1 is zero. |

*Figure 19-23. Results: DIVIDE*

## DIVIDE TO INTEGER

```
Mnemonic     R1,R3,R2,M4        [RRF]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| DIEBR | 'B353' | Short BFP |
| DIDBR | 'B35B' | Long BFP |

The first operand (the dividend) is divided by the second operand (the divisor). An integer quotient in BFP form is produced and placed at the third-operand location. The remainder replaces the dividend at the first-operand location. The first, second, and third operands must be in different registers. The condition code indicates whether partial or complete results have been produced and whether the quotient is numeric and finite.

The remainder result is

$r = a - b \cdot n$

where a is the dividend, b the divisor, and n an integer obtained by rounding the precise quotient

$q = a \div b.$

The first-operand result is r with the sign determined by the above expression. The third-operand result is n with a sign that is the exclusive or of the dividend and divisor signs.

If the precise quotient is not an integer and the two integers closest to this precise quotient cannot both be represented exactly in the precision of the quotient, then a partial quotient and partial remainder are formed. This partial quotient n and the corresponding partial remainder

$r = a - b \cdot n$

are used as the results. The sign of a partial remainder is the same as the sign of the dividend. The sign of a partial quotient is the exclusive or of the dividend and divisor signs.



LzLabs GmbH. Ex. 1008-830

If the remainder is zero, then the precise quotient is an integer and can be represented exactly in the precision of the quotient.

The $M_4$ field, called the modifier field, specifies rounding of the final quotient. This rounding is called the "specified quotient rounding mode" as contrasted to the "current rounding mode" specified by the rounding-mode bits in the FPC register. The final quotient is rounded according to the specified quotient rounding mode. The specified quotient rounding mode affects only the final quotient; partial quotients are rounded toward zero.

Since the partial quotient is rounded toward zero, the partial remainder is always exact. For the specified quotient rounding modes of round toward 0, round to nearest, and biased round to nearest, the final remainder is exact. For the specified quotient rounding modes of round toward $+\infty$ and round toward $-\infty$, the final remainder may not be exact.

The final quotient is rounded to an integer by rounding as specified by the modifier in the $M_4$ field:

| $M_4$ | Rounding Method |
|---|---|
| 0 | According to current rounding mode |
| 1 | Biased round to nearest |
| 4 | Round to nearest |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |
| 7 | Round toward $-\infty$ |

A modifier other than 0, 1, or 4-7 is invalid.

When the modifier field is zero, rounding of the final quotient is controlled by the current rounding mode specified in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current rounding mode. Rounding for modifiers 4-7 is the same as for rounding modes 0-3 (binary 00-11), respectively. Biased round to nearest (modifier 1) is the same as round to nearest (modifier 4), except when the final quotient is exactly halfway between two integers, in which case the result for biased rounding is the next integer that is greater in magnitude.

Underflow is recognized only on the final remainder, not on the partial remainder.

For the specified quotient rounding modes of round toward $+\infty$ and round toward $-\infty$, the final remainder may not be exact. When, in these cases, the final remainder is inexact, it is rounded according to the current rounding mode specified in the FPC register.

The sign of a zero quotient is the exclusive or of the divisor and dividend signs.

A zero remainder has the sign of the dividend.

See Figure 19-24 on page 19-33 for a detailed description of the results of this instruction.

If the quotient exponent is greater than the largest exponent that can be represented in the operand format, the correct remainder or partial remainder still is produced, and the third-operand result is the correct value, but with the exponent reduced by 192 or 1536 for short or long operands, respectively. The condition code indicates this out-of-range condition.

The $M_4$ field must designate a valid modifier, and the $R_1$, $R_2$, and $R_3$ fields must designate different registers; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Remainder complete; normal quotient
1 Remainder complete; quotient overflow or NaN
2 Remainder incomplete; normal quotient
3 Remainder incomplete; quotient overflow or NaN

***IEEE Exception Conditions:***

- Invalid operation
- Underflow
- Inexact

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification


LzLabs GmbH. Ex. 1008-831

**Programming Notes:**

1. The Remainder operation, as defined in the IEEE standard, is produced by issuing DIVIDE TO INTEGER in an iterative loop, with the $M_4$ field set to 4.
2. The rounding specifications of round to nearest, round toward 0, and round toward $-\infty$ permit the instruction to be used directly to produce the Remainder, MOD, and modulo functions, respectively.
3. When DIVIDE TO INTEGER is used in an iterative loop, all quotients are produced in BFP format but may be considered as portions of a multiple-precision fixed-point number.
4. In the case when the resulting remainder is denormalized, the IEEE standard requires that if traps are implemented and the underflow mask is one, then an underflow trap must occur. To accomplish this, DIVIDE TO INTEGER recognizes underflow on the final remainder but not on the partial remainder. Since in all cases when underflow occurs on the partial remainder it will occur again on the final remainder, recognizing overflow on only the final remainder avoids two underflow traps to be reported for what the standard considers a single Remainder operation.



LzLabs GmbH. Ex. 1008-832

| Dividend (a) | Results for DIVIDE TO INTEGER (a÷b) when Divisor (b) Is | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | −∞ | −Fn | −0 | +0 | +Fn | +∞ | QNaN | SNaN |
| −∞ | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | T(b), cc1 | Xi: T(b*), cc1 |
| −Fn | T(a,+0), cc0 | D(a,b) | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | D(a,b) | T(a,−0), cc0 | T(b), cc1 | Xi: T(b*), cc1 |
| −0 | T(−0,+0), cc0 | T(−0,+0), cc0 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | T(−0,−0), cc0 | T(−0,−0), cc0 | T(b), cc1 | Xi: T(b*), cc1 |
| +0 | T(+0,−0), cc0 | T(+0,−0), cc0 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | T(+0,+0), cc0 | T(+0,+0), cc0 | T(b), cc1 | Xi: T(b*), cc1 |
| +Fn | T(a,−0), cc0 | D(a,b) | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | D(a,b) | T(a,+0), cc0 | T(b), cc1 | Xi: T(b*), cc1 |
| +∞ | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | Xi: T(dNaN), cc1 | T(b), cc1 | Xi: T(b*), cc1 |
| QNaN | T(a), cc1 | T(a), cc1 | T(a), cc1 | T(a), cc1 | T(a), cc1 | T(a), cc1 | T(a), cc1 | Xi: T(b*), cc1 |
| SNaN | Xi: T(a*), cc1 | Xi: T(a*), cc1 | Xi: T(a*), cc1 | Xi: T(a*), cc1 | Xi: T(a*), cc1 | Xi: T(a*), cc1 | Xi: T(a*), cc1 | Xi: T(a*), cc1 |

**Explanation:**

| | |
|---|---|
| * | The SNaN is converted to the corresponding QNaN before it is placed at the target operand location. |
| ccn | Condition code is set to n. |
| D(a,b) | Basic divide-to-integer results. See Part 2 of this figure. |
| Fn | Finite nonzero number (includes both denormalized and normalized). |
| T(r,q) | Results r (the remainder) and q (the quotient) are placed in target operands 1 and 3, respectively. |
| T(x) | Value x is placed in both target operands 1 and 3. |
| Xi: | IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero. |

*Figure 19-24 (Part 1 of 2). Results: DIVIDE TO INTEGER*


LzLabs GmbH. Ex. 1008-833

| $\|q\| < 2^P$ | r = 0 | Case | Is r Tiny | Is r Inexact | Underflow Mask (FPC 0.3) | Inexact mask (FPC 0.4) | Quotient Overflow | Is r Incre-mented | Results for D(a,b) |
|---|---|---|---|---|---|---|---|---|---|
| Yes | Yes | Final | No[1] | No[1] | – | – | No[1] | – | T(r,n), cc0 |
| Yes | No | Final | No | No | – | – | No[1] | – | T(r,n), cc0 |
| Yes | No | Final | Yes | No[1] | 0 | – | No[1] | – | T(r,n), cc0 |
| Yes | No | Final | Yes | No[1] | 1 | – | No[1] | No[1] | T(r·β, n), cc0, PIDu(10) |
| Yes | No | Final | No | Yes | – | 0 | No[1] | – | T(r,n), SFx←1, cc0 |
| Yes | No | Final | No | Yes | – | 1 | No[1] | No | T(r,n), cc0, PIDx(08) |
| Yes | No | Final | No | Yes | – | 1 | No[1] | Yes | T(r,n), cc0, PIDy(0C) |
| No | Yes | Final | No[1] | No[1] | – | – | No | – | T(r,n), cc0 |
| No | Yes | Final | No[1] | No[1] | – | – | Yes | – | T(r,n÷β), cc1 |
| No | No | Partial | –[2] | No[1] | – | – | No | – | T(r,n), cc2 |
| No | No | Partial | –[2] | No[1] | – | – | Yes | – | T(r, n÷β), cc3 |

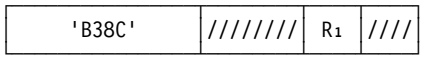
**Explanation:**

| | |
|---|---|
| – | The results do not depend on this condition or mask bit. |
| [1] | This condition is true by virtue of the state of some condition to the left of this column. That is, when $\|q\| < 2^P$, there cannot be a quotient overflow; the cases of remainder is zero, tiny, or inexact are mutually exclusive; and when r is exact, it is not incremented. |
| [2] | Underflow is not recognized for a partial remainder. |
| β | Wrap adjust, which depends on the target format: $\beta = 2^\alpha$, where $\alpha$ is 192 for short and 1536 for long. |
| \|q\| | The absolute value of q., where q is the exact result of a÷b before rounding, assuming unbounded precision and unbounded exponent range. |
| cc0 | Condition code is set to 0 (remainder complete; normal quotient). |
| cc1 | Condition code is set to 1 (remainder complete; quotient overflow). |
| cc2 | Condition code is set to 2 (remainder incomplete; normal quotient). |
| cc3 | Condition code is set to 3 (remainder incomplete; quotient overflow). |
| n | Integer quotient. n = q, rounded toward 0 for partial results and rounded according to the specified quotient rounding mode for final results. The sign of the integer quotient, including the cases of partial and final, wrapped-around overflow and zero, is the exclusive or of the signs of the dividend (a) and divisor (b). |
| r | Remainder. r = a–b·n. A partial remainder is always exact; no rounding is necessary. The sign of a partial remainder is always the same as the sign of the dividend (a). A final remainder is rounded according to the current rounding mode (if necessary). The sign of a zero remainder is the same as the sign of the dividend (a). The sign of a nonzero final remainder is determined by the rules of algebra. |
| P | Precision of the operand, which depends on the target format: P = 24 for short and 53 for long. |
| PIDc(h) | Program interruption for data exception, condition c, with DXC of h in hex. See Figure 19-13 on page 19-15. |
| SFi | IEEE invalid-operation flag, FPC 1.0. |
| SFu | IEEE underflow flag, FPC 1.3. |
| SFx | IEEE inexact flag, FPC 1.4. |
| T(r,n) | Results r (the remainder) and n (the integer quotient) are placed in target operands 1 and 3, respectively. |

*Figure 19-24 (Part 2 of 2). Results: DIVIDE TO INTEGER*

## EXTRACT FPC

EFPC R1 [RRE]




The contents of the FPC (floating-point-control) register are placed in the general register designated by R1.



LzLabs GmbH. Ex. 1008-834

***Condition Code:*** The code remains unchanged.

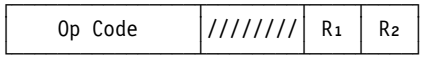
***IEEE Exception Conditions:*** None.

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)

## LOAD AND TEST

```
Mnemonic     R1,R2              [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| LTEBR | 'B302' | Short BFP |
| LTDBR | 'B312' | Long BFP |
| LTXBR | 'B342' | Extended BFP |

The second operand is placed at the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code. The condition code is set the same as for a comparison of the second operand with zero.

The second operand is placed unchanged at the first-operand location. If the second operand is an SNaN, an IEEE-invalid-operation condition is recognized; if there is no interruption, the result is the corresponding QNaN.

See Figure 19-21 on page 19-26 for a detailed description of the results of this instruction.

For LTXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Result is a NaN

***IEEE Exception Conditions:***

- Invalid operation

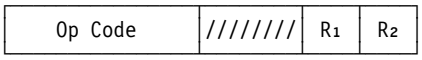
***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (LTXBR only)

**Programming Note:** The IEEE standard makes it optional whether operations such as LOAD AND TEST signal invalid operation when the operand is an SNaN. TEST DATA CLASS may be used to test an operand if signaling is not desired.

## LOAD COMPLEMENT

```
Mnemonic     R1,R2              [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| LCEBR | 'B303' | Short BFP |
| LCDBR | 'B313' | Long BFP |
| LCXBR | 'B343' | Extended BFP |

The second operand is placed at the first-operand location with the sign bit inverted.

The sign bit is inverted even if the operand is zero. The rest of the second operand is placed unchanged at the first-operand location. The sign is inverted for any operand, including a QNaN or SNaN, without causing an arithmetic exception.

For LCXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Result is a NaN

***IEEE Exception Conditions:*** None.

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)
- Specification (LCXBR only)

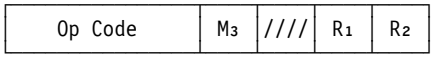
**Programming Note:** The IEEE standard makes it optional whether operations such as LOAD COMPLEMENT signal invalid operation when the operand is an SNaN. LOAD AND TEST may be


LzLabs GmbH. Ex. 1008-835

used in conjunction with this instruction if signalling is desired.

## LOAD FP INTEGER

```
Mnemonic     R1,M3,R2          [RRF]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| FIEBR | 'B357' | Short BFP |
| FIDBR | 'B35F' | Long BFP |
| FIXBR | 'B347' | Extended BFP |

The second operand is rounded to an integer value in the same floating-point format, and the result is placed at the first-operand location.

The second operand, if numeric, is rounded to an integer value as specified by the modifier in the $M_3$ field:

| $M_3$ | Rounding Method |
|---|---|
| 0 | According to current rounding mode |
| 1 | Biased round to nearest |
| 4 | Round to nearest |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |
| 7 | Round toward $-\infty$ |

A modifier other than 0, 1, or 4-7 is invalid.

When the modifier field is zero, rounding is controlled by the current rounding mode in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current rounding mode. Rounding for modifiers 4-7 is the same as for rounding modes 0-3 (binary 00-11), respectively. Biased round to nearest (modifier 1) is the same as round to nearest (modifier 4), except when the second operand is exactly halfway between two integers, in which case the result for biased rounding is the next integer that is greater in magnitude.

In the absence of an interruption, if the second operand is an infinity or a QNaN, the result is that operand; if the second operand is an SNaN, the result is the corresponding QNaN.

The sign of the result is the sign of the second operand, even when the result is zero.

See Figure 19-25 on page 19-37 for a detailed description of the results of this instruction.

The $M_3$ field must designate a valid modifier, and, for FIXBR, the R fields must designate valid floating-point-register pairs. Otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***IEEE Exception Conditions:***

- Invalid operation
- Inexact

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification

**Programming Notes:**

1. LOAD FP INTEGER rounds a BFP number to an integer value. These integers, which remain in the BFP format, should not be confused with binary integers, which have a fixed-point format.
2. If the BFP operand is numeric with a large enough exponent so that it is already an integer, the result value remains the same.



LzLabs GmbH. Ex. 1008-836

| Operand (a) | Is n Inexact (n≠a) | Inv.-Op. Mask (FPC 0.0) | Inexact Mask (FPC 0.4) | Is n Incremented (\|n\|>\|a\|) | Results |
|---|---|---|---|---|---|
| −∞ | No[1] | – | – | – | T(−∞) |
| −Fn | No | – | – | – | T(n) |
| −Fn | Yes | – | 0 | – | T(n), SFx←1 |
| −Fn | Yes | – | 1 | No | T(n), PIDx(08) |
| −Fn | Yes | – | 1 | Yes | T(n), PIDy(0C) |
| −0 | No[1] | – | – | – | T(−0) |
| +0 | No[1] | – | – | – | T(+0) |
| +Fn | No | – | – | – | T(n) |
| +Fn | Yes | – | 0 | – | T(n), SFx←1 |
| +Fn | Yes | – | 1 | No | T(n), PIDx(08) |
| +Fn | Yes | – | 1 | Yes | T(n), PIDy(0C) |
| +∞ | No[1] | – | – | – | T(+∞) |
| QNaN | No[1] | – | – | – | T(a) |
| SNaN | No[1] | 0 | – | – | T(a*), SFi←1 |
| SNaN | No[1] | 1 | – | – | PIDi(80) |

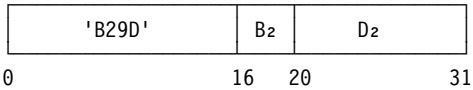
**Explanation:**

| | |
|---|---|
| – | The results do not depend on this condition or mask bit. |
| * | The SNaN is converted to the corresponding QNaN before it is placed at the target operand location. |
| 1 | This condition is true by virtue of the state of some condition to the left of this column. |
| n | The value derived when the source value, a, is rounded to an integer using the specified rounding mode. |
| Fn | Finite nonzero number (includes both denormalized and normalized). |
| PIDc(h) | Program interruption for data exception, condition c, with DXC of h in hex. See Figure 19-13 on page 19-15. |
| SFi | IEEE invalid-operation flag, FPC 1.0. |
| SFx | IEEE inexact flag, FPC 1.4. |
| T(x) | The value x is placed at the target operand location. |

*Figure 19-25. Results: LOAD FP INTEGER*

## LOAD FPC

LFPC $D_2(B_2)$ [S]




The four-byte second operand in storage is loaded into the FPC (floating-point-control) register.

Bits corresponding to unassigned bit positions in the FPC register must be zero; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.


LzLabs GmbH. Ex. 1008-837

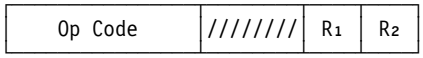
***IEEE Exception Conditions:*** None.

***Program Exceptions:***

- Access (fetch, operand 2)
- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)
- Specification

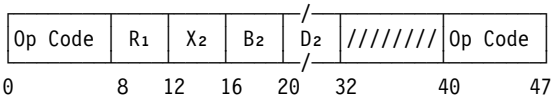
## LOAD LENGTHENED

```
Mnemonic1    R1,R2              [RRE]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| LDEBR | 'B304' | Short BFP operand 2, long BFP operand 1 |
| LXDBR | 'B305' | Long BFP operand 2, extended BFP operand 1 |
| LXEBR | 'B306' | Short BFP operand 2, extended BFP operand 1 |

```
Mnemonic2    R1,D2(X2,B2)       [RXE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| LDEB | 'ED04' | Short BFP operand 2, long BFP operand 1 |
| LXDB | 'ED05' | Long BFP operand 2, extended BFP operand 1 |
| LXEB | 'ED06' | Short BFP operand 2, extended BFP operand 1 |

The second operand is extended to a longer format, and the result is placed at the first-operand location.

The sign of the result is the same as the sign of the source. The exponent of the second operand is converted to the corresponding exponent in the result format, and the fraction is extended by appending zeros on the right. If the second operand is an infinity, the result is an infinity of the same sign. If the second operand is an SNaN, an IEEE-invalid-operation condition is recognized; if there is no interruption, the result is the corresponding QNaN with the fraction extended.

See Figure 19-21 on page 19-26 for a detailed description of the results of this instruction.

For LXDB, LXDBR, LXEB, and LXEBR, the $R_1$ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***IEEE Exception Conditions:***

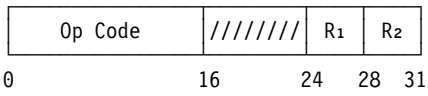
- Invalid operation

***Program Exceptions:***

- Access (fetch, operand 2 of LDEB, LXEB, and LXDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (LXEB, LXEBR, LXDB, LXDBR)

## LOAD NEGATIVE

```
Mnemonic     R1,R2              [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| LNEBR | 'B301' | Short BFP |
| LNDBR | 'B311' | Long BFP |
| LNXBR | 'B341' | Extended BFP |

The second operand is placed at the first-operand location with the sign bit made one.

The sign bit is made one even if the operand is zero. The rest of the second operand is placed unchanged at the first-operand location. The sign is set for any operand, including a QNaN or SNaN, without causing an arithmetic exception.

For LNXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result is zero
1 Result is less than zero
2 --
3 Result is a NaN

***IEEE Exception Conditions:*** None.



LzLabs GmbH. Ex. 1008-838

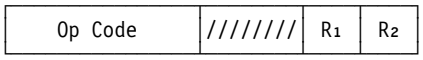
***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)
- Specification (LNXBR only)

**Programming Note:** The IEEE standard makes it optional whether operations such as LOAD NEGATIVE signal invalid operation when the operand is an SNaN. LOAD AND TEST may be used in conjunction with this instruction if signalling is desired.

## LOAD POSITIVE

```
Mnemonic    R1,R2              [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| LPEBR | 'B300' | Short BFP |
| LPDBR | 'B310' | Long BFP |
| LPXBR | 'B340' | Extended BFP |

The second operand is placed at the first-operand location with the sign bit made zero.

The sign bit is made zero, and the rest of the second operand is placed unchanged at the first-operand location. The sign is set for any operand, including a QNaN or SNaN, without causing an arithmetic exception.

For LPXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result is zero
1 --
2 Result is greater than zero
3 Result is a NaN

***IEEE Exception Conditions:*** None.

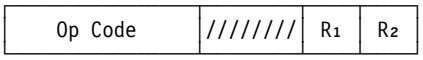
***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)
- Specification (LPXBR only)

**Programming Note:** The IEEE standard makes it optional whether operations such as LOAD POSITIVE signal invalid operation when the operand is an SNaN. LOAD AND TEST may be used in conjunction with this instruction if signalling is desired.

## LOAD ROUNDED

```
Mnemonic    R1,R2              [RRE]
```




| Mnemonic | Op Code | Operands |
|---|---|---|
| LEDBR | 'B344' | Long BFP source, short BFP target |
| LDXBR | 'B345' | Extended BFP source, long BFP target |
| LEXBR | 'B346' | Extended BFP source, short BFP target |

The second operand, in the format of the source, is rounded to the precision of the target, and the result is placed at the first-operand location. The sign of the result is the same as the sign of the second operand.

The second operand, if numeric, is rounded to the precision of the target fraction according to the current rounding mode. Normally, the result is in the format and length of the target. However, when an IEEE overflow or an IEEE underflow occurs and the corresponding mask bit is one, the operation is completed by producing a wrapped result in the same format and length as the source but rounded to the precision of the target. A short-format result replaces the leftmost 32 bits of the target register, and the rightmost 32 bit positions of the target register remain unchanged. A long-format result is placed in a floating-point register, and the other register of the floating-point register pair, if any, remains unchanged. An extended-format result is placed in a floating-point register pair.

See Figure 19-21 on page 19-26 for a detailed description of the results of this instruction.

For LDXBR and LEXBR, the R1 and R2 fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.


LzLabs GmbH. Ex. 1008-839

***IEEE Exception Conditions:***

- Invalid operation
- Overflow
- Underflow
- Inexact

***Program Exceptions:***

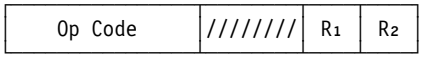
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (LDXBR and LEXBR)

**Programming Notes:**

1. The sign of the rounded result is the same as the sign of the operand, even when the result is zero.
2. The $R_1$ field for LDXBR and LEXBR must designate a valid floating-point-register pair since in certain cases the result is in the extended format. In normal operation for LDXBR and LEXBR, the result format is long or short, respectively, and this result replaces the leftmost 32 bits or 64 bits of the target-register pair. However, when an IEEE overflow or an IEEE underflow occurs and the corresponding mask bit is one, the operation is completed by placing a result in the extended format at the target location. Thus, the program must take into account the fact that these instructions sometimes update both registers of the pair.

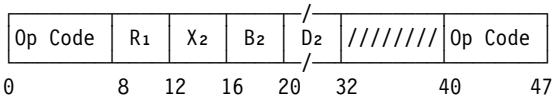
# MULTIPLY

Mnemonic1 $R_1$,$R_2$ [RRE]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| MEEBR | 'B317' | Short BFP |
| MDBR | 'B31C' | Long BFP |
| MXBR | 'B34C' | Extended BFP |
| MDEBR | 'B30C' | Short BFP multiplier and multiplicand, long BFP product |
| MXDBR | 'B307' | Long BFP multiplier and multiplicand, extended BFP product |

Mnemonic2 $R_1$,$D_2$($X_2$,$B_2$) [RXE]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| MEEB | 'ED17' | Short BFP |
| MDB | 'ED1C' | Long BFP |
| MDEB | 'ED0C' | Short BFP multiplier and multiplicand, long BFP product |
| MXDB | 'ED07' | Long BFP multiplier and multiplicand, extended BFP product |

The product of the second operand (the multiplier) and the first operand (the multiplicand) is placed at the first-operand location.

The two BFP operands, if numeric and finite, are multiplied, forming an intermediate product. For MDEB, MDEBR, MXDB, and MXDBR, the intermediate product is converted to the longer target format; the result cannot overflow or underflow and is exact. For MDB, MDBR, MEEB, MEEBR, and MXBR, the result is rounded to the operand format according to the current rounding mode. For MEEB and MEEBR, the result, as for all short-format results, replaces the leftmost 32 bits of the target register, and the rightmost 32 bit positions of the target register remain unchanged.

The sign of the product, if the product is numeric, is the exclusive or of the operand signs. This includes the sign of a zero or infinite product.

If one operand is a zero and the other an infinity, an IEEE-invalid-operation condition is recognized.

See Figure 19-26 on page 19-41 for a detailed description of the results of this instruction.

The $R_1$ field for MXDB, MXDBR, and MXBR, and the $R_2$ field for MXBR, must designate valid floating-point-register pairs. Otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***IEEE Exception Conditions:***

- Invalid operation
- Overflow (MDB, MDBR, MEEB, MEEBR, MXBR)
- Underflow (MDB, MDBR, MEEB, MEEBR, MXBR)



LzLabs GmbH. Ex. 1008-840

- Inexact (MDB, MDBR, MEEB, MEEBR, MXBR)

***Program Exceptions:***

- Access (fetch, operand 2 of MDEB, MEEB, MDB, and MXDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (MXDB, MXDBR, MXBR)

**Programming Note:** Interchanging the two operands in a BFP multiplication does not affect the value of the product when the result is numeric. This is not true, however, when both operands are QNaNs, in which case the result is the first operand; or when both operands are SNaNs and the IEEE-invalid-operation mask bit in the FPC register is zero, in which case the result is the QNaN derived from the first operand.

| First Operand (a) Is | Results for MULTIPLY (a·b) when Second Operand (b) Is | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | −∞ | −Fn | −0 | +0 | +Fn | +∞ | QNaN | SNaN |
| −∞ | T(+∞) | T(+∞) | Xi: T(dNaN) | Xi: T(dNaN) | T(−∞) | T(−∞) | T(b) | Xi: T(b*) |
| −Fn | T(+∞) | R(a·b) | T(+0) | T(−0) | R(a·b) | T(−∞) | T(b) | Xi: T(b*) |
| −0 | Xi: T(dNaN) | T(+0) | T(+0) | T(−0) | T(−0) | Xi: T(dNaN) | T(b) | Xi: T(b*) |
| +0 | Xi: T(dNaN) | T(−0) | T(−0) | T(+0) | T(+0) | Xi: T(dNaN) | T(b) | Xi: T(b*) |
| +Fn | T(−∞) | R(a·b) | T(−0) | T(+0) | R(a·b) | T(+∞) | T(b) | Xi: T(b*) |
| +∞ | T(−∞) | T(−∞) | Xi: T(dNaN) | Xi: T(dNaN) | T(+∞) | T(+∞) | T(b) | Xi: T(b*) |
| QNaN | T(a) | T(a) | T(a) | T(a) | T(a) | T(a) | T(a) | Xi: T(b*) |
| SNaN | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) |

**Explanation:**

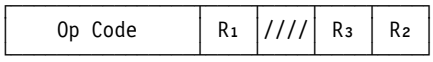
| | |
|---|---|
| * | The SNaN is converted to the corresponding QNaN before it is placed at the target operand location. |
| dNaN | Default quiet NaN. |
| Fn | Finite nonzero number (includes both denormalized and normalized). |
| R(v) | Rounding and range action is performed on the value v. See Figure 19-17 on page 19-21. |
| T(x) | The value x is placed at the target operand location. |
| Xi: | IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero. |

*Figure 19-26. Results: MULTIPLY*


LzLabs GmbH. Ex. 1008-841

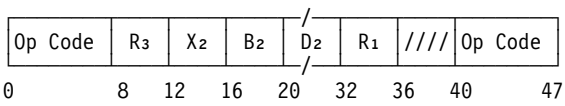
## MULTIPLY AND ADD

Mnemonic1 $R_1,R_3,R_2$ [RRF]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| MAEBR | 'B30E' | Short BFP |
| MADBR | 'B31E' | Long BFP |

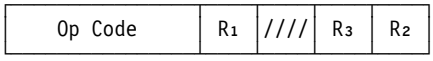
Mnemonic2 $R_1,R_3,D_2(X_2,B_2)$ [RXF]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| MAEB | 'ED0E' | Short BFP |
| MADB | 'ED1E' | Long BFP |

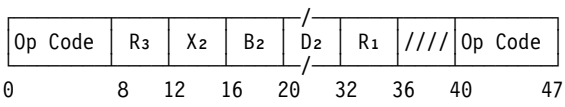
## MULTIPLY AND SUBTRACT

Mnemonic1 $R_1,R_3,R_2$ [RRF]




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| MSEBR | 'B30F' | Short BFP |
| MSDBR | 'B31F' | Long BFP |

Mnemonic2 $R_1,R_3,D_2(X_2,B_2)$ [RXF]




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| MSEB | 'ED0F' | Short BFP |
| MSDB | 'ED1F' | Long BFP |

The third operand is multiplied by the second operand, and then the first operand is added to or subtracted from the product. The sum or difference is placed at the first-operand location. The MULTIPLY AND ADD and MULTIPLY AND SUBTRACT operations may be summarized as:

$$op_1 = op_3 \cdot op_2 \pm op_1$$

When the operands are numeric and finite, the third and second BFP operands are multiplied, forming an intermediate product, and the first operand is then added (or subtracted) algebraically to (or from) the intermediate product, forming an intermediate sum. The intermediate sum, if nonzero, is normalized and rounded to the operand format according to the current rounding mode and then placed at the first-operand location. The exponent and fraction of the intermediate product are maintained exactly; rounding and range checking occur only on the intermediate sum.

See Figure 19-27 on page 19-43 for a detailed description of the results of MULTIPLY AND ADD. The results of MULTIPLY AND SUBTRACT are the same, except that the first operand participates in the operation with its sign bit inverted.

***Condition Code:*** The code remains unchanged.

***IEEE Exception Conditions:***

- Invalid operation
- Overflow
- Underflow
- Inexact

***Program Exceptions:***

- Access (fetch, operand 2 of MAEB, MADB, MSEB, MSDB)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)

**Programming Note:** MULTIPLY AND ADD and MULTIPLY AND SUBTRACT produce a precise intermediate result, and a single rounding operation is performed after the addition or subtraction. This definition is consistent with the RS/6000, and, in certain applications, can be used to great advantage, especially in algorithms used in math libraries.



LzLabs GmbH. Ex. 1008-842

| Third Operand (a) Is | Results, Part 1, for MULTIPLY AND ADD (a·b+c) when Second Operand (b) Is | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | −∞ | −Fn | −0 | +0 | +Fn | +∞ | QNaN | SNaN |
| −∞ | P(+∞) | P(+∞) | Xi: T(dNaN) | Xi: T(dNaN) | P(−∞) | P(−∞) | P(b) | Xi: T(b*) |
| −Fn | P(+∞) | P(a·b) | P(+0) | P(−0) | P(a·b) | P(−∞) | P(b) | Xi: T(b*) |
| −0 | Xi: T(dNaN) | P(+0) | P(+0) | P(−0) | P(−0) | Xi: T(dNaN) | P(b) | Xi: T(b*) |
| +0 | Xi: T(dNaN) | P(−0) | P(−0) | P(+0) | P(+0) | Xi: T(dNaN) | P(b) | Xi: T(b*) |
| +Fn | P(−∞) | P(a·b) | P(−0) | P(+0) | P(a·b) | P(+∞) | P(b) | Xi: T(b*) |
| +∞ | P(−∞) | P(−∞) | Xi: T(dNaN) | Xi: T(dNaN) | P(+∞) | P(+∞) | P(b) | Xi: T(b*) |
| QNaN | P(a) | P(a) | P(a) | P(a) | P(a) | P(a) | P(a) | Xi: T(b*) |
| SNaN | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) | Xi: T(a*) |

*Figure 19-27 (Part 1 of 2). Results: MULTIPLY AND ADD*


LzLabs GmbH. Ex. 1008-843

| Value from Part 1 (p) Is | Results, Part 2, for MULTIPLY AND ADD (a·b+c) when First Operand (c) Is | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | −∞ | −Fn | −0 | +0 | +Fn | +∞ | QNaN | SNaN |
| −∞ | T(−∞) | T(−∞) | T(−∞) | T(−∞) | T(−∞) | Xi: T(dNaN) | T(c) | Xi: T(c*) |
| −Fn | T(−∞) | R(p+c) | R(p) | R(p) | R(p+c) | T(+∞) | T(c) | Xi: T(c*) |
| −0 | T(−∞) | R(c) | T(−0) | Rezd | R(c) | T(+∞) | T(c) | Xi: T(c*) |
| +0 | T(−∞) | R(c) | Rezd | T(+0) | R(c) | T(+∞) | T(c) | Xi: T(c*) |
| +Fn | T(−∞) | R(p+c) | R(p) | R(p) | R(p+c) | T(+∞) | T(c) | Xi: T(c*) |
| +∞ | Xi: T(dNaN) | T(+∞) | T(+∞) | T(+∞) | T(+∞) | T(+∞) | T(c) | Xi: T(c*) |
| QNaN | T(p) | T(p) | T(p) | T(p) | T(p) | T(p) | T(p) | Xi: T(c*) |

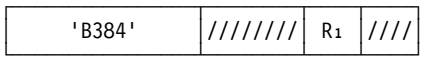
**Explanation:**

| | |
|---|---|
| * | The SNaN is converted to the corresponding QNaN before it is placed at the target operand location. |
| dNaN | Default quiet NaN. |
| Fn | Finite nonzero number (includes both denormalized and normalized). |
| P(x) | The value x is passed to Part 2 of this figure. |
| R(v) | Rounding and range action is performed on the value v. See Figure 19-17 on page 19-21. |
| Rezd | Exact zero-difference result. See Figure 19-17 on page 19-21. |
| T(x) | The value x is placed at the target operand location. |
| Xi: | IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero. |

*Figure 19-27 (Part 2 of 2). Results: MULTIPLY AND ADD*

## SET FPC

SFPC R1 [RRE]




The contents of the general register designated by $R_1$ are placed in the FPC (floating-point-control) register.

Bits corresponding to unassigned bit positions in the FPC must be zero; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

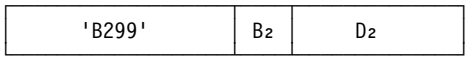
***IEEE Exception Conditions:*** None.

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)
- Specification

## SET ROUNDING MODE

SRNM $D_2(B_2)$ [S]




The rounding-mode bits are set from the second-operand address.

The second-operand address is not used to address data; instead, the rounding-mode bits in the FPC register are set with bits 30 and 31 of the address.

Bits other than 30 and 31 of the second-operand address are ignored.



LzLabs GmbH. Ex. 1008-844

***Condition Code:*** The code remains unchanged.

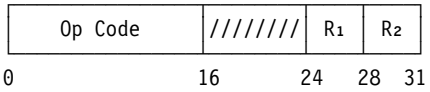
***IEEE Exception Conditions:*** None.

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)

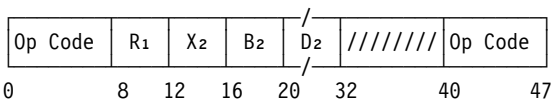
## SQUARE ROOT

```
Mnemonic1     R1,R2                [RRE]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| SQEBR | 'B314' | Short BFP |
| SQDBR | 'B315' | Long BFP |
| SQXBR | 'B316' | Extended BFP |

```
Mnemonic2     R1,D2(X2,B2)         [RXE]
```




| Mnemonic2 | Op Code | Operands |
|---|---|---|
| SQEB | 'ED14' | Short BFP |
| SQDB | 'ED15' | Long BFP |

The square root of the second operand is placed at the first-operand location.

The result rounded according to the current rounding mode is placed at the first-operand location.

If the second operand is a finite positive number, the result is the square root of that number with a plus sign. If the operand is a zero of either sign, the result is a zero of the same sign. If the operand is +∞, the result is +∞.

If the second operand is less than zero, an IEEE-invalid-operation condition is recognized.

See Figure 19-21 on page 19-26 for a detailed description of the results of this instruction.

For SQXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Condition Code:*** The code remains unchanged.

***IEEE Exception Conditions:***

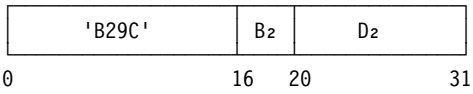
- Invalid operation
- Inexact

***Program Exceptions:***

- Access (fetch, operand 2 of SQEB and SQDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (SQXBR only)

## STORE FPC

```
STFPC   D2(B2)                       [S]
```




The contents of the FPC (floating-point-control) register are placed in storage at the second-operand location.

The operand is four bytes in length. All 32 bits of the FPC register are stored.

***Condition Code:*** The code remains unchanged.

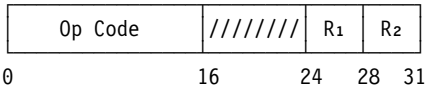
***IEEE Exception Conditions:*** None.

***Program Exceptions:***

- Access (store, operand 2)
- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)

## SUBTRACT

```
Mnemonic1     R1,R2                [RRE]
```




| Mnemonic1 | Op Code | Operands |
|---|---|---|
| SEBR | 'B30B' | Short BFP |
| SDBR | 'B31B' | Long BFP |
| SXBR | 'B34B' | Extended BFP |



LzLabs GmbH. Ex. 1008-845

```
Mnemonic2    R1,D2(X2,B2)        [RXE]

 ┌─────────┬────┬────┬────┬──/──┬────────┬─────────┐
 │ Op Code │ R1 │ X2 │ B2 │ D2  │ //////// │ Op Code │
 └─────────┴────┴────┴────┴──/──┴────────┴─────────┘
 0         8    12   16   20    32       40        47
```

| Mnemonic2 | Op Code | Operands |
|---|---|---|
| SEB | 'ED0B' | Short BFP |
| SDB | 'ED1B' | Long BFP |

The second operand is subtracted from the first operand, and the difference is placed at the first-operand location.

The execution of SUBTRACT is identical to that of ADD, except that the second operand participates in the operation with its sign bit inverted. See Figure 19-16 on page 19-20 for the detailed results of ADD.

For SXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Result is a NaN

***IEEE Exception Conditions:***

- Invalid operation
- Overflow
- Underflow
- Inexact

***Program Exceptions:***

- Access (fetch, operand 2 of SEB and SDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception condition
- Operation (if the BFP facility is not installed)
- Specification (SXBR only)

# TEST DATA CLASS

```
Mnemonic     R1,D2(X2,B2)        [RXE]

 ┌─────────┬────┬────┬────┬──/──┬────────┬─────────┐
 │ Op Code │ R1 │ X2 │ B2 │ D2  │ //////// │ Op Code │
 └─────────┴────┴────┴────┴──/──┴────────┴─────────┘
 0         8    12   16   20    32       40        47
```

| Mnemonic | Op Code | Operands |
|---|---|---|
| TCEB | 'ED10' | Short BFP |
| TCDB | 'ED11' | Long BFP |
| TCXB | 'ED12' | Extended BFP |

The class and sign of the first operand are examined to select one bit from the second-operand address. Condition code 0 or 1 is set according to whether the selected bit is zero or one, respectively.

The second-operand address is not used to address data; instead, the rightmost 12 bits of the address, bits 20-31, are used to specify 12 combinations of operand class and sign. Bits 0-19 of the second-operand address are ignored.

As shown in Figure 19-28, BFP operands are divided into six classes: zero, normalized number, denormalized number, infinity, quiet NaN, and signaling NaN.

| BFP Operand Class | Bit Used when Sign Is + | Bit Used when Sign Is − |
|---|---|---|
| Zero | 20 | 21 |
| Normalized number | 22 | 23 |
| Denormalized number | 24 | 25 |
| Infinity | 26 | 27 |
| Quiet NaN | 28 | 29 |
| Signaling NaN | 30 | 31 |

*Figure 19-28. Second-Operand-Address Bits for TEST DATA CLASS*

One or more of the second-operand-address bits may be set to one. If the second-operand-address bit corresponding to the class and sign of the first operand is one, condition code 1 is set; otherwise, condition code 0 is set.

Operands, including SNaNs and QNaNs, are examined without causing an arithmetic exception.



LzLabs GmbH. Ex. 1008-846

For TCXB, the $R_1$ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

***Resulting Condition Code:***

0 Selected bit is 0 (no match)
1 Selected bit is 1 (match)
2 --
3 --

***IEEE Exception Conditions:*** None.

***Program Exceptions:***

- Data with DXC 2, BFP instruction
- Operation (if the BFP facility is not installed)
- Specification (TCXB only)

**Programming Note:** TEST DATA CLASS provides a way to test an operand without risk of an exception or setting the IEEE flags.


LzLabs GmbH. Ex. 1008-847

LzLabs GmbH. Ex. 1008-848

# Appendix A. Number Representation and Instruction-Use Examples


Number Representation . . . . . A-2
Binary Integers . . . . . A-2
Signed Binary Integers . . . . . A-2
Unsigned Binary Integers . . . . . A-3
Decimal Integers . . . . . A-4
Hexadecimal-Floating-Point Numbers . . . A-5
Conversion Example . . . . . A-6
Instruction-Use Examples . . . . . A-6
Machine Format . . . . . A-6
Assembler-Language Format . . . . . A-7
Addressing Mode in Examples . . . . . A-7
General Instructions . . . . . A-7
ADD HALFWORD (AH) . . . . . A-7
AND (N, NC, NI, NR) . . . . . A-7
NI Example . . . . . A-8
Linkage Instructions (BAL, BALR, BAS, BASR, BASSM, BSM) . . . . . A-8
Other BALR and BASR Examples . . . A-9
BRANCH AND STACK (BAKR) . . . . . A-9
BAKR Example 1 . . . . . A-10
BAKR Example 2 . . . . . A-10
BAKR Example 3 . . . . . A-11
BRANCH ON CONDITION (BC, BCR) . A-11
BRANCH ON COUNT (BCT, BCTR) . . A-12
BRANCH ON INDEX HIGH (BXH) . . . . A-12
BXH Example 1 . . . . . A-12
BXH Example 2 . . . . . A-12
BRANCH ON INDEX LOW OR EQUAL (BXLE) . . . . . A-13
BXLE Example 1 . . . . . A-13
BXLE Example 2 . . . . . A-14
COMPARE AND FORM CODEWORD (CFC) . . . . . A-14
COMPARE HALFWORD (CH) . . . . . A-14
COMPARE LOGICAL (CL, CLC, CLI, CLR) . . . . . A-14
CLC Example . . . . . A-14
CLI Example . . . . . A-15
CLR Example . . . . . A-15
COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) . . . . . A-15
COMPARE LOGICAL LONG (CLCL) . . A-16
COMPARE LOGICAL STRING (CLST) . A-17
CONVERT TO BINARY (CVB) . . . . . A-18
CONVERT TO DECIMAL (CVD) . . . . . A-18
DIVIDE (D, DR) . . . . . A-19
EXCLUSIVE OR (X, XC, XI, XR) . . . . A-19
XC Example . . . . . A-19
XI Example . . . . . A-20
EXECUTE (EX) . . . . . A-21
INSERT CHARACTERS UNDER MASK (ICM) . . . . . A-21
LOAD (L, LR) . . . . . A-22
LOAD ADDRESS (LA) . . . . . A-22
LOAD HALFWORD (LH) . . . . . A-23
MOVE (MVC, MVI) . . . . . A-23
MVC Example . . . . . A-23
MVI Example . . . . . A-24
MOVE INVERSE (MVCIN) . . . . . A-24
MOVE LONG (MVCL) . . . . . A-25
MOVE NUMERICS (MVN) . . . . . A-25
MOVE STRING (MVST) . . . . . A-26
MOVE WITH OFFSET (MVO) . . . . . A-26
MOVE ZONES (MVZ) . . . . . A-27
MULTIPLY (M, MR) . . . . . A-27
MULTIPLY HALFWORD (MH) . . . . . A-27
OR (O, OC, OI, OR) . . . . . A-28
OI Example . . . . . A-28
PACK (PACK) . . . . . A-28
SEARCH STRING (SRST) . . . . . A-29
SRST Example 1 . . . . . A-29
SRST Example 2 . . . . . A-29
SHIFT LEFT DOUBLE (SLDA) . . . . . A-29
SHIFT LEFT SINGLE (SLA) . . . . . A-30
STORE CHARACTERS UNDER MASK (STCM) . . . . . A-30
STORE MULTIPLE (STM) . . . . . A-30
TEST UNDER MASK (TM) . . . . . A-31
TRANSLATE (TR) . . . . . A-31
TRANSLATE AND TEST (TRT) . . . . . A-32
UNPACK (UNPK) . . . . . A-33
UPDATE TREE (UPT) . . . . . A-34
Decimal Instructions . . . . . A-34
ADD DECIMAL (AP) . . . . . A-34
COMPARE DECIMAL (CP) . . . . . A-34
DIVIDE DECIMAL (DP) . . . . . A-34
EDIT (ED) . . . . . A-35
EDIT AND MARK (EDMK) . . . . . A-36
MULTIPLY DECIMAL (MP) . . . . . A-36
SHIFT AND ROUND DECIMAL (SRP) . A-37
Decimal Left Shift . . . . . A-37
Decimal Right Shift . . . . . A-37
Decimal Right Shift and Round . . . . A-38
Multiplying by a Variable Power of 10 . A-38
ZERO AND ADD (ZAP) . . . . . A-38
Hexadecimal-Floating-Point Instructions . . A-39
ADD NORMALIZED (AD, ADR, AE, AER, AXR) . . . . . A-39


© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-849

ADD UNNORMALIZED (AU, AUR, AW, AWR) . . . A-39
COMPARE (CD, CDR, CE, CER) . . . . A-40
DIVIDE (DD, DDR, DE, DER) . . . . . A-40
HALVE (HDR, HER) . . . . . A-41
MULTIPLY (MD, MDR, MDE, MDER, MXD, MXDR, MXR) . . . . . A-41
Hexadecimal-Floating-Point-Number Conversion . . . . . A-42
Fixed Point to Hexadecimal Floating Point . . . . . A-42
Hexadecimal Floating Point to Fixed Point . . . . . A-42
Multiprogramming and Multiprocessing Examples . . . . . A-43
Example of a Program Failure Using OR Immediate . . . . . A-43
Conditional Swapping Instructions (CS, CDS) . . . . . A-44
Setting a Single Bit . . . . . A-44
Updating Counters . . . . . A-45
Bypassing Post and Wait . . . . . A-45
Bypass Post Routine . . . . . A-45
Bypass Wait Routine . . . . . A-46
Lock/Unlock . . . . . A-46
Lock/Unlock with LIFO Queuing for Contentions . . . . . A-46
Lock/Unlock with FIFO Queuing for Contentions . . . . . A-47
Free-Pool Manipulation . . . . . A-48
PERFORM LOCKED OPERATION (PLO) A-50
Sorting Instructions . . . . . A-51
Tree Format . . . . . A-51
Example of Use of Sort Instructions . . . A-53

# Number Representation

## Binary Integers

### Signed Binary Integers

Signed binary integers are most commonly represented as halfwords (16 bits) or words (32 bits). In both lengths, the leftmost bit (bit 0) is the sign of the number. The remaining bits (bits 1-15 for halfwords and 1-31 for words) are used to specify the magnitude of the number. Binary integers are also referred to as fixed-point numbers, because the radix point (binary point) is considered to be fixed at the right, and any scaling is done by the programmer.

Positive binary integers are in true binary notation with a zero sign bit. Negative binary integers are in two's-complement notation with a one bit in the sign position. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit (that is, all zeros for positive numbers, all ones for negative numbers).

Negative binary integers are formed in two's-complement notation by inverting each bit of the positive binary integer and adding one. As an example using the halfword format, the binary number with the decimal value +26 is made negative (-26) in the following manner:

```
+26    0 000 0000 0001 1010
Invert 1 111 1111 1110 0101
Add 1                     1
       ____________________

-26    1 111 1111 1110 0110 (Two's comple-
                             ment form)
(S is the sign bit.)
```

This is equivalent to subtracting the number:

```
        00000000 00011010
from
      1 00000000 00000000
```

Negative binary integers are changed to positive in the same manner.

The following addition examples illustrate two's-complement arithmetic and overflow conditions. Only eight bit positions are used.

```
1.  +57 = 0011 1001
    +35 = 0010 0011
    ___________________

    +92 = 0101 1100

2.  +57 = 0011 1001
    -35 = 1101 1101
    ___________________

    +22 = 0001 0110 No overflow — carry into
                    leftmost position and
                    carry out

3.  +35 = 0010 0011
    -57 = 1100 0111
    ___________________

    -22 = 1110 1010 Sign change only — no
                    carry into leftmost posi-
                    tion and no carry out
```



LzLabs GmbH. Ex. 1008-850

```
4.  -57 = 1100 0111
    -35 = 1101 1101
    ---------------
    -92 = 1010 0100 No overflow — carry into
                    leftmost position and
                    carry out

5.  +57 = 0011 1001
    +92 = 0101 1100
    ---------------
   +149 =*1001 0101 *Overflow — carry into
                    leftmost position, no
                    carry out

6.  -57 = 1100 0111
    -92 = 1010 0100
    ---------------
   -149 =*0110 1011 *Overflow — no carry into
                    leftmost position but carry
                    out
```

The presence or absence of an overflow condition may be recognized from the carries:

- There is no overflow:
  1. If there is no carry into the leftmost bit position and no carry out (examples 1 and 3).
  2. If there is a carry into the leftmost position and also a carry out (examples 2 and 4).
- There is an overflow:
  1. If there is a carry into the leftmost position but no carry out (example 5).
  2. If there is no carry into the leftmost position but there is a carry out (example 6).

The following are 16-bit signed binary integers. The first is the maximum positive 16-bit binary integer. The last is the maximum negative 16-bit binary integer (the negative 16-bit binary integer with the greatest absolute value).

```
 2¹⁵-1 =  32,767 = 0 111 1111 1111 1111
 2⁰    =       1 = 0 000 0000 0000 0001
 0     =       0 = 0 000 0000 0000 0000
-2⁰    =      -1 = 1 111 1111 1111 1111
-2¹⁵   = -32,768 = 1 000 0000 0000 0000
```

Figure A-1 illustrates several 32-bit signed binary integers arranged in descending order. The first is the maximum positive binary integer that can be represented by 32 bits, and the last is the maximum negative binary integer that can be represented by 32 bits.

```
 2³¹-1 =  2 147 483 647 = 0 111 1111 1111 1111 1111 1111 1111 1111
 2¹⁶   =         65 536 = 0 000 0000 0000 0001 0000 0000 0000 0000
 2⁰    =              1 = 0 000 0000 0000 0000 0000 0000 0000 0001
 0     =              0 = 0 000 0000 0000 0000 0000 0000 0000 0000
-2⁰    =             -1 = 1 111 1111 1111 1111 1111 1111 1111 1111
-2¹    =             -2 = 1 111 1111 1111 1111 1111 1111 1111 1110
-2¹⁶   =        -65 536 = 1 111 1111 1111 1111 0000 0000 0000 0000
-2³¹+1 = -2 147 483 647 = 1 000 0000 0000 0000 0000 0000 0000 0001
-2³¹   = -2 147 483 648 = 1 000 0000 0000 0000 0000 0000 0000 0000
```

*Figure A-1. 32-Bit Signed Binary Integers*

## Unsigned Binary Integers

Certain instructions, such as ADD LOGICAL, treat binary integers as unsigned rather than signed. Unsigned binary integers have the same format as signed binary integers, except that the leftmost bit is interpreted as another numeric bit rather than a sign bit. There is no complement notation because all unsigned binary integers are considered positive.

The following examples illustrate the addition of unsigned binary integers. Only eight bit positions are used. The examples are numbered the same as the corresponding examples for signed binary integers.

```
1.   57 = 0011 1001
     35 = 0010 0011
     --------------
     92 = 0101 1100

2.   57 = 0011 1001
    221 = 1101 1101
    ---------------
    278 =*0001 0110 *Carry out of leftmost
                    position

3.   35 = 0010 0011
    199 = 1100 0111
    ---------------
    234 = 1110 1010

4.  199 = 1100 0111
    221 = 1101 1101
    ---------------
    420 =*1010 0100 *Carry out of leftmost
                    position
```



LzLabs GmbH. Ex. 1008-851

```
5.   57 = 0011 1001
     92 = 0101 1100
     ---------------
    149 = 1001 0101

6.  199 = 1100 0111
    164 = 1010 0100
    ---------------
    363 =*0110 1011 *Carry out of leftmost
                     position
```

A carry out of the leftmost bit position may or may not imply an overflow, depending on the application.

Figure A-2 illustrates several 32-bit unsigned binary integers arranged in descending order.

```
2³²-1  =  4 294 967 295 = 1111 1111 1111 1111 1111 1111 1111 1111
2³¹    =  2 147 483 648 = 1000 0000 0000 0000 0000 0000 0000 0000
2³¹-1  =  2 147 483 647 = 0111 1111 1111 1111 1111 1111 1111 1111
2¹⁶    =         65 536 = 0000 0000 0000 0001 0000 0000 0000 0000
2⁰     =              1 = 0000 0000 0000 0000 0000 0000 0000 0001
0      =              0 = 0000 0000 0000 0000 0000 0000 0000 0000
```

*Figure A-2. 32-Bit Unsigned Binary Integers*

## Decimal Integers

Decimal integers consist of one or more decimal digits and a sign. Each digit and the sign are represented by a 4-bit code. The decimal digits are in binary-coded decimal (BCD) form, with the values 0-9 encoded as 0000-1001. The sign is usually represented as 1100 (C hex) for plus and 1101 (D hex) for minus. These are the preferred sign codes, which are generated by the machine for the results of decimal-arithmetic operations. There are also several alternate sign codes (1010, 1110, and 1111 for plus; 1011 for minus). The alternate sign codes are accepted by the machine as valid in source operands but are not generated for results.

Decimal integers may have different lengths, from one to 16 bytes. There are two decimal formats: packed and zoned. In the packed format, each byte contains two decimal digits, except for the rightmost byte, which contains the sign code in the right half. For decimal arithmetic, the number of decimal digits in the packed format can vary from one to 31. Because decimal integers must consist of whole bytes and there must be a sign code on the right, the number of decimal digits is always odd. If an even number of significant digits is desired, a leading zero must be inserted on the left.

In the zoned format, each byte consists of a decimal digit on the right and the zone code 1111 (F hex) on the left, except for the rightmost byte where the sign code replaces the zone code. Thus, a decimal integer in the zoned format can have from one to 16 digits. The zoned format may be used directly for input and output in the extended binary-coded-decimal interchange code (EBCDIC), except that the sign must be separated from the rightmost digit and handled as a separate character. For positive (unsigned) numbers, however, the sign can simply be represented by the zone code of the rightmost digit because the zone code is one of the acceptable alternate codes for plus.

In either format, negative decimal integers are represented in true notation with a separate sign. As for binary integers, the radix point (decimal point) of decimal integers is considered to be fixed at the right, and any scaling is done by the programmer.

The following are some examples of decimal integers shown in hexadecimal notation:

| Decimal Value | Packed Format | Zoned Format |
|---|---|---|
| +123 | 12 3C<br>or<br>12 3F | F1 F2 C3<br>or<br>F1 F2 F3 |
| -4321 | 04 32 1D | F4 F3 F2 D1 |
| +000050 | 00 00 05 0C<br>or<br>00 00 05 0F | F0 F0 F0 F0 F5 C0<br>or<br>F0 F0 F0 F0 F5 F0 |
| -7 | 7D | D7 |
| 00000 | 00 00 0C<br>or<br>00 00 0F | F0 F0 F0 F0 C0<br>or<br>F0 F0 F0 F0 F0 |



LzLabs GmbH. Ex. 1008-852

Under some circumstances, a zero with a minus sign (negative zero) is produced. For example, the multiplicand:

```
00 12 3D   (-123)
```

times the multiplier:

```
   0C      (+0)
```

generates the product:

```
00 00 0D   (-0)
```

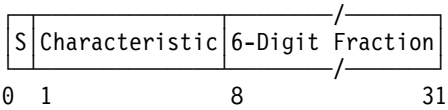
because the product sign follows the algebraic rule of signs even when the value is zero. A negative zero, however, is equivalent to a positive zero in that they compare equal in a decimal comparison.

## Hexadecimal-Floating-Point Numbers

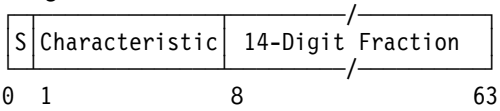
A hexadecimal-floating-point (HFP) number is expressed as a hexadecimal fraction multiplied by a separate power of 16. The term floating point indicates that the placement, of the radix (hexadecimal) point, or scaling, is automatically maintained by the machine.

The part of an HFP number which represents the significant digits of the number is called the fraction. A second part specifies the power (exponent) to which 16 is raised and indicates the location of the radix point of the number. The fraction and exponent may be represented by 32 bits (short format), 64 bits (long format), or 128 bits (extended format).

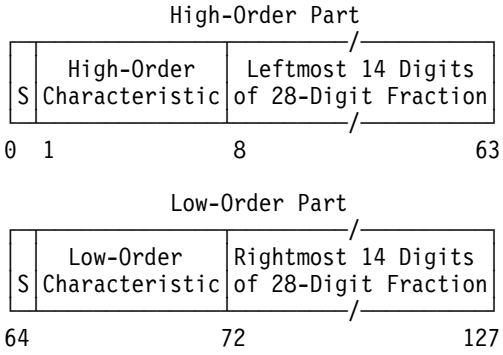
Short HFP Number




Long HFP Number




Extended HFP Number




An HFP number has two signs: one for the fraction and one for the exponent. The fraction sign, which is also the sign of the entire number, is the leftmost bit of each format (0 for plus, 1 for minus). The numeric part of the fraction is in true notation regardless of the sign. The numeric part is contained in bits 8-31 for the short format, in bits 8-63 for the long format, and in bits 8-63 followed by bits 72-127 for the extended format.

The exponent sign is obtained by expressing the exponent in excess-64 notation; that is, the exponent is added as a signed number to 64. The resulting number is called the characteristic. It is located in bits 1-7 for all formats. The characteristic can vary from 0 to 127, permitting the exponent to vary from -64 through 0 to +63. This provides a scale multiplier in the range of $16^{-64}$ to $16^{+63}$. A nonzero fraction, if normalized, has a value less than one and greater than or equal to 1/16, so that the range covered by the magnitude M of a normalized floating-point number is:

$$16^{-65} \le M < 16^{63}$$

In decimal terms:

$16^{-65}$ is approximately $5.4 \times 10^{-79}$

$16^{63}$ is approximately $7.2 \times 10^{75}$

More precisely,

In the short format:

$$16^{-65} \le M \le (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \le M \le (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \le M \le (1 - 16^{-28}) \times 16^{63}$$

Within a given fraction length (6, 14, or 28 digits), an HFP operation will provide the greatest preci-


LzLabs GmbH. Ex. 1008-853

sion if the fraction is normalized. A fraction is normalized when the leftmost digit (bit positions 8, 9, 10, and 11) is nonzero. It is unnormalized if the leftmost digit contains all zeros.

If normalization of the operand is desired, the HFP instructions that provide automatic normalization are used. This automatic normalization is accomplished by left-shifting the fraction (four bits per shift) until a nonzero digit occupies the leftmost digit position. The characteristic is reduced by one for each digit shifted.

Figure A-3 illustrates sample normalized short HFP numbers. The last two numbers represent the smallest and the largest positive normalized numbers.

```
  1.0        = +1/16x16¹      = 0 100 0001 0001 0000 0000 0000 0000 0000₂
  0.5        = +8/16x16⁰      = 0 100 0000 1000 0000 0000 0000 0000 0000₂
  1/64       = +4/16x16-¹     = 0 011 1111 0100 0000 0000 0000 0000 0000₂
  0.0        = +0   x16-⁶⁴    = 0 000 0000 0000 0000 0000 0000 0000 0000₂
 -15.0       = -15/16x16¹     = 1 100 0001 1111 0000 0000 0000 0000 0000₂
  5.4x10-⁷⁹ ≃ +1/16x16-⁶⁴    = 0 000 0000 0001 0000 0000 0000 0000 0000₂
  7.2x10⁷⁵  ≃ (1-16-⁶)x16⁶³  = 0 111 1111 1111 1111 1111 1111 1111 1111₂
```

*Figure A-3. Normalized Short Hexadecimal-Floating-Point Numbers*

## Conversion Example

Convert the decimal number 59.25 to a short HFP number. (In another appendix are tables for the conversion of hexadecimal and decimal integers and fractions.)

1. The number is separated into a decimal integer and a decimal fraction.

   59.25 = 59 plus 0.25

2. The decimal integer is converted to its hexadecimal representation.

   $59_{10} = 3B_{16}$

3. The decimal fraction is converted to its hexadecimal representation.

   $0.25_{10} = 0.4_{16}$

4. The integral and fractional parts are combined and expressed as a fraction times a power of 16 (exponent).

   $3B.4_{16} = 0.3B4_{16} \times 16^2$

5. The characteristic is developed from the exponent and converted to binary.

   ```
   base + exponent  = characteristic
   64   + 2         = 66 = 1000010
   ```

6. The fraction is converted to binary and grouped hexadecimally.

   $.3B4_{16} = .0011\ 1011\ 0100$

7. The characteristic and the fraction are stored in the short format. The sign position contains the sign of the fraction.

   ```
   S Char      Fraction
   0 1000010   0011 1011 0100 0000 0000 0000
   ```

Examples of instruction sequences that may be used to convert between signed binary integers and HFP numbers are shown in "Hexadecimal-Floating-Point-Number Conversion" on page A-42.

## Instruction-Use Examples

The following examples illustrate the use of many of the unprivileged instructions. Before studying one of these examples, the reader should consult the instruction description.

The instruction-use examples are written principally for assembler-language programmers, to be used in conjunction with the appropriate assembler-language publications.

Most examples present one particular instruction, both as it is written in an assembler-language statement and as it appears when assembled in storage (machine format).

## Machine Format

All machine-format values are given in hexadecimal notation unless otherwise specified. Storage addresses are also given in hexadecimal. Hexadecimal operands are shown converted into binary, decimal, or both if such conversion helps to clarify the example for the reader.



LzLabs GmbH. Ex. 1008-854

## Assembler-Language Format

In assembler-language statements, registers and lengths are presented in decimal. Displacements, immediate operands, and masks may be shown in decimal, hexadecimal, or binary notation; for example, 12, X'C', and B'1100' represent the same value. Whenever the value in a register or storage location is referred to as "not significant," this value is replaced during the execution of the instruction.

When SS-format instructions are written in the assembler language, lengths are given as the total number of bytes in the field. This differs from the machine definition, in which the length field specifies the number of bytes to be added to the field address to obtain the address of the last byte of the field. Thus, the machine length is one less than the assembler-language length. The assembler program automatically subtracts one from the length specified when the instruction is assembled.

In some of the examples, symbolic addresses are used in order to simplify the examples. In assembler-language statements, a symbolic address is represented as a mnemonic term written in all capitals, such as FLAGS, which may denote the address of a storage location containing data or program-control information. When symbolic addresses are used, the assembler supplies actual base and displacement values according to the programmer's specifications. Therefore, the actual values for base and displacement are not shown in the assembler-language format or in the machine-language format. For assembler-language formats, in the labels that designate instruction fields, the letter "S" is used to indicate the combination of base and displacement fields for an operand address. (For example, S2 represents the combination of B2 and D2.) In the machine-language format, the base and displacement address components are shown as asterisks (****).

### Addressing Mode in Examples

Except where otherwise specified, the examples assume the 24-bit addressing mode.

## General Instructions

(See Chapter 7, "General Instructions" for a complete description of the general instructions.)

## ADD HALFWORD (AH)

The ADD HALFWORD instruction algebraically adds the contents of a two-byte field in storage to the contents of a register. The storage operand is expanded to 32 bits after it is fetched and before it is used in the add operation. The expansion consists in propagating the leftmost (sign) bit 16 positions to the left. For example, assume that the contents of storage locations 2000-2001 are to be added to register 5. Initially:

Register 5 contains 00 00 00 19 = $25_{10}$.

Storage locations 2000-2001 contain FF FE = $-2_{10}$.

Register 12 contains 00 00 18 00.

Register 13 contains 00 00 01 50.

The format of the required instruction is:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 4A | 5 | D | C | 6B0 |

Assembler Format

```
Op Code  R1,D2(X2,B2)
-------------------------
   AH    5,X'6B0'(13,12)
```

After the instruction is executed, register 5 contains 00 00 00 17 = $23_{10}$. Condition code 2 is set to indicate a result greater than zero.

## AND (N, NC, NI, NR)

When the Boolean operator AND is applied to two bits, the result is one when both bits are one; otherwise, the result is zero. When two bytes are ANDed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of ANDing two bytes:

```
First-operand byte:   0011 0101₂
Second-operand byte:  0101 1100₂
-------------------------------
Result byte:          0001 0100₂
```



LzLabs GmbH. Ex. 1008-855

### NI Example

A frequent use of the AND instruction is to set a particular bit to zero. For example, assume that storage location 4891 contains 0100 0011₂. To set the rightmost bit of this byte to zero without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

| Op Code | $I_2$ | $B_1$ | $D_1$ |
|---|---|---|---|
| 94 | FE | 8 | 001 |

Assembler Format

```
Op Code  D1(B1),I2
NI       1(8),X'FE'
```

When this instruction is executed, the byte in storage is ANDed with the immediate byte (the $I_2$ field of the instruction):

```
Location 4891:    0100 0011₂
Immediate byte:   1111 1110₂
Result:           0100 0010₂
```

The resulting byte, with bit 7 set to zero, is stored back in location 4891. Condition code 1 is set.

## Linkage Instructions (BAL, BALR, BAS, BASR, BASSM, BSM)

Four unprivileged instructions (BRANCH AND LINK, BRANCH AND SAVE, BRANCH AND SAVE AND SET MODE, and BRANCH AND SET MODE) are available, together with the unconditional branch (BRANCH ON CONDITION with a mask of 15), to provide linkage between subroutines. BRANCH AND LINK (BAL or BALR) is provided primarily for compatibility with programs written for System/370; BRANCH AND SAVE (BAS or BASR) is recommended instead for programs which are to be executed using ESA/370. The instructions BRANCH AND SAVE AND SET MODE (BASSM) and BRANCH AND SET MODE (BSM) provide subroutine linkage together with switching between the 24-bit and the 31-bit addressing modes. The use of these instructions is discussed in a programming note at the end of "Subroutine Linkage without the Linkage Stack." (See also the semiprivileged instruction BRANCH AND STACK.)

The following example compares the operation of these instructions and of the unconditional-branch instruction BRANCH ON CONDITION (BC or BCR with a mask of 15). Assume that each instruction in turn is located at the current instruction address, ready to be executed next. For the first set of examples, the addressing-mode bit, PSW bit 32, is initially zero (24-bit addressing in effect). For the second set, PSW bit 32 is initially one (31-bit addressing). Assume also that general register 5 is to receive the linkage information, and that general register 6 contains the branch address.

The format of the BALR instruction is:

Machine Format

| Op Code | $R_1$ | $R_2$ |
|---|---|---|
| 05 | 5 | 6 |

Assembler Format

```
Op Code  R1,R2
BALR     5,6
```

The other linkage instructions in the RR format have the same format but different op codes:

```
BASR   0D
BASSM  0C
BSM    0B
```

For comparison with the RR-format instructions, the results of two RX-format instructions are also shown.

The format of the BAL instruction is:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 45 | 5 | 0 | 6 | 000 |

Assembler Format

```
Op Code  R1,D2(X2,B2)
BAL      5,0(0,6)
```

The BAS instruction has the same format, but the op code is 4D.

The BCR instruction specifies only one register:



LzLabs GmbH. Ex. 1008-856

Machine Format

| Op Code | $M_1$ | $R_2$ |
|---|---|---|
| 07 | F | 6 |

Assembler Format

| Op Code | $M_1$,$R_2$ |
|---|---|
| BCR | 15,6 |

Assume that:

Register 5 contains BB BB BB BB.

Register 6 contains 82 46 8A CE.

PSW bits 32-63 contain

00 00 10 D6 (for 24-bit addressing).
80 00 10 D6 (for 31-bit addressing).

Condition code is $01_2$.

Program mask is $1100_2$.

The effect of executing each instruction in turn is as follows:

**24-Bit Mode Initially**

| Instruction | Register 5 | PSW (32-63) |
|---|---|---|
| Before | BB BB BB BB | 00 00 10 D6 |
| BCR 15,6 | BB BB BB BB | 00 46 8A CE |
| BAL 5,0(0,6) | 9C 00 10 DA | 00 46 8A CE |
| BAS 5,0(0,6) | 00 00 10 DA | 00 46 8A CE |
| BALR 5,6 | 5C 00 10 D8 | 00 46 8A CE |
| BASR 5,6 | 00 00 10 D8 | 00 46 8A CE |
| BASSM 5,6 | 00 00 10 D8 | 82 46 8A CE |
| BSM 5,6 | 3B BB BB BB | 82 46 8A CE |

**31-Bit Mode Initially**

| Instruction | Register 5 | PSW (32-63) |
|---|---|---|
| Before | BB BB BB BB | 80 00 10 D6 |
| BCR 15,6 | BB BB BB BB | 82 46 8A CE |
| BAL 5,0(0,6) | 80 00 10 DA | 82 46 8A CE |
| BAS 5,0(0,6) | 80 00 10 DA | 82 46 8A CE |
| BALR 5,6 | 80 00 10 D8 | 82 46 8A CE |
| BASR 5,6 | 80 00 10 D8 | 82 46 8A CE |
| BASSM 5,6 | 80 00 10 D8 | 82 46 8A CE |
| BSM 5,6 | BB BB BB BB | 82 46 8A CE |

Note that a value of zero in the $R_2$ field of any of the RR-format instructions indicates that the branching function is not to be performed; it does not refer to register 0. Likewise, a value of zero in the $R_1$ field of the BSM instruction indicates that the old value of PSW bit 32 is not to be saved and that register 0 is to be left unchanged. Register 0 can be designated by the $R_1$ field of instructions BAL, BALR, BAS, BASR, and BASSM, however. In the RX-format branch instructions, branching occurs independent of whether there is a value of zero in the $B_2$ field or $X_2$ field of the instruction. However, when the field is zero, instead of using the contents of general register 0, a value of zero is used for that component of address generation.

**Programming Note:** It should be noted that execution of BAL in the 24-bit addressing mode results in bit 0 of register 5 being set to one. This is because the ILC for an RX-format instruction is 10. This is the only case in which bit zero of the return register does not correctly reflect the addressing mode of the caller. Thus, BSM may be used to return for BALR, BAS, BASR, and BASSM in both the 24-bit and the 31-bit addressing modes, but it cannot be used to return if the program was called by using BAL in the 24-bit addressing mode.

### Other BALR and BASR Examples

The BALR or BASR instruction with the $R_2$ field set to zero may be used to load a register for use as a base register. For example, in the assembler language, the two statements:

```
BALR    15,0
USING   *,15
```

or

```
BASR    15,0
USING   *,15
```

indicate that the address of the next sequential instruction following the BALR or BASR instruction will be placed in register 15, and that the assembler may use register 15 as a base register until otherwise instructed. (The USING statement is an “assembler instruction” and is thus not a part of the object program.)

## BRANCH AND STACK (BAKR)

The semiprivileged BRANCH AND STACK instruction facilitates linkage between subroutines by saving status in a linkage-stack state entry (sometimes called a branch state entry to distinguish it from a program-call state entry). When BRANCH AND STACK has been used, the return from the called program is made by means of the


LzLabs GmbH. Ex. 1008-857

PROGRAM RETURN instruction. PROGRAM RETURN restores access registers 2-14, general registers 2-14, and the PSW with values saved in the state entry, except that it leaves the PER mask unchanged and sets the condition code to an unpredictable value. The use of BRANCH AND STACK is discussed in "Branching Using the Linkage Stack" on page 5-62.

BRANCH AND STACK can be used to perform a calling linkage, or it can be used at or near the entry point of the called program, depending on whether the $R_1$ field of the instruction is zero or nonzero, respectively. If the $R_1$ field is zero, bits 32-63 of the PSW saved in the state entry indicate the current addressing mode (24-bit or 31-bit) and the address of the next sequential instruction after the BRANCH AND STACK instruction or an EXECUTE instruction. If the $R_1$ field is nonzero, bits 32-63 of the PSW saved in the state entry are set with a value generated from the contents of general register $R_1$: bit 32 of the PSW is set equal to bit 0 of the register, and bits 1-31 of the PSW are set with an address generated from bits 1-31 of the register under the control of bit 0 of the register. Bits 32-63 of the PSW saved in the state entry are referred to in the following examples as the return value.

The branch address for the instruction is generated from the contents of general register $R_2$ under the control of the current addressing mode. Bit 0 of general register $R_2$ does not affect the operation. If the $R_2$ field of the instruction is zero, the operation is performed without branching.

In addition to saving a complete PSW (except with an unpredictable PER mask) in the state entry, BRANCH AND STACK saves the new value of bits 32-63 of the current PSW in the state entry. Bits 32-63 are referred to in the following examples as the branch value.

The following examples contain cases in which bit 32 of the current PSW is either zero or one (24-bit or 31-bit addressing) before BRANCH AND STACK is executed and in which bit 0 of the general register designated by a nonzero $R_1$ or $R_2$ field is either zero or one.

## BAKR Example 1

This example shows BAKR used in a calling program. BAKR performs a branch, and the return is to be to the next sequential instruction.

The format of the BAKR instruction is:

Machine Format

| Op Code | | $R_1$ | $R_2$ |
|---|---|---|---|
| B240 | | 0 | 6 |

Assembler Format

```
Op Code  R1,R2
-------------
  BAKR   0,6
```

Assume four cases of initial values, as follows:

```
    PSW (32-63)  Register 6

1.  00 00 10 D6  02 46 8A CE
2.  00 00 10 D6  82 46 8A CE
3.  80 00 10 D6  02 46 8A CE
4.  80 00 10 D6  82 46 8A CE
```

The results in the four cases are as follows:

```
    Return       Branch Value
    Value        and PSW (32-63)

1.  00 00 10 DA  00 46 8A CE
2.  00 00 10 DA  00 46 8A CE
3.  80 00 10 DA  82 46 8A CE
4.  80 00 10 DA  82 46 8A CE
```

## BAKR Example 2

This example shows BAKR used in a called program. BAKR does not perform a branch, and the return is to be as specified in general register $R_1$.

The format of the BAKR instruction is:

Machine Format

| Op Code | | $R_1$ | $R_2$ |
|---|---|---|---|
| B240 | | 5 | 0 |

Assembler Format

```
Op Code  R1,R2
-------------
  BAKR   5,0
```



LzLabs GmbH. Ex. 1008-858

Assume four cases of initial values, as follows:

```
    Register 5   PSW (32-63)

1.  04 00 10 D6  00 46 8A CE
2.  04 00 10 D6  82 46 8A CE
3.  84 00 10 D6  00 46 8A CE
4.  84 00 10 D6  82 46 8A CE
```

The results in the four cases are as follows:

```
    Return       Branch Value
    Value        and PSW (32-63)

1.  00 00 10 D6  00 46 8A D2
2.  00 00 10 D6  82 46 8A D2
3.  84 00 10 D6  00 46 8A D2
4.  84 00 10 D6  82 46 8A D2
```

### BAKR Example 3

This example shows BAKR used in a called program. BAKR performs a branch, and the return is to be as specified in general register $R_1$.

The format of the BAKR instruction is:

Machine Format

| Op Code | | $R_1$ | $R_2$ |
|---|---|---|---|
| B240 | | 5 | 6 |

Assembler Format

```
Op Code  R1,R2
-------------
  BAKR   5,6
```

Assume eight cases of initial values, as follows:

```
    Register 5   Register 6   PSW (32-63)

1.  04 00 10 D6  06 99 99 00  00 46 8A CE
2.  04 00 10 D6  06 99 99 00  82 46 8A CE
3.  04 00 10 D6  86 99 99 00  00 46 8A CE
4.  04 00 10 D6  86 99 99 00  82 46 8A CE
5.  84 00 10 D6  06 99 99 00  00 46 8A CE
6.  84 00 10 D6  06 99 99 00  82 46 8A CE
7.  84 00 10 D6  86 99 99 00  00 46 8A CE
8.  84 00 10 D6  86 99 99 00  82 46 8A CE
```

The results in the eight cases are as follows:

```
    Return       Branch Value
    Value        and PSW (32-63)

1.  00 00 10 D6  00 99 99 00
2.  00 00 10 D6  86 99 99 00
3.  00 00 10 D6  00 99 99 00
4.  00 00 10 D6  86 99 99 00
5.  84 00 10 D6  00 99 99 00
6.  84 00 10 D6  86 99 99 00
7.  84 00 10 D6  00 99 99 00
8.  84 00 10 D6  86 99 99 00
```

## BRANCH ON CONDITION (BC, BCR)

The BRANCH ON CONDITION instruction tests the condition code to see whether a branch should or should not occur. The branch occurs only if the current condition code corresponds to a one bit in a mask specified by the instruction.

| Condition Code | Instruction (Mask) Bit | Mask Value |
|---|---|---|
| 0 | 8 | 8 |
| 1 | 9 | 4 |
| 2 | 10 | 2 |
| 3 | 11 | 1 |

For example, assume that an ADD (A or AR) operation has been performed and that a branch to address 6050 is desired if the sum is zero or less (condition code is 0 or 1). Also assume:

Register 10 contains 00 00 50 00.

Register 11 contains 00 00 10 00.

The RX form of the instruction performs the required test (and branch if necessary) when written as:

Machine Format

| Op Code | $M_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 47 | C | B | A | 050 |

Assembler Format

```
Op Code  M1,D2(X2,B2)
--------------------
   BC    12,X'50'(11,10)
```

A mask of $12_{10}$ means that there are ones in instruction bits 8 and 9 and zeros in bits 10 and 11, so that branching takes place when the condition code is either 0 or 1.



LzLabs GmbH. Ex. 1008-859

A mask of 15 would indicate a branch on any condition (an unconditional branch). A mask of zero would indicate that no branch is to occur (a no-operation).

(See also “Linkage Instructions (BAL, BALR, BAS, BASR, BASSM, BSM)” on page A-8 for an example of the BCR instruction.)

## BRANCH ON COUNT (BCT, BCTR)

The BRANCH ON COUNT instruction is often used to execute a program loop for a specified number of times. For example, assume that the following represents some lines of coding in an assembler-language program:

```
⋮
LUPE  AR   8,1
⋮
BACK  BCT  6,LUPE
⋮
```

where register 6 contains 00 00 00 03 and the address of LUPE is 6826. Assume that, in order to address this location, register 10 is used as a base register and contains 00 00 68 00.

The format of the BCT instruction is:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 46 | 6 | 0 | A | 026 |

Assembler Format

```
Op Code  R1,D2(X2,B2)
BCT      6,X'26'(0,10)
```

The effect of the coding is to execute three times the loop defined by the instructions labeled LUPE through BACK, while register 6 is decremented from three to zero.

## BRANCH ON INDEX HIGH (BXH)

### BXH Example 1

The BRANCH ON INDEX HIGH instruction is an index-incrementing and loop-controlling instruction that causes a branch whenever the sum of an index value and an increment value is greater than some compare value. For example, assume that:

Register 4 contains 00 00 00 8A = $138_{10}$ = the index.

Register 6 contains 00 00 00 02 = $2_{10}$ = the increment.

Register 7 contains 00 00 00 AA = $170_{10}$ = the compare value.

Register 10 contains 00 00 71 30 = the branch address.

The format of the BXH instruction is:

Machine Format

| Op Code | $R_1$ | $R_3$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 86 | 4 | 6 | A | 000 |

Assembler Format

```
Op Code  R1,R3,D2(B2)
BXH      4,6,0(10)
```

When the instruction is executed, first the contents of register 6 are added to register 4, second the sum is compared with the contents of register 7, and third the decision whether to branch is made. After execution:

Register 4 contains 00 00 00 8C = $140_{10}$.

Registers 6 and 7 are unchanged.

Since the new value in register 4 is not yet greater than the value in register 7, the branch to address 7130 is not taken. Repeated use of the instruction will eventually cause the branch to be taken when the value in register 4 reaches $172_{10}$.

### BXH Example 2

When the register used to contain the increment is odd, that register also becomes the compare-value register. The following assembler-language subroutine illustrates how this may be used to search a table.



LzLabs GmbH. Ex. 1008-860

| Table | |
|---|---|
| 2 Bytes | 2 Bytes |
| ARG1<br>ARG2<br>ARG3<br>ARG4<br>ARG5<br>ARG6 | FUNCT1<br>FUNCT2<br>FUNCT3<br>FUNCT4<br>FUNCT5<br>FUNCT6 |

Assume that:

Register 8 contains the search argument.

Register 9 contains the width of the table in bytes (00 00 00 04).

Register 10 contains the length of the table in bytes (00 00 00 18).

Register 11 contains the starting address of the table.

Register 14 contains the return address to the main program.

As the following subroutine is executed, the argument in register 8 is successively compared with the arguments in the table, starting with argument 6 and working backward to argument 1. If an equality is found, the corresponding function replaces the argument in register 8. If an equality is not found, zero replaces the argument in register 8.

```
SEARCH    LNR  9,9
NOTEQUAL  BXH  10,9,LOOP
NOTFOUND  SR   8,8
          BCR  15,14
LOOP      CH   8,0(10,11)
          BC   7,NOTEQUAL
          LH   8,2(10,11)
          BCR  15,14
```

The first instruction (LNR) causes the value in register 9 to be made negative. After execution of this instruction, register 9 contains FF FF FF FC = $-4_{10}$. Considering the case when no equality is found, the BXH instruction will be executed seven times. Each time BXH is executed, a value of -4 is added to register 10, thus reducing the value in register 10 by 4. The new value in register 10 is compared with the -4 value in register 9. The branch is taken each time until the value in register 10 is -4. Then the branch is not taken, and the SR instruction sets register 8 to zero.

## BRANCH ON INDEX LOW OR EQUAL (BXLE)

The BRANCH ON INDEX LOW OR EQUAL instruction performs the same operation as BRANCH ON INDEX HIGH, except that branching occurs when the sum is lower than or equal to (instead of higher than) the compare value. As the instruction which increments and tests an index value in a program loop, BXLE is useful at the end of the loop and BXH at the beginning. The following assembler-language routines illustrate loops with BXLE.

### BXLE Example 1

Assume that a group of ten 32-bit signed binary integers are stored at consecutive locations, starting at location GROUP. The integers are to be added together, and the sum is to be stored at location SUM.

```
          SR   5,5         Set sum to zero
          LA   6,GROUP     Load first address
          SR   7,7         Set index to zero
          LA   8,4         Load increment 4
          LA   9,39        Load compare value
LOOP      A    5,0(7,6)    Add integer to sum
          BXLE 7,8,LOOP    Test end of loop
          ST   5,SUM       Store sum
```

The two-instruction loop contains an ADD (A) instruction which adds each integer to the contents of general register 5. The ADD instruction uses the contents of general register 7 as an index value to modify the starting address obtained from register 6. Next, BXLE increments the index value by 4, the increment previously loaded into register 8, and compares it with the compare value in register 9, the odd register of this even-odd pair. The compare value was previously set to 39, which is one less than the number of bytes in the data area; this is also the address, relative to the starting address, of the rightmost byte of the last integer to be added. When the last integer has been added, BXLE increments the index value to the next relative address (40), which is found to be greater than the compare value (39) so that no branching takes place.


LzLabs GmbH. Ex. 1008-861

### BXLE Example 2

The technique illustrated in Example 1 is restricted to loops containing instructions in the RX instruction format. That format allows both a base register and an index register to be specified (double indexing).

For instructions in other formats, where an index register cannot be specified, the previous technique may be modified by having the address itself serve as the index value in a BXLE instruction and by using as the compare value the address of the last byte rather than its relative address. The base register then provides the address directly at each iteration of the loop, and it is not necessary to specify a second register to hold the index value (single indexing).

In the following example, an AND (NI) instruction in the SI instruction format sets to zero the rightmost bit of each of the same group of integers as in Example 1, thus making all of them even. The $I_2$ field of the NI instruction contains the byte X'FE', which consists of seven ones and a zero. That byte is ANDed into byte 3, the rightmost byte, of each of the integers in turn.

```
         LA   6,GROUP    Load first address
         LA   8,4        Load increment 4
         LA   9,GROUP+39 Load compare value
    LOOP NI   3(6),X'FE' AND immediate
         BXLE 6,8,LOOP   Test end of loop
```

The technique shown in Example 2 does not work, however, on an ESA/370 system when it is in the 31-bit addressing mode and the data is located at the rightmost end of a 31-bit address space. In this case, the compare value would be set to $2^{31}$-1, which is the largest possible 32-bit signed binary value. The reason the technique does not work is that the BXLE and BXH instructions treat their operands as 32-bit signed binary integers. When the address in general register 6 reaches the value $2^{31}$-4, BXLE increments it to a value that is interpreted as -$2^{31}$, rather than $2^{31}$, and the comparison remains low, which causes looping to continue indefinitely.

This situation can be avoided by not allowing data areas to extend to the rightmost location in a 31-bit address space or by using other techniques; these may include double indexing when possible, as in Example 1, or starting at the end and stepping downward through the data area with a negative increment.

## COMPARE AND FORM CODEWORD (CFC)

See “Sorting Instructions” on page A-51.

## COMPARE HALFWORD (CH)

The COMPARE HALFWORD instruction compares a 16-bit signed binary integer in storage with the contents of a register. For example, assume that:

Register 4 contains FF FF 80 00 = -32,768$_{10}$.

Register 13 contains 00 01 60 50.

Storage locations 16080-16081 contain 8000 = -32,768$_{10}$.

When the instruction:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 49 | 4 | 0 | D | 030 |

Assembler Format

| Op Code | $R_1$,$D_2$($X_2$,$B_2$) |
|---|---|
| CH | 4,X'30'(0,13) |

is executed, the contents of locations 16080-16081 are fetched, expanded to 32 bits (the sign bit is propagated to the left), and compared with the contents of register 4. Because the two numbers are equal, condition code 0 is set.

## COMPARE LOGICAL (CL, CLC, CLI, CLR)

The COMPARE LOGICAL instruction differs from the signed-binary comparison instructions (C, CH, CR) in that all quantities are handled as unsigned binary integers or as unstructured data.

### CLC Example

The COMPARE LOGICAL (CLC) instruction can be used to perform the byte-by-byte comparison of storage fields up to 256 bytes in length. For example, assume that the following two fields of data are in storage:



LzLabs GmbH. Ex. 1008-862

Field 1

| 1886 | | | | | | | | | | | 1891 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| D1 | D6 | C8 | D5 | E2 | D6 | D5 | 6B | C1 | 4B | C2 | 4B |

Field 2

| 1900 | | | | | | | | | | | 190B |
|---|---|---|---|---|---|---|---|---|---|---|---|
| D1 | D6 | C8 | D5 | E2 | D6 | D5 | 6B | C1 | 4B | C3 | 4B |

Also assume:

Register 9 contains 00 00 18 80.

Register 7 contains 00 00 19 00.

Execution of the instruction:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D5 | 0B | 9 | 006 | 7 | 000 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
CLC      6(12,9),0(7)
```

sets condition code 1, indicating that the contents of field 1 are lower in value than the contents of field 2.

Because the collating sequence of the EBCDIC code is determined simply by a logical comparison of the bits in the code, the CLC instruction can be used to collate EBCDIC-coded fields. For example, in EBCDIC, the above two data fields are:

```
Field 1:  JOHNSON,A.B.
Field 2:  JOHNSON,A.C.
```

Condition code 1 indicates that JOHNSON,A.B. should precede JOHNSON,A.C. for the fields to be in alphabetic sequence.

### CLI Example

The COMPARE LOGICAL (CLI) instruction compares a byte from the instruction stream with a byte from storage. For example, assume that:

Register 10 contains 00 00 17 00.

Storage location 1703 contains 7E.

Execution of the instruction:

Machine Format

| Op Code | $I_2$ | $B_1$ | $D_1$ |
|---|---|---|---|
| 95 | AF | A | 003 |

Assembler Format

```
Op Code  D1(B1),I2
CLI      3(10),X'AF'
```

sets condition code 1, indicating that the first operand (the quantity in main storage) is lower than the second (immediate) operand.

### CLR Example

Assume that:

Register 4 contains 00 00 00 01 = 1.

Register 7 contains FF FF FF FF = $2^{32}$ - 1.

Execution of the instruction:

Machine Format

| Op Code | $R_1$ | $R_2$ |
|---|---|---|
| 15 | 4 | 7 |

Assembler Format

```
Op Code  R1,R2
CLR      4,7
```

sets condition code 1. Condition code 1 indicates that the first operand is lower than the second.

If, instead, the signed-binary comparison instruction COMPARE (CR) had been executed, the contents of register 4 would have been interpreted as +1 and the contents of register 7 as -1. Thus, the first operand would have been higher, so that condition code 2 would have been set.

## COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)

The COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) instruction provides a means of comparing bytes selected from a general register to a contiguous field of bytes in storage. The $M_3$ field of the CLM instruction is a



LzLabs GmbH. Ex. 1008-863

four-bit mask that selects zero to four bytes from a general register, each mask bit corresponding, left to right, to a register byte. In the comparison, the register bytes corresponding to ones in the mask are treated as a contiguous field. The operation proceeds left to right. For example, assume that:

Storage locations 10200-10202 contain F0 BC 7B.

Register 12 contains 00 01 00 00.

Register 6 contains F0 BC 5C 7B.

Execution of the instruction:

Machine Format

| Op Code | $R_1$ | $M_3$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| BD | 6 | D | C | 200 |

Assembler Format

```
Op Code  R1,M3,D2(B2)
-------------------------
  CLM    6,B'1101',X'200'(12)
```

causes the following comparison:

```
Register 6:  F0   BC   5C   7B
   Mask M3:   1    1    0    1
              --   --        --

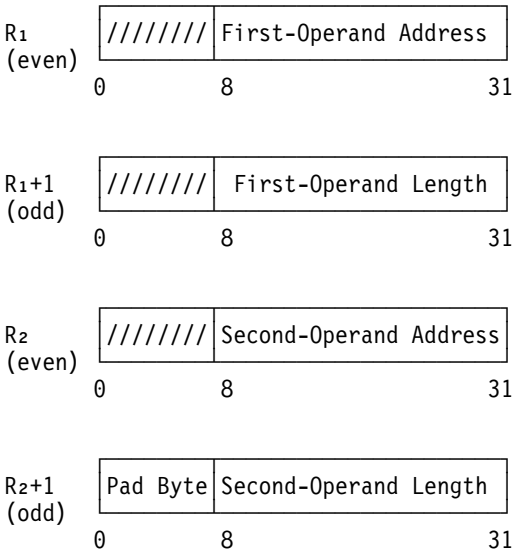
              F0   BC        7B
              |    |         |
Storage       |    |         |
locations     |    |         |
10200-10202:  +----+----+----+
              | F0 | BC | 7B |
              +----+----+----+
```

Because the selected bytes are equal, condition code 0 is set.

## COMPARE LOGICAL LONG (CLCL)

The COMPARE LOGICAL LONG instruction is used to compare two operands in storage, byte by byte. Each operand can be of any length. Two even-odd pairs of general registers (four registers in all) are used to locate the operands and to control the execution of the CLCL instruction, as illustrated in the following diagram. The first register of each pair must be an even register, and it contains the storage address of an operand. The odd register of each pair contains the length of the operand it covers, and the leftmost byte of the second-operand odd register contains a padding byte which is used to extend the shorter operand, if any, to the same length as the longer operand.

The following illustrates the assignment of registers in the 24-bit addressing mode:




In the 31-bit addressing mode, the operand addresses would be in bit positions 1-31 of the even registers shown above.

Since the CLCL instruction may be interrupted during execution, the interrupting program must preserve the contents of the four registers for use when the instruction is resumed.

The following instructions set up two register pairs to control a text-string comparison. For example, assume:

Operand 1
Address: $20800_{16}$
Length: $100_{10}$

Operand 2
Address: $20A00_{16}$
Length: $132_{10}$

Padding Byte
Address: $20003_{16}$
Length: 1
Value: $40_{16}$

Register 12 contains 00 02 00 00.

The setup instructions are:



LzLabs GmbH. Ex. 1008-864

```
LA    4,X'800'(12)      Set register 4 to start of
                        first operand
LA    5,100             Set register 5 to length
                        of first operand
LA    8,X'A00'(12)      Set register 8 to start of
                        second operand
LA    9,132             Set register 9 to length
                        of second operand
ICM   9,B'1000',3(12)   Insert padding byte in
                        leftmost byte position of
                        register 9
```

Register pair 4,5 defines the first operand. Bits 8-31 of register 4 contain the storage address of the start of an EBCDIC text string, and bits 8-31 of register 5 contain the length of the string, in this case 100 bytes.

Register pair 8,9 defines the second operand, with bits 8-31 of register 8 containing the starting location of the second operand and bits 8-31 of register 9 containing the length of the second operand, in this case 132 bytes. Bits 0-7 of register 9 contain an EBCDIC blank character (X'40') to pad the shorter operand. In this example, the padding byte is used in the first operand, after the 100th byte, to compare with the remaining bytes in the second operand.

With the register pairs thus set up, the format of the CLCL instruction is:

Machine Format

| Op Code | $R_1$ | $R_2$ |
|---|---|---|
| 0F | 4 | 8 |

Assembler Format

```
Op Code  R1,R2
-------------
  CLCL    4,8
```

When this instruction is executed, the comparison starts at the left end of each operand and proceeds to the right. The operation ends as soon as an inequality is detected or the end of the longer operand is reached.

If this CLCL instruction is interrupted after 60 bytes have compared equal, the operand lengths in registers 5 and 9 will have been decremented to 40 and 72, respectively. The operand addresses in registers 4 and 8 will have been incremented to X'2083C' and X'20A3C'; the leftmost byte of registers 4 and 8 will have been set to zero. The padding byte X'40' remains in register 9. When the CLCL instruction is reexecuted with these register contents, the comparison resumes at the point of interruption.

Now, assume that the instruction is interrupted after 110 bytes. That is, the first 100 bytes of the second operand have compared equal to the first operand, and the next 10 bytes of the second operand have compared equal to the padding byte (blank). The residual operand lengths in registers 5 and 9 are 0 and 22, respectively, and the operand addresses in registers 4 and 8 are X'20864' (the value when the first operand was exhausted) and X'20A6E' (the current value for the second operand).

When the comparison ends, the condition code is set to 0, 1, or 2, depending on whether the first operand is equal to, less than, or greater than the second operand, respectively.

When the operands are unequal, the addresses in registers 4 and 8 indicate the bytes that caused the mismatch.

## COMPARE LOGICAL STRING (CLST)

The COMPARE LOGICAL STRING instruction is used to compare a first operand designated by general register $R_1$ and a second operand designated by general register $R_2$. The comparison is made left to right, byte by byte, until unequal bytes are compared, an ending character specified in general register 0 is encountered in either operand, or a CPU-determined number of bytes have been compared. The condition code is set to 0 if the two operands are equal, to 1 if the first operand is low, to 2 if the second operand is low, or to 3 if a CPU-determined number of bytes have been compared. If the ending character is found in both operands simultaneously, the operands are equal. If it is found in only one operand, that operand is low.

When condition code 1 or 2 is set, the addresses of the last bytes processed in the first and second operands are placed in general registers $R_1$ and $R_2$, respectively. These are the addresses of unequal bytes in the two operands, or they are the


LzLabs GmbH. Ex. 1008-865

address of an ending character in one operand and of the byte in the corresponding byte position in the other operand. When condition code 3 is set, the addresses of the next bytes to be processed are placed in the registers. When condition code 0 is set, the contents of the registers remain unchanged.

Following are examples of first and second operands beginning at decimal locations 1000 and 2000, respectively. The addresses in general registers $R_1$ and $R_2$ are 1000 and 2000, respectively. The ending character in general register 0 is 00 hex (as in the C programming language). The values of the operand bytes are shown in hex, and the resulting condition code and final contents of general registers $R_1$ and $R_2$ are shown.

**Example 1**
```
1000           2000
C1 C2 C3 00    C1 C2 C3 00

CC: 0; (R1): 1000; (R2): 2000
```

**Example 2**
```
1000           2000
40 40 40 C1    40 40 40 C2

CC: 1; (R1): 1003; (R2): 2003
```

**Example 3**
```
1000           2000
40 40 40 C2    40 40 40 C1

CC: 2; (R1): 1003; (R2): 2003
```

**Example 4**
```
1000           2000
C1 C2 C3 00    C1 C2 C3 C4

CC: 1; (R1): 1003; (R2): 2003
```

**Example 5**
```
1000           2000
C1 C2 C3 C4    C1 C2 C3 00

CC: 2; (R1): 1003; (R2): 2003
```

**Example 6**
```
Assuming that the CPU-determined number of
bytes compared is 256:

1000     1256  2000     2256
40 .. 40 00    40 .. 40 00

CC: 3; (R1): 1256; (R2): 2256
```

**Example 7**
```
1000           2000
00 40 40 40    40 40 40 40

CC: 1; (R1): 1000; (R2): 2000
```

**Example 8**
```
1000           2000
40 40 40 40    00 40 40 40

CC: 2; (R1): 1000; (R2): 2000
```

**Example 9**
```
1000           2000
00 40 40 40    00 40 40 40

CC: 0; (R1): 1000; (R2): 2000
```

## CONVERT TO BINARY (CVB)

The CONVERT TO BINARY instruction converts an eight-byte, packed-decimal number into a signed binary integer and loads the result into a general register. After the conversion operation is completed, the number is in the proper form for use as an operand in signed binary arithmetic. For example, assume:

Storage locations 7608-760F contain a decimal number in the packed format: 00 00 00 00 00 25 59 4C (+25,594).

The contents of register 7 are not significant.

Register 13 contains 00 00 76 00.

The format of the conversion instruction is:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 4F | 7 | 0 | D | 008 |

Assembler Format

```
Op Code  R1,D2(X2,B2)
-------------------------
  CVB    7,8(0,13)
```

After the instruction is executed, register 7 contains 00 00 63 FA.

## CONVERT TO DECIMAL (CVD)

The CONVERT TO DECIMAL instruction is the opposite of the CONVERT TO BINARY instruction. CVD converts a signed binary integer in a register to packed decimal and stores the eight-byte result. For example, assume:

Register 1 contains the signed binary integer: 00 00 0F 0F.

Register 13 contains 00 00 76 00.



LzLabs GmbH. Ex. 1008-866

The format of the instruction is:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 4E | 1 | 0 | D | 008 |

Assembler Format

| Op Code | $R_1$,$D_2$($X_2$,$B_2$) |
|---|---|
| CVD | 1,8(0,13) |

After the instruction is executed, storage locations 7608-760F contain 00 00 00 00 00 03 85 5C (+3855).

The plus sign generated is the preferred plus sign, $1100_2$.

## DIVIDE (D, DR)

The DIVIDE instruction divides the dividend in an even-odd register pair by the divisor in a register or in storage. Since the instruction assumes the dividend to be 64 bits long, it is important first to extend a 32-bit dividend on the left with bits equal to the sign bit. For example, assume that:

Storage locations 3550-3553 contain 00 00 08 DE = $2270_{10}$ (the dividend).

Storage locations 3554-3557 contain 00 00 00 32 = $50_{10}$ (the divisor).

The initial contents of registers 6 and 7 are not significant.

Register 8 contains 00 00 35 50.

The following assembler-language statements load the registers properly and perform the divide operation:

| Statement | Comments |
|---|---|
| L 6,0(0,8) | Places 00 00 08 DE into register 6. |
| SRDA 6,32(0) | Shifts 00 00 08 DE into register 7. Register 6 is filled with zeros (sign bits). |
| D 6,4(0,8) | Performs the division. |

The machine format of the above DIVIDE instruction is:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 5D | 6 | 0 | 8 | 004 |

After the instructions listed above are executed:

Register 6 contains 00 00 00 14 = $20_{10}$ = the remainder.

Register 7 contains 00 00 00 2D = $45_{10}$ = the quotient.

Note that if the dividend had not been first placed in register 6 and shifted into register 7, register 6 might not have been filled with the proper dividend-sign bits (zeros in this example), and the DIVIDE instruction might not have given the expected results.

## EXCLUSIVE OR (X, XC, XI, XR)

When the Boolean operator EXCLUSIVE OR is applied to two bits, the result is one when either, but not both, of the two bits is one; otherwise, the result is zero. When two bytes are EXCLUSIVE ORed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of the EXCLUSIVE OR of two bytes:

| | |
|---|---|
| First-operand byte: | $0011\ 0101_2$ |
| Second-operand byte: | $0101\ 1100_2$ |
| Result byte: | $0110\ 1001_2$ |

### XC Example

The EXCLUSIVE OR (XC) instruction can be used to exchange the contents of two areas in storage without the use of an intermediate storage area. For example, assume two three-byte fields in storage:

| | 359 | | 35B |
|---|---|---|---|
| Field 1 | 00 | 17 | 90 |

| | 360 | | 362 |
|---|---|---|---|
| Field 2 | 00 | 14 | 01 |

Execution of the instruction (assume that register 7 contains 00 00 03 58):



LzLabs GmbH. Ex. 1008-867

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D7 | 02 | 7 | 001 | 7 | 008 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
   XC    1(3,7),8(7)
```

Field 1 is EXCLUSIVE ORed with field 2 as follows:

```
Field 1:  00000000 00010111 10010000₂ = 00 17 90₁₆
Field 2:  00000000 00010100 00000001₂ = 00 14 01₁₆
Result:   00000000 00000011 10010001₂ = 00 03 91₁₆
```

The result replaces the former contents of field 1. Condition code 1 is set to indicate a nonzero result.

Now, execution of the instruction:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D7 | 02 | 7 | 008 | 7 | 001 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
   XC    8(3,7),1(7)
```

produces the following result:

```
Field 1:  00000000 00000011 10010001₂ = 00 03 91₁₆
Field 2:  00000000 00010100 00000001₂ = 00 14 01₁₆
Result:   00000000 00010111 10010000₂ = 00 17 90₁₆
```

The result of this operation replaces the former contents of field 2. Field 2 now contains the original value of field 1. Condition code 1 is set to indicate a nonzero result.

Lastly, execution of the instruction:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D7 | 02 | 7 | 001 | 7 | 008 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
   XC    1(3,7),8(7)
```

produces the following result:

```
Field 1:  00000000 00000011 10010001₂ = 00 03 91₁₆
Field 2:  00000000 00010111 10010000₂ = 00 17 90₁₆
Result:   00000000 00010100 00000001₂ = 00 14 01₁₆
```

The result of this operation replaces the former contents of field 1. Field 1 now contains the original value of field 2. Condition code 1 is set to indicate a nonzero result.

## XI Example

A frequent use of the EXCLUSIVE OR (XI) instruction is to invert a bit (change a zero bit to a one or a one bit to a zero). For example, assume that storage location 8082 contains 0110 $1001_2$. To invert the leftmost and rightmost bits without affecting any of the other bits, the following instruction can be used (assume that register 9 contains 00 00 80 80):

Machine Format

| Op Code | $I_2$ | $B_1$ | $D_1$ |
|---|---|---|---|
| 97 | 81 | 9 | 002 |

Assembler Format

```
Op Code  D1(B1),I2
   XI    2(9),X'81'
```

When the instruction is executed, the byte in storage is EXCLUSIVE ORed with the immediate byte (the $I_2$ field of the instruction):

```
Location 8082:   0110 1001₂
Immediate byte:  1000 0001₂
Result:          1110 1000₂
```

The resulting byte is stored back in location 8082. Condition code 1 is set to indicate a nonzero result.

**Notes:**

1. With the XC instruction, fields up to 256 bytes in length can be exchanged.
2. With the XR instruction, the contents of two registers can be exchanged.
3. Because the X instruction operates storage to



LzLabs GmbH. Ex. 1008-868

register only, an exchange cannot be made solely by the use of X.

4. A field EXCLUSIVE ORed with itself is cleared to zeros.
5. For additional examples of the use of EXCLUSIVE OR, see "Hexadecimal-Floating-Point-Number Conversion" on page A-42.

## EXECUTE (EX)

The EXECUTE instruction causes one *target instruction* in main storage to be executed out of sequence without actually branching to the target instruction. Unless the $R_1$ field of the EXECUTE instruction is zero, bits 8-15 of the target instruction are ORed with bits 24-31 of the $R_1$ register before the target instruction is executed. Thus, EXECUTE may be used to supply the length field for an SS instruction without modifying the SS instruction in storage. For example, assume that a MOVE (MVC) instruction is the target that is located at address 3820, with a format as follows:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D2 | 00 | C | 003 | D | 000 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
MVC      3(1,12),0(13)
```

where register 12 contains 00 00 89 13 and register 13 contains 00 00 90 A0.

Further assume that at storage address 5000, the following EXECUTE instruction is located:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 44 | 1 | 0 | A | 000 |

Assembler Format

```
Op Code  R1,D2(X2,B2)
EX       1,0(0,10)
```

where register 10 contains 00 00 38 20 and register 1 contains 00 0F F0 03.

When the instruction at 5000 is executed, the rightmost byte of register 1 is ORed with the second byte of the target instruction:

```
Instruction byte:   0000 0000₂ = 00
Register byte:      0000 0011₂ = 03
-------------------------------------
Result:             0000 0011₂ = 03
```

causing the instruction at 3820 to be executed as if it originally were:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D2 | 03 | C | 003 | D | 000 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
MVC      3(4,12),0(13)
```

However, after execution:

Register 1 is unchanged.

The instruction at 3820 is unchanged.

The contents of the four bytes starting at location 90A0 have been moved to the four bytes starting at location 8916.

The CPU next executes the instruction at address 5004 (PSW bits 40-63 contain 00 50 04).

## INSERT CHARACTERS UNDER MASK (ICM)

The INSERT CHARACTERS UNDER MASK (ICM) instruction may be used to replace all or selected bytes in a general register with bytes from storage and to set the condition code to indicate the value of the inserted field.

For example, if it is desired to insert a three-byte address from FIELDA into register 5 and leave the leftmost byte of the register unchanged, assume:

Machine Format

| Op Code | $R_1$ | $M_3$ | $S_2$ |
|---|---|---|---|
| BF | 5 | 7 | * * * * |



LzLabs GmbH. Ex. 1008-869

Assembler Format

| Op Code | $R_1$,$M_3$,$S_2$ |
|---|---|
| ICM | 5,B'0111',FIELDA |

```
FIELDA:                  FE DC BA
Register 5 (before):     12 34 56 78
Register 5 (after):      12 FE DC BA
Condition code (after):  1   (leftmost bit of
                              inserted field
                              is one)
```

As another example:

Machine Format

| Op Code | $R_1$ | $M_3$ | $S_2$ |
|---|---|---|---|
| BF | 6 | 9 | * * * * |

Assembler Format

| Op Code | $R_1$,$M_3$,$S_2$ |
|---|---|
| ICM | 6,B'1001',FIELDB |

```
FIELDB:                  12 34
Register 6 (before):     00 00 00 00
Register 6 (after):      12 00 00 34
Condition code (after):  2  (inserted field is
                             nonzero with left-
                             most zero bit)
```

When the mask field contains 1111, the ICM instruction produces the same result as LOAD (L) (provided that the indexing capability of the RX format is not needed), except that ICM also sets the condition code. The condition-code setting is useful when an all-zero field (condition code 0) or a leftmost one bit (condition code 1) is used as a flag.

## LOAD (L, LR)

The LOAD instruction takes four bytes from storage or from a general register and place them unchanged into a general register. For example, assume that the four bytes starting with location 21003 are to be loaded into register 10. Initially:

Register 5 contains 00 02 00 00.

Register 6 contains 00 00 10 03.

The contents of register 10 are not significant.

Storage locations 21003-21006 contain 00 00 AB CD.

To load register 10, the RX form of the instruction can be used:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 58 | A | 5 | 6 | 000 |

Assembler Format

| Op Code | $R_1$,$D_2$($X_2$,$B_2$) |
|---|---|
| L | 10,0(5,6) |

After the instruction is executed, register 10 contains 00 00 AB CD.

## LOAD ADDRESS (LA)

The LOAD ADDRESS instruction provides a convenient way to place a nonnegative binary integer up to $4095_{10}$ in a register without first defining a constant and then using it as an operand. For example, the following instruction places the number $2048_{10}$ in register 1:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 41 | 1 | 0 | 0 | 800 |

Assembler Format

| Op Code | $R_1$,$D_2$($X_2$,$B_2$) |
|---|---|
| LA | 1,2048(0,0) |

The LOAD ADDRESS instruction can also be used to increment a register by an amount up to $4095_{10}$ specified in the $D_2$ field. Depending on the addressing mode, only the rightmost 24 or 31 bits of the sum are retained, however. The leftmost bits of the 32-bit result are set to zeros. For example, assume that register 5 contains 00 12 34 56.

The instruction:



LzLabs GmbH. Ex. 1008-870

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 41 | 5 | 0 | 5 | 00A |

Assembler Format

```
Op Code  R1,D2(X2,B2)
LA       5,10(0,5)
```

adds 10 (decimal) to the contents of register 5 as follows:

```
Register 5 (old): 00 12 34 56
D2 field:         00 00 00 0A
Register 5 (new): 00 12 34 60
```

The register may be specified as either $B_2$ or $X_2$. Thus, the instruction LA 5,10(5,0) produces the same result.

As the most general example, the instruction LA 6,10(5,4) forms the sum of three values: the contents of register 4, the contents of register 5, and a displacement of 10 and places the 24-bit or 31-bit sum with zeros appended on the left in register 6.

## LOAD HALFWORD (LH)

The LOAD HALFWORD instruction places unchanged a halfword from storage into the right half of a register. The left half of the register is loaded with zeros or ones according to the sign (leftmost bit) of the halfword.

For example, assume that the two bytes in storage locations 1803-1804 are to be loaded into register 6. Also assume:

The contents of register 6 are not significant.

Register 14 contains 00 00 18 03.

Locations 1803-1804 contain 00 20.

The instruction required to load the register is:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 48 | 6 | 0 | E | 000 |

Assembler Format

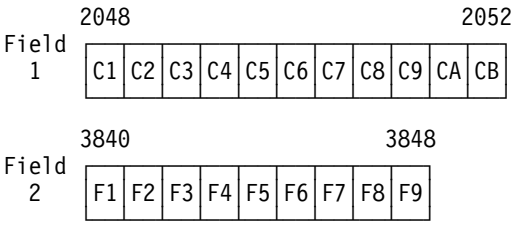
```
Op Code  R1,D2(X2,B2)
LH       6,0(0,14)
```

After the instruction is executed, register 6 contains 00 00 00 20. If locations 1803-1804 had contained a negative number, for example, A7 B6, a minus sign would have been propagated to the left, giving FF FF A7 B6 as the final result in register 6.

## MOVE (MVC, MVI)

### MVC Example

The MOVE (MVC) instruction can be used to move data from one storage location to another. For example, assume that the following two fields are in storage:




Also assume:

Register 1 contains 00 00 20 48.

Register 2 contains 00 00 38 40.

With the following instruction, the first eight bytes of field 2 replace the first eight bytes of field 1:

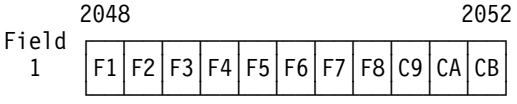
Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D2 | 07 | 1 | 000 | 2 | 000 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
MVC      0(8,1),0(2)
```

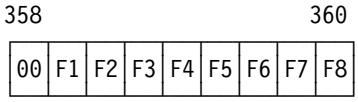
After the instruction is executed, field 1 becomes:




Field 2 is unchanged.

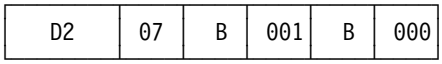
MVC can also be used to propagate a byte through a field by starting the first-operand field one byte location to the right of the second-operand field. For example, suppose that an area in storage starting with address 358 contains the following data:


LzLabs GmbH. Ex. 1008-871




With the following MVC instruction, the zeros in location 358 can be propagated throughout the entire field (assume that register 11 contains 00 00 03 58):

Machine Format




Assembler Format

```
Op Code  D1(L,B1),D2(B2)
MVC      1(8,11),0(11)
```

Because MVC is executed as if one byte were processed at a time, the above instruction, in effect, takes the byte at address 358 and stores it at 359 (359 now contains 00), takes the byte at 359 and stores it at 35A, and so on, until the entire field is filled with zeros. Note that an MVI instruction could have been used originally to place the byte of zeros in location 358.

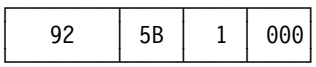
**Notes:**

1. Although the field occupying locations 358-360 contains nine bytes, the length coded in the assembler format is equal to the number of moves (one less than the field length).
2. The order of operands is important even though only one field is involved.

### MVI Example

The MOVE (MVI) instruction places one byte of information from the instruction stream into storage. For example, the instruction:

Machine Format



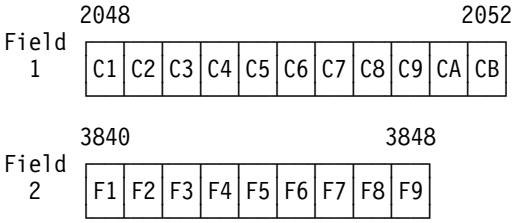

Assembler Format

```
Op Code  D1(B1),I2
MVI      0(1),C'$'
```

may be used, in conjunction with the instruction EDIT AND MARK, to insert the EBCDIC code for a dollar symbol at the storage address contained in general register 1 (see also the example for EDIT AND MARK).

## MOVE INVERSE (MVCIN)

The MOVE INVERSE (MVCIN) instruction can be used to move data from one storage location to another while reversing the order of the bytes within the field. For example, assume that the following two fields are in storage:



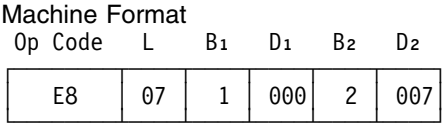

Also assume:

Register 1 contains 00 00 20 48.

Register 2 contains 00 00 38 40.

With the following instruction, the first eight bytes of field 2 replace the first eight bytes of field 1:

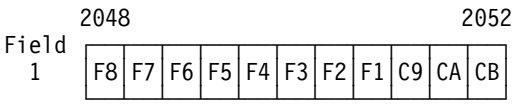
Machine Format




Assembler Format

```
Op Code  D1(L,B1),D2(B2)
MVCIN    0(8,1),7(2)
```

After the instruction is executed, field 1 becomes:




Field 2 is unchanged.

**Note:** This example uses the same general registers, storage locations, and original values as the first example for MVC. For MVCIN, the second-operand address must designate the rightmost byte of the field to be moved, in this case location 3847. This is accomplished by means of the 7 in the $D_2$ field of the instruction.



LzLabs GmbH. Ex. 1008-872

## MOVE LONG (MVCL)

The MOVE LONG (MVCL) instruction can be used for moving data in storage as in the first example of the MVC instruction, provided that the two operands do not overlap. MVCL differs from MVC in that the address and length of each operand are specified in an even-odd pair of general registers. Consequently, MVCL can be used to move more than 256 bytes of data with one instruction. As an example, assume:

Register 2 contains 00 0A 00 00.

Register 3 contains 00 00 08 00.

Register 8 contains 00 06 00 00.

Register 9 contains 00 00 08 00.

Execution of the instruction:

Machine Format

| Op Code | $R_1$ | $R_2$ |
|---|---|---|
| 0E | 8 | 2 |

Assembler Format

| Op Code | $R_1$,$R_2$ |
|---|---|
| MVCL | 8,2 |

moves $2{,}048_{10}$ bytes from locations A0000-A07FF to locations 60000-607FF. Assuming that the CPU is in the 24-bit addressing mode, bits 8-31 of registers 2 and 8 are incremented by $800_{16}$, and bits 0-7 of registers 2 and 8 are set to zeros. Bits 8-31 of registers 3 and 9 are decremented to zero. Condition code 0 is set to indicate that the operand lengths are equal.

If register 3 had contained F0 00 04 00, only the $1{,}024_{10}$ bytes from locations A0000-A03FF would have been moved to locations 60000-603FF. The remaining locations 60400-607FF of the first operand would have been filled with 1,024 copies of the padding byte X'F0', as specified by the leftmost byte of register 3. Bits 8-31 of register 2 would have been incremented by $400_{16}$, bits 8-31 of register 8 would have been incremented by $800_{16}$, and bits 0-7 of registers 2 and 8 would have been set to zeros. Bits 8-31 of registers 3 and 9 would still have been decremented to zero. Condition code 2 would have been set to indicate that the first operand was longer than the second.

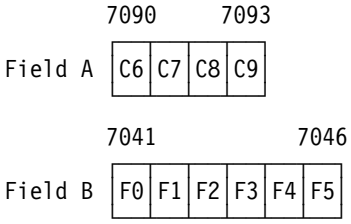
The technique for setting a field to zeros that is illustrated in the second example of MVC cannot be used with MVCL. If the registers were set up to attempt such an operation with MVCL, no data movement would take place and condition code 3 would indicate destructive overlap.

Instead, MVCL may be used to clear a storage area to zeros as follows. Assume register 8 and 9 are set up as before. Register 3 contains only zeros, specifying zero length for the second operand and a zero padding byte. Register 2 is not used to access storage, and its contents are not significant. Executing the instruction MVCL 8,2 causes locations 60000-607FF to be filled with zeros. Bits 8-31 of register 8 are incremented by $800_{16}$, and bits 0-7 of registers 2 and 8 are set to zeros. Bits 8-31 of register 9 are decremented to zero, and condition code 2 is set to indicate that the first operand is longer than the second.

## MOVE NUMERICS (MVN)

Two related instructions, MOVE NUMERICS and MOVE ZONES, may be used with decimal data in the zoned format to operate separately on the rightmost four bits (the numeric bits) and the leftmost four bits (the zone bits) of each byte. Both are similar to MOVE (MVC), except that MOVE NUMERICS moves only the numeric bits and MOVE ZONES moves only the zone bits.

To illustrate the operation of the MOVE NUMERICS instruction, assume that the following two fields are in storage:




Also assume:

Register 14 contains 00 00 70 90.

Register 15 contains 00 00 70 40.

After the instruction:



LzLabs GmbH. Ex. 1008-873

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D1 | 03 | F | 001 | E | 000 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
-------------------------
  MVN    1(4,15),0(14)
```

is executed, field B becomes:

| 7041 | | | | | 7046 |
|---|---|---|---|---|---|
| F6 | F7 | F8 | F9 | F4 | F5 |

The numeric bits of the bytes at locations 7090-7093 have been stored in the numeric bits of the bytes at locations 7041-7044. The contents of locations 7090-7093 and 7045-7046 are unchanged.

## MOVE STRING (MVST)

The MOVE STRING instruction is used to move a second operand designated by general register $R_2$ to a first-operand location designated by general register $R_1$. The movement is made left to right until an ending character specified in general register 0 has been moved or a CPU-determined number of bytes have been moved. The condition code is set to 1 if the ending character was moved or to 3 if a CPU-determined number of bytes were moved.

When condition code 1 is set, the address of the ending character in the first operand is placed in general register $R_1$, and the contents of general register $R_2$ remain unchanged. When condition code 3 is set, the address of the next byte to be processed in the first and second operands is placed in general registers $R_1$ and $R_2$, respectively.

Following is an example program that sets string A equal to the concatenation of string B followed by string C, where the length of each of strings B and C is unknown, and the end of each of strings B and C is indicated by an ending character of 00 hex (as in the C programming language). The program is not written for execution in the access-register mode.

```
         L     4,STRAADR
         L     5,STRBADR
         SR    0,0
LOOP1    MVST  4,5
         BC    1,LOOP1
         L     5,STRCADR
LOOP2    MVST  4,5
         BC    1,LOOP2
         [Any instruction]
```

## MOVE WITH OFFSET (MVO)

MOVE WITH OFFSET may be used to shift a packed-decimal number an odd number of digit positions or to concatenate a sign to an unsigned packed-decimal number.

Assume that the three-byte unsigned packed-decimal number in storage locations 4500-4502 is to be moved to locations 5600-5603 and given the sign of the packed-decimal number ending at location 5603. Also assume:

Register 12 contains 00 00 56 00.

Register 15 contains 00 00 45 00.

Storage locations 5600-5603 contain 77 88 99 0C.

Storage locations 4500-4502 contain 12 34 56.

After the instruction:

Machine Format

| Op Code | $L_1$ | $L_2$ | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|---|
| F1 | 3 | 2 | C | 000 | F | 000 |

Assembler Format

```
Op Code  D1(L1,B1),D2(L2,B2)
---------------------------
  MVO    0(4,12),0(3,15)
```

is executed, the storage locations 5600-5603 contain 01 23 45 6C. Note that the second operand is extended on the left with one zero to fill out the first-operand field.



LzLabs GmbH. Ex. 1008-874

## MOVE ZONES (MVZ)

The MOVE ZONES instruction can operate on overlapping or nonoverlapping fields, as can the instructions MOVE (MVC) and MOVE NUMERICS. When operating on nonoverlapping fields, MOVE ZONES works like the MOVE NUMERICS instruction (see its example), except that MOVE ZONES moves only the zone bits of each byte. To illustrate the use of MOVE ZONES with overlapping fields, assume that the following data field is in storage:

| 800 | | | | | 805 |
|---|---|---|---|---|---|
| F1 | C2 | F3 | C4 | F5 | C6 |

Also assume that register 15 contains 00 00 08 00. The instruction:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| D3 | 04 | F | 001 | F | 000 |

Assembler Format

| Op Code | $D_1$(L,$B_1$),$D_2$($B_2$) |
|---|---|
| MVZ | 1(5,15),0(15) |

propagates the zone bits from the byte at address 800 through the entire field, so that the field becomes:

| 800 | | | | | 805 |
|---|---|---|---|---|---|
| F1 | F2 | F3 | F4 | F5 | F6 |

## MULTIPLY (M, MR)

Assume that a number in register 5 is to be multiplied by the contents of a four-byte field at address 3750. Initially:

The contents of register 4 are not significant.

Register 5 contains 00 00 00 9A = $154_{10}$ = the multiplicand.

Register 11 contains 00 00 06 00.

Register 12 contains 00 00 30 00.

Storage locations 3750-3753 contain 00 00 00 83 = $131_{10}$ = the multiplier.

The instruction required for performing the multiplication is:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 5C | 4 | B | C | 150 |

Assembler Format

| Op Code | $R_1$,$D_2$($X_2$,$B_2$) |
|---|---|
| M | 4,X'150'(11,12) |

After the instruction is executed, the product is in the register pair 4 and 5:

Register 4 contains 00 00 00 00.

Register 5 contains 00 00 4E CE = $20{,}174_{10}$.

Storage locations 3750-3753 are unchanged.

The RR format of the instruction can be used to square the number in a register. Assume that register 7 contains 00 01 00 05. The contents of register 6 are not significant. The instruction:

Machine Format

| Op Code | $R_1$ | $R_2$ |
|---|---|---|
| 1C | 6 | 7 |

Assembler Format

| Op Code | $R_1$,$R_2$ |
|---|---|
| MR | 6,7 |

multiplies the number in register 7 by itself and places the result in the pair of registers 6 and 7:

Register 6 contains 00 00 00 01.

Register 7 contains 00 0A 00 19.

## MULTIPLY HALFWORD (MH)

The MULTIPLY HALFWORD instruction is used to multiply the contents of a register by a two-byte field in storage. For example, assume that:

Register 11 contains 00 00 00 15 =$21_{10}$ = the multiplicand.

Register 14 contains 00 00 01 00.

Register 15 contains 00 00 20 00.



LzLabs GmbH. Ex. 1008-875

Storage locations 2102-2103 contain FF D9 = $-39_{10}$ = the multiplier.

The instruction:

Machine Format

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 4C | B | E | F | 002 |

Assembler Format

| Op Code | $R_1$,$D_2$($X_2$,$B_2$) |
|---|---|
| MH | 11,2(14,15) |

multiplies the two numbers. The product, FF FF FC CD = $-819_{10}$, replaces the original contents of register 11.

Only the rightmost 32 bits of a product are stored in a register; any significant bits on the left are lost. No program interruption occurs on overflow.

## OR (O, OC, OI, OR)

When the Boolean operator OR is applied to two bits, the result is one when either bit is one; otherwise, the result is zero. When two bytes are ORed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of ORing two bytes:

```
First-operand byte:   0011 0101₂
Second-operand byte:  0101 1100₂
-------------------------------
Result byte:          0111 1101₂
```

### OI Example

A frequent use of the OR instruction is to set a particular bit to one. For example, assume that storage location 4891 contains 0100 $0010_2$. To set the rightmost bit of this byte to one without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

| Op Code | $I_2$ | $B_1$ | $D_1$ |
|---|---|---|---|
| 96 | 01 | 8 | 001 |

Assembler Format

| Op Code | $D_1$($B_1$),$I_2$ |
|---|---|
| OI | 1(8),X'01' |

When this instruction is executed, the byte in storage is ORed with the immediate byte (the $I_2$ field of the instruction):

```
Location 4891:     0100 0010₂
Immediate byte:    0000 0001₂
-----------------------------
Result:            0100 0011₂
```

The resulting byte with bit 7 set to one is stored back in location 4891. Condition code 1 is set.

## PACK (PACK)

Assume that storage locations 1000-1003 contain the following zoned-decimal number that is to be converted to a packed-decimal number and left in the same location:

| | 1000 | | | 1003 |
|---|---|---|---|---|
| Zoned number | F1 | F2 | F3 | C4 |

Also assume that register 12 contains 00 00 10 00. After the instruction:

Machine Format

| Op Code | $L_1$ | $L_2$ | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|---|
| F2 | 3 | 3 | C | 000 | C | 000 |

Assembler Format

| Op Code | $D_1$($L_1$,$B_1$),$D_2$($L_2$,$B_2$) |
|---|---|
| PACK | 0(4,12),0(4,12) |

is executed, the result in locations 1000-1003 is in the packed-decimal format:

| | 1000 | | | 1003 |
|---|---|---|---|---|
| Packed number | 00 | 01 | 23 | 4C |

**Notes:**

1. This example illustrates the operation of PACK when the first- and second-operand fields overlap completely.
2. During the operation, the second operand was extended on the left with zeros.



LzLabs GmbH. Ex. 1008-876

## SEARCH STRING (SRST)

The SEARCH STRING instruction is used to search a second operand designated by general register $R_2$ for a character specified in general register 0. The length of the second operand is known— the address of the first byte after the second operand is in general register $R_1$.

When the specified character is found, condition code 1 is set, the address of the character is placed in general register $R_1$, and the contents of general register $R_2$ remain unchanged. When the address of the next second-operand byte to be examined equals the address in general register $R_1$, condition code 2 is set, and the contents of general register $R_1$ and $R_2$ remain unchanged. When a CPU-determined number of second-operand bytes have been examined, condition code 3 is set, the address of the next byte to be processed in the second operand is placed in general register $R_2$, and the contents of general register $R_1$ remain unchanged.

### SRST Example 1

Following is an example program that determines the end of string A, as indicated by an ending character equal to 00 hex (as in the C programming language), and then determines the address of the first character equal to C1 hex in the string. The program is based on the assumption that the second operand does not begin at location 0 or wrap around in storage, and, therefore, condition code 2 will not be set by the first SEARCH STRING instruction because of the address in general register 0. The program is not written for execution in the access-register mode.

```
         L     5,STRAADR
         SR    0,0
LOOP1    SRST  0,5
         BC    1,LOOP1
         L     5,STRAADR
         LR    4,0
         LA    0,X'C1'
LOOP2    SRST  4,5
         BC    1,LOOP2
         BC    2,NOTFND
FOUND    [Any instruction]
         ...
NOTFND   [Any instruction]
```

### SRST Example 2

Following is an example program that determines the address of the first character equal to C1 hex in the string A whose length is known. The program is not written for execution in the access-register mode.

```
         L     5,STRAADR
         L     4,STRALEN
         AR    4,5
         LA    0,X'C1'
LOOP1    SRST  4,5
         BC    1,LOOP1
         BC    2,NOTFND
FOUND    [Any instruction]
         ...
NOTFND   [Any instruction]
```

In this example, the value in STRALEN may be a length that either does or does not include an ending character at the end of the string, provided that the ending character is not the character for which the search is made.

## SHIFT LEFT DOUBLE (SLDA)

The SHIFT LEFT DOUBLE instruction shifts the 63 numeric bits of an even-odd register pair to the left, leaving the sign bit unchanged. Thus, the instruction performs an algebraic left shift of a 64-bit signed binary integer.

For example, if the contents of registers 2 and 3 are:

```
00 7F 0A 72   FE DC BA 98 =
00000000 01111111 00001010 01110010
11111110 11011100 10111010 10011000₂
```

The instruction:

Machine Format

| Op Code | $R_1$ | | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 8F | 2 | //// | 0 | 01F |

Assembler Format

```
Op Code  R1,D2(B2)

SLDA     2,31(0)
```

results in registers 2 and 3 both being left-shifted 31 bit positions, so that their new contents are:

```
7F 6E 5D 4C   00 00 00 00 =
01111111 01101110 01011101 01001100
00000000 00000000 00000000 00000000₂
```



LzLabs GmbH. Ex. 1008-877

Because significant bits are shifted out of bit position 1 of register 2, overflow is indicated by setting condition code 3, and, if the fixed-point-overflow mask bit in the PSW is one, a fixed-point-overflow program interruption occurs.

## SHIFT LEFT SINGLE (SLA)

The SHIFT LEFT SINGLE instruction is similar to SHIFT LEFT DOUBLE, except that it shifts only the 31 numeric bits of a single register. Therefore, this instruction performs an algebraic left shift of a 32-bit signed binary integer.

For example, if the contents of register 2 are:

```
00 7F 0A 72 = 00000000 01111111 00001010 01110010₂
```

The instruction:

Machine Format

| Op Code | $R_1$ | | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 8B | 2 | //// | 0 | 008 |

Assembler Format

```
Op Code  R1,D2(B2)
SLA      2,8(0)
```

results in register 2 being shifted left eight bit positions so that its new contents are:

```
7F 0A 72 00 = 01111111 00001010 01110010 00000000₂
```

Condition code 2 is set to indicate that the result is greater than zero.

If a left shift of nine places had been specified, a significant bit would have been shifted out of bit position 1. Condition code 3 would have been set to indicate this overflow and, if the fixed-point-overflow mask bit in the PSW were one, a fixed-point overflow interruption would have occurred.

## STORE CHARACTERS UNDER MASK (STCM)

STORE CHARACTERS UNDER MASK (STCM) may be used to place selected bytes from a register into storage. For example, if it is desired to store a three-byte address from general register 8 into location FIELD3, assume:

Machine Format

| Op Code | $R_1$ | $M_3$ | $S_2$ |
|---|---|---|---|
| BE | 8 | 7 | * * * * |

Register Format

```
Op Code  R1,M3,S2
STCM     8,B'0111',FIELD3
```

```
Register 8:        12 34 56 78
FIELD3 (before):   not significant
FIELD3 (after):    34 56 78
```

As another example:

Machine Format

| Op Code | $R_1$ | $M_3$ | $S_2$ |
|---|---|---|---|
| BE | 9 | 5 | * * * * |

Register Format

```
Op Code  R1,M3,S2
STCM     9,B'0101',FIELD2
```

```
Register 9:       01 23 45 67
FIELD2 (before): not significant
FIELD2 (after):   23 67
```

## STORE MULTIPLE (STM)

Assume that the contents of general registers 14, 15, 0, and 1 are to be stored in consecutive four-byte fields starting with location 4050 and that:

Register 14 contains 00 00 25 63.

Register 15 contains 00 01 27 36.

Register 0 contains 12 43 00 62.

Register 1 contains 73 26 12 57.

Register 6 contains 00 00 40 00.

The initial contents of locations 4050-405F are not significant.

The STORE MULTIPLE instruction allows the use of just one instruction to store the contents of the four registers:

Machine Format

| Op Code | $R_1$ | $R_3$ | $B_2$ | $D_2$ |
|---|---|---|---|---|
| 90 | E | 1 | 6 | 050 |



LzLabs GmbH. Ex. 1008-878

Assembler Format

| Op Code | $R_1,R_3,D_2(B_2)$ |
|---|---|
| STM | 14,1,X'50'(6) |

After the instruction is executed:

Locations 4050-4053 contain 00 00 25 63.

Locations 4054-4057 contain 00 01 27 36.

Locations 4058-405B contain 12 43 00 62.

Locations 405C-405F contain 73 26 12 57.

## TEST UNDER MASK (TM)

The TEST UNDER MASK instruction examines selected bits of a byte and sets the condition code accordingly. For example, assume that:

Storage location 9999 contains FB.

Register 7 contains 00 00 99 90.

Assume the instruction to be:

Machine Format

| Op Code | $I_2$ | $B_1$ | $D_1$ |
|---|---|---|---|
| 91 | C3 | 7 | 009 |

Assembler Format

| Op Code | $D_1(B_1),I_2$ |
|---|---|
| TM | 9(7),B'11000011' |

The instruction tests only those bits of the byte in storage for which the mask bits are ones:

```
FB   = 1111 1011₂
Mask = 1100 0011₂
-------------------
Test = 11xx xx11₂
```

Condition code 3 is set: all selected bits in the test result are ones. (The bits marked “x” are ignored.)

If location 9999 had contained B9, the test would have been:

```
B9   = 1011 1001₂
Mask = 1100 0011₂
-------------------
Test = 10xx xx01₂
```

Condition code 1 is set: the selected bits are both zeros and ones.

If location 9999 had contained 3C, the test would have been:

```
3C   = 0011 1100₂
Mask = 1100 0011₂
-------------------
Test = 00xx xx00₂
```

Condition code 0 is set: all selected bits are zeros.

**Note:** Storage location 9999 remains unchanged.

## TRANSLATE (TR)

The TRANSLATE instruction can be used to translate data from any character code to any other desired code, provided that each character code consists of eight bits or fewer. An appropriate translation table is required in storage.

In the following example, EBCDIC code is translated to ASCII code. The first step is to create a 256-byte table in storage locations 1000-10FF. This table contains the characters of the ASCII code in the sequence of the binary representation of the EBCDIC code; that is, the ASCII representation of a character is placed in storage at the starting address of the table plus the binary value of the EBCDIC representation of the same character.

For simplicity, the example shows only the part of the table containing the decimal digits:

| 10F0 | | | | | | | | | 10F9 |
|---|---|---|---|---|---|---|---|---|---|
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |

Assume that the four-byte field at storage location 2100 contains the EBCDIC code for the digits 1984:

Locations 2100-2103 contain F1 F9 F8 F4.

Register 12 contains 00 00 21 00.

Register 15 contains 00 00 10 00.

As the instruction:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| DC | 03 | C | 000 | F | 000 |



LzLabs GmbH. Ex. 1008-879

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
-------------------------
   TR    0(4,12),0(15)
```

is executed, the binary value of each EBCDIC byte is added to the starting address of the table, and the resulting address is used to fetch an ASCII byte:

```
Table starting address:    1000
First EBCDIC byte:           F1
-------------------------------
Address of ASCII byte:     10F1
```

After execution of the instruction:

Locations 2100-2103 contain 31 39 38 34.

Thus, the ASCII code for the digits 1984 has replaced the EBCDIC code in the four-byte field at storage location 2100.

## TRANSLATE AND TEST (TRT)

The TRANSLATE AND TEST instruction can be used to scan a data field for characters with a special meaning. To indicate which characters have a special meaning, a table similar to the one used for the TRANSLATE instruction is set up, except that zeros in the table indicate characters without any special meaning and nonzero values indicate characters with a special meaning.

Figure A-4 has been set up to distinguish alphameric characters (A to Z and 0 to 9) from blanks, certain special symbols, and all other characters which are considered invalid. EBCDIC coding is assumed. The 256-byte table is assumed stored at locations 2000-20FF.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 200_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 201_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 202_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 203_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 204_ | 04 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 08 | 40 | 0C | 10 | 40 |
| 205_ | 14 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 18 | 1C | 20 | 40 | 40 |
| 206_ | 24 | 28 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 2C | 40 | 40 | 40 | 40 |
| 207_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 30 | 34 | 38 | 3C | 40 |
| 208_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 209_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 20A_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 20B_ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 20C_ | 40 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 40 | 40 | 40 | 40 | 40 | 40 |
| 20D_ | 40 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 40 | 40 | 40 | 40 | 40 | 40 |
| 20E_ | 40 | 40 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 40 | 40 | 40 | 40 | 40 | 40 |
| 20F_ | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 40 | 40 | 40 | 40 | 40 | 40 |

**Note**: If the character codes in the statement being translated occupy a range smaller than 00 through $FF_{16}$, a table of fewer than 256 bytes can be used.

*Figure A-4. Translate and Test Table*

The table entries for the alphameric characters in EBCDIC are 00; thus, the letter A (code C1) corresponds to byte location 20C1, which contains 00.

The 15 special symbols have nonzero entries from $04_{16}$ to $3C_{16}$ in increments of 4. Thus, the blank (code 40) has the entry $04_{16}$, the period (code 4B) has the entry $08_{16}$, and so on.

All other table positions have the entry $40_{16}$ to indicate an invalid character.

The table entries are chosen so that they may be used to select one of a list of 16 words containing addresses of different routines to be entered for each special symbol or invalid character encountered during the scan.

Assume that this list of 16 branch addresses is stored at locations 3004-3043.

Starting at storage location CA80, there is the following sequence of $21_{10}$ EBCDIC characters, where “b” stands for a blank.



LzLabs GmbH. Ex. 1008-880

Locations CA80-CA94:
UNPKbPROUT(9),WORD(5)

Also assume:

Register 1 contains 00 00 CA 7F.

Register 2 contains 00 00 30 00.

Register 15 contains 00 00 20 00.

As the instruction:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| DD | 14 | 1 | 001 | F | 000 |

Assembler Format

```
Op Code  D1(L,B1),D2(B2)
TRT      1(21,1),0(15)
```

is executed, the value of the first source byte, the EBCDIC code for the letter U, is added to the starting address of the table to produce the address of the table entry to be examined:

```
Table starting address      2000
First source byte (U)         E4
Address of table entry      20E4
```

Because zeros were placed in storage location 20E4, no special action occurs. The operation continues with the second and subsequent source bytes until it reaches the blank in location CA84. When this symbol is reached, its value is added to the starting address of the table, as usual:

```
Table starting address      2000
Source byte (blank)           40
Address of table entry      2040
```

Because location 2040 contains a nonzero value, the following actions occur:

The address of the source byte, 00CA84, is placed in the rightmost 24 bits of register 1.

The table entry, 04, is placed in the rightmost eight bits of register 2, which now contains 00 00 30 04.

Condition code 1 is set (scan not completed).

The TRANSLATE AND TEST instruction may be followed by instructions to branch to the routine at the address found at location 3004, which corresponds to the blank character encountered in the scan. When this routine is completed, program control may return to the TRANSLATE AND TEST instruction to continue the scan, except that the length must first be adjusted for the characters already scanned.

For this purpose, the TRANSLATE AND TEST may be executed by the use of an EXECUTE instruction, which supplies the length specification from a general register. In this way, a complete statement scan can be performed with a single TRANSLATE AND TEST instruction used repeatedly by means of EXECUTE, and without modifying any instructions in storage. In the example, after the first execution of TRANSLATE AND TEST, register 1 contains the address of the last source byte translated. It is then a simple matter to subtract this address from the address of the last source byte (CA94) to produce a length specification. This length minus one is placed in the register that is referenced as the $R_1$ field of the EXECUTE instruction. (Note that the length code in the machine format is one less than the total number of bytes in the field.) The second-operand address of the EXECUTE instruction points to the TRANSLATE AND TEST instruction, which is the same as illustrated above, except for the length (L) which is set to zero.

## UNPACK (UNPK)

Assume that storage locations 2501-2502 contain a signed, packed-decimal number that is to be unpacked and placed in storage locations 1000-1004. Also assume:

Register 12 contains 00 00 10 00.

Register 13 contains 00 00 25 00.

Storage locations 2501-2502 contain 12 3D.

The initial contents of storage locations 1000-1004 are not significant.

After the instruction:

Machine Format

| Op Code | $L_1$ | $L_2$ | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|---|
| F3 | 4 | 1 | C | 000 | D | 001 |

Assembler Format

```
Op Code  D1(L1,B1),D2(L2,B2)
UNPK     0(5,12),1(2,13)
```



LzLabs GmbH. Ex. 1008-881

is executed, the storage locations 1000-1004 contain F0 F0 F1 F2 D3.

## UPDATE TREE (UPT)

See “Sorting Instructions” on page A-51.

# Decimal Instructions

(See Chapter 8, “Decimal Instructions” for a complete description of the decimal instructions.)

## ADD DECIMAL (AP)

Assume that the signed, packed-decimal number at storage locations 500-503 is to be added to the signed, packed-decimal number at locations 2000-2002. Also assume:

Register 12 contains 00 00 20 00.

Register 13 contains 00 00 05 00.

Storage locations 2000-2002 contain 38 46 0D (a negative number).

Storage locations 500-503 contain 01 12 34 5C (a positive number).

After the instruction:

Machine Format

| Op Code | $L_1$ | $L_2$ | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|---|
| FA | 2 | 3 | C | 000 | D | 000 |

Assembler Format

```
Op Code  D1(L1,B1),D2(L2,B2)
AP       0(3,12),0(4,13)
```

is executed, the storage locations 2000-2002 contain 73 88 5C; condition code 2 is set to indicate that the result is greater than zero. Note that:

1. Because the two numbers had different signs, they were in effect subtracted.
2. Although the second operand is longer than the first operand, no overflow interruption occurs because the result can be entirely contained within the first operand.

## COMPARE DECIMAL (CP)

Assume that the signed, packed-decimal contents of storage locations 700-703 are to be algebraically compared with the signed, packed-decimal contents of locations 500-502. Also assume:

Register 12 contains 00 00 06 00.

Register 13 contains 00 00 03 00.

Storage locations 700-703 contain 17 25 35 6D.

Storage locations 500-502 contain 72 14 2D.

After the instruction:

Machine Format

| Op Code | $L_1$ | $L_2$ | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|---|
| F9 | 3 | 2 | C | 100 | D | 200 |

Assembler Format

```
Op Code  D1(L1,B1),D2(L2,B2)
CP       X'100'(4,12),X'200'(3,13)
```

is executed, condition code 1 is set, indicating that the first operand (the contents of locations 700-703) is less than the second.

## DIVIDE DECIMAL (DP)

Assume that the signed, packed-decimal number at storage locations 2000-2004 (the dividend) is to be divided by the signed, packed-decimal number at locations 3000-3001 (the divisor). Also assume:

Register 12 contains 00 00 20 00.

Register 13 contains 00 00 30 00.

Storage locations 2000-2004 contain 01 23 45 67 8C.

Storage locations 3000-3001 contain 32 1D.

After the instruction:

Machine Format

| Op Code | $L_1$ | $L_2$ | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|---|
| FD | 4 | 1 | C | 000 | D | 000 |

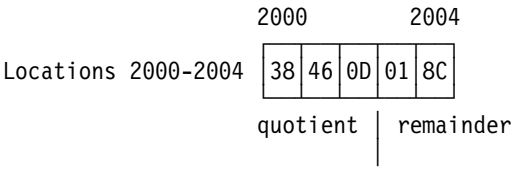


LzLabs GmbH. Ex. 1008-882

Assembler Format

| Op Code | $D_1(L_1,B_1),D_2(L_2,B_2)$ |
|---|---|
| DP | 0(5,12),0(2,13) |

is executed, the dividend is entirely replaced by the signed quotient and remainder, as follows:




**Notes:**

1. Because the dividend and divisor have different signs, the quotient receives a negative sign.
2. The remainder receives the sign of the dividend and the length of the divisor.
3. If an attempt were made to divide the dividend by the one-byte field at location 3001, the quotient would be too long to fit within the four bytes allotted to it. A decimal-divide exception would exist, causing a program interruption.

## EDIT (ED)

Before decimal data in the packed format can be used in a printed report, digits and signs must be converted to printable characters. Moreover, punctuation marks, such as commas and decimal points, may have to be inserted in appropriate places. The highly flexible EDIT instruction performs these functions in a single instruction execution.

This example shows step-by-step one way that the EDIT instruction can be used. The field to be edited (the source) is four bytes long; it is edited against a pattern 13 bytes long. The following symbols are used:

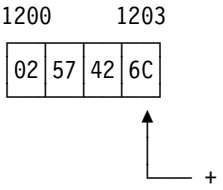
| Symbol | Meaning |
|---|---|
| b (Hexadecimal 40) | Blank character |
| ( (Hexadecimal 21) | Significance starter |
| d (Hexadecimal 20) | Digit selector |

Assume that register 12 contains:

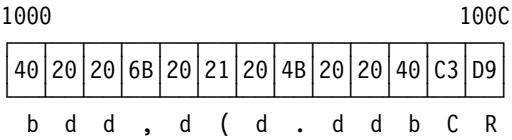
```
00 00 10 00
```

and that the source and pattern fields are:

**Source**




**Pattern**




Execution of the instruction:

Machine Format

| Op Code | L | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| DE | 0C | C | 000 | C | 200 |

Assembler Format

| Op Code | $D_1(L,B_1),D_2(B_2)$ |
|---|---|
| ED | 0(13,12),X'200'(12) |

alters the pattern field as follows:

| Pattern | Digit | Significance Indicator (Before/ After) | Rule | Location 1000-100C |
|---|---|---|---|---|
| b | | off/off | leave(1) | bdd,d(d.ddbCR |
| d | 0 | off/off | fill | bbd,d(d.ddbCR |
| d | 2 | off/on(2) | digit | bb2,d(d.ddbCR |
| , | | on/on | leave | same |
| d | 5 | on/on | digit | bb2,5(d.ddbCR |
| ( | 7 | on/on | digit | bb2,57d.ddbCR |
| d | 4 | on/on | digit | bb2,574.ddbCR |
| . | | on/on | leave | same |
| d | 2 | on/on | digit | bb2,574.2dbCR |
| d | 6+ | on/off(3) | digit | bb2,574.26bCR |
| b | | off/off | fill | same |
| C | | off/off | fill | bb2,574.26bbR |
| R | | off/off | fill | bb2,574.26bbb |

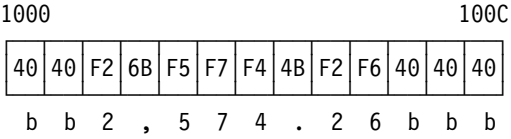
**Notes:**

1. This character is the fill byte.
2. First nonzero decimal source digit turns on significance indicator.
3. Plus sign in the four rightmost bits of the byte turns off significance indicator.


LzLabs GmbH. Ex. 1008-883

Thus, after the instruction is executed, the pattern field contains the result as follows:

**Pattern**



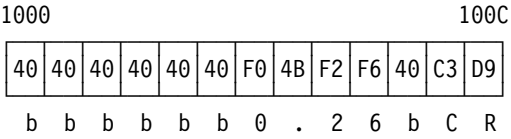

This pattern field prints as:

```
2,574.26
```

The source field remains unchanged. Condition code 2 is set because the number was greater than zero.

If the number in the source field is changed to the negative number 00 00 02 6D and the original pattern is used, the edited result this time is:

**Pattern**




This pattern field prints as:

```
0.26 CR
```

The significance starter forces the significance indicator to the on state and hence causes a leading zero and the decimal point to be preserved. Because the minus-sign code has no effect on the significance indicator, the characters CR are printed to show a negative (credit) amount.

Condition code 1 is set (number less than zero).

## EDIT AND MARK (EDMK)

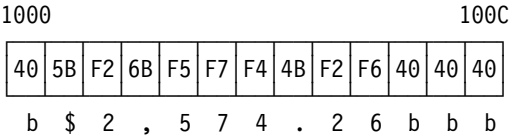
The EDIT AND MARK instruction may be used, in addition to the functions of EDIT, to insert a currency symbol, such as a dollar sign, at the appropriate position in the edited result. Assume the same source in storage locations 1200-1203, the same pattern in locations 1000-100C, and the same contents of general register 12 as for the EDIT instruction above. The previous contents of general register 1 (GR1) are not significant; a LOAD ADDRESS instruction is used to set up the first digit position that is forced to print if no significant digits occur to the left.

The instructions:

| | | |
|---|---|---|
| LA | 1,6(0,12) | Load address of forced significant digit into GR1 |
| EDMK | 0(13,12),X'200'(12) | Leave address of first significant digit in GR1 |
| BCTR | 1,0 | Subtract 1 from address in GR1 |
| MVI | 0(1),C'$' | Store dollar sign at address in GR1 |

produce the following results for the two examples under EDIT:

**Pattern**

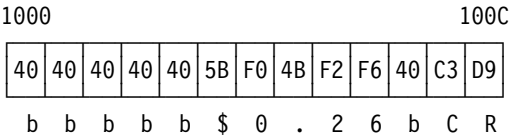



This pattern field prints as:

```
$2,574.26
```

Condition code 2 is set to indicate that the number edited was greater than zero.

**Pattern**



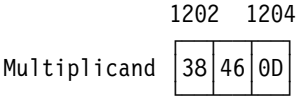

This pattern field prints as:

```
$0.26 CR
```

Condition code 1 is set because the number is less than zero.

## MULTIPLY DECIMAL (MP)

Assume that the signed, packed-decimal number in storage locations 1202-1204 (the multiplicand) is to be multiplied by the signed, packed-decimal number in locations 500-501 (the multiplier).






LzLabs GmbH. Ex. 1008-884

```
                 500 501
Multiplier      | 32 | 1D |
```

The multiplicand must first be extended to have at least two bytes of leftmost zeros, corresponding to the multiplier length, so as to avoid a data exception during the multiplication. ZERO AND ADD can be used to move the multiplicand into a longer field. Assume:

Register 4 contains 00 00 12 00.

Register 6 contains 00 00 05 00.

Then execution of the instruction:

```
ZAP X'100'(5,4),2(3,4)
```

sets up a new multiplicand in storage locations 1300-1304:

```
                        1300                  1304
Multiplicand (new)     | 00 | 00 | 38 | 46 | 0D |
```

Now, after the instruction:

Machine Format

| Op Code | $L_1$ | $L_2$ | $B_1$ | $D_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|---|
| FC | 4 | 1 | 4 | 100 | 6 | 000 |

Assembler Format

```
Op Code  D1(L1,B1),D2(L2,B2)
-----------------------------
   MP  X'100'(5,4),0(2,6)
```

is executed, storage locations 1300-1304 contain the product: 01 23 45 66 0C.

# SHIFT AND ROUND DECIMAL (SRP)

The SHIFT AND ROUND DECIMAL (SRP) instruction can be used for shifting decimal numbers in storage to the left or right. When a number is shifted right, rounding can also be done.

## Decimal Left Shift

In this example, the contents of storage location FIELD1 are shifted three places to the left, effectively multiplying the contents of FIELD1 by 1000. FIELD1 is six bytes long. The following instruction performs the operation:

Machine Format

| Op Code | $L_1$ | $I_3$ | $S_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| F0 | 5 | 0 | **** | 0 | 003 |

Assembler Format

```
Op Code  S1(L1),S2,I3
---------------------
  SRP   FIELD1(6),3,0

FIELD1 (before):  00 01 23 45 67 8C

FIELD1 (after):   12 34 56 78 00 0C
```

The second-operand address in this instruction specifies the shift amount (three places). The rounding digit, $I_3$, is not used in a left shift, but it must be a valid decimal digit. After execution, condition code 2 is set to show that the result is greater than zero.

## Decimal Right Shift

In this example, the contents of storage location FIELD2 are shifted one place to the right, effectively dividing the contents of FIELD2 by 10 and discarding the remainder. FIELD2 is five bytes in length. The following instruction performs this operation:

Machine Format

| Op Code | $L_1$ | $I_3$ | $S_1$ | $B_2$ | $D_2$ |
|---|---|---|---|---|---|
| F0 | 4 | 0 | **** | 0 | 03F |

```
00111111
   |
   6-bit two's
   complement
   for -1
```

Assembler Format

```
Op Code  S1(L1),S2,I3
---------------------
  SRP   FIELD2(5),64-1,0

FIELD 2 (before):  01 23 45 67 8C

FIELD 2 (after):   00 12 34 56 7C
```



LzLabs GmbH. Ex. 1008-885

In the SRP instruction, shifts to the right are specified in the second-operand address by negative shift values, which are represented as a six-bit value in two's complement form.

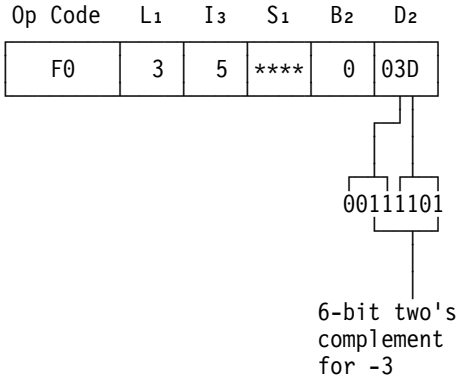
The six-bit two's complement of a number, n, can be specified as 64 - n. In this example, a right shift of one is represented as 64 - 1.

Condition code 2 is set.

### Decimal Right Shift and Round

In this example, the contents of storage location FIELD3 are shifted three places to the right and rounded, in effect dividing by 1000 and rounding up. FIELD3 is four bytes in length.

Machine Format




Assembler Format

```
Op Code  S1(L1),S2,I3
-------------------------
  SRP    FIELD3(4),64-3,5

FIELD 3 (before):  12 39 60 0D

FIELD 3 (after):   00 01 24 0D
```

The shift amount (three places) is specified in the $D_2$ field. The $I_3$ field specifies a rounding digit of 5. The rounding digit is added to the last digit shifted out (which is a 6), and the carry is propagated to the left. The sign is ignored during the addition.

Condition code 1 is set because the result is less than zero.

### Multiplying by a Variable Power of 10

Since the shift value specified by the SRP instruction specifies both the direction and amount of the shift, the operation is equivalent to multiplying the decimal first operand by 10 raised to the power specified by the shift value.

If the shift value is to be variable, it may be specified by the $B_2$ field instead of the displacement $D_2$ of the SRP instruction. The general register designated by $B_2$ should contain the shift value (power of 10) as a signed binary integer.

A fixed scale factor modifying the variable power of 10 may be specified by using both the $B_2$ field (variable part in a general register) and the $D_2$ field (fixed part in the displacement).

The SRP instruction uses only the rightmost six bits of the effective address $D_2(B_2)$ and interprets them as a six-bit signed binary integer to control the left or right shift as in the preceding shift examples.

## ZERO AND ADD (ZAP)

Assume that the signed, packed-decimal number at storage locations 4500-4502 is to be moved to locations 4000-4004 with four leading zeros in the result field. Also assume:

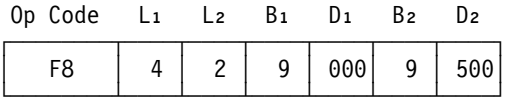
Register 9 contains 00 00 40 00.

Storage locations 4000-4004 contain 12 34 56 78 90.

Storage locations 4500-4502 contain 38 46 0D.

After the instruction:

Machine Format




Assembler Format

```
Op Code  D1(L1,B1),D2(L2,B2)
-----------------------------
  ZAP    0(5,9),X'500'(3,9)
```

is executed, the storage locations 4000-4004 contain 00 00 38 46 0D; condition code 1 is set to indicate a negative result without overflow.



LzLabs GmbH. Ex. 1008-886

Note that, because the first operand is not checked for valid sign and digit codes, it may contain any combination of hexadecimal digits before the operation.

## Hexadecimal-Floating-Point Instructions

(See Chapter 9, "Floating-Point Overview and Support Instructions" for a complete description of the hexadecimal-floating-point instructions.)

In this section, the abbreviations FPR0, FPR2, FPR4, and FPR6 stand for floating-point registers 0, 2, 4, and 6 respectively.

## ADD NORMALIZED (AD, ADR, AE, AER, AXR)

The ADD NORMALIZED instruction performs the addition of two HFP numbers and places the normalized result in a floating-point register. Neither of the two numbers to be added must necessarily be in normalized form before addition occurs. For example, assume that:

FPR6 contains the unnormalized number C3 08 21 00 00 00 00 00 = -82.1$_{16}$ = -130.06$_{10}$ approximately.

Storage locations 2000-2007 contain the normalized number 41 12 34 56 00 00 00 00 = +1.23456$_{16}$ = +1.14$_{10}$ approximately.

Register 13 contains 00 00 20 00.

The instruction:

Machine Format

| Op Code | R$_1$ | X$_2$ | B$_2$ | D$_2$ |
|---|---|---|---|---|
| 7A | 6 | 0 | D | 000 |

Assembler Format

| Op Code | R$_1$,D$_2$(X$_2$,B$_2$) |
|---|---|
| AE | 6,0(0,13) |

performs the short-precision addition of the two operands, as follows.

The characteristics of the two numbers (43 and 41) are compared. Since the number in storage has a characteristic that is smaller by 2, it is right-shifted two hexadecimal digit positions. One guard digit is retained on the right. The fractions of the two numbers are then added algebraically:

| | Fraction | GD[1] |
|---|---|---|
| FPR6 | -43 08 21 00 | |
| Shifted number from storage | +43 00 12 34 | 5 |
| Intermediate sum | -43 08 0E CB | B |
| Left-shifted sum | -42 80 EC BB | |

[1] Guard digit

Because the intermediate sum is unnormalized, it is left-shifted to form the normalized HFP number -80.ECBB$_{16}$ = -128.92$_{10}$ approximately. Combining the sign with the characteristic, the result is C2 80 EC BB, which replaces the left half of FPR6. The right half of FPR6 and the contents of storage locations 2000-2007 are unchanged. Condition code 1 is set to indicate a result less than zero.

If the long-precision instruction AD were used, the result in FPR6 would be C2 80 EC BA A0 00 00 00. Note that use of the long-precision instruction would avoid a loss of precision in this example.

## ADD UNNORMALIZED (AU, AUR, AW, AWR)

The ADD UNNORMALIZED instruction operates the same as the ADD NORMALIZED instruction, except that the final result is not normalized. For example, using the the same operands as in the example for ADD NORMALIZED, when the short-precision instruction:

Machine Format

| Op Code | R$_1$ | X$_2$ | B$_2$ | D$_2$ |
|---|---|---|---|---|
| 7E | 6 | 0 | D | 000 |

Assembler Format

| Op Code | R$_1$,D$_2$(X$_2$,B$_2$) |
|---|---|
| AU | 6,0(0,13) |

is executed, the two numbers are added as follows:



LzLabs GmbH. Ex. 1008-887

```
                                    Fraction GD¹
FPR6                              -43 08 21 00
Shifted number from storage       +43 00 12 34  5
-------------------------------------------------
Intermediate sum                  -43 08 0E CB  B
```

¹ Guard digit

The guard digit participates in the addition but is discarded. The unnormalized sum replaces the left half of FPR6. Condition code 1 is set because the result is less than zero.

The truncated result in FPR6 (C3 08 0E CB 00 00 00 00) shows a loss of a significant digit when compared to the result of short-precision normalized addition.

## COMPARE (CD, CDR, CE, CER)

Assume that FPR4 contains 43 00 00 00 00 00 00 00 (zero), and FPR6 contains 35 12 34 56 78 9A BC DE (a positive number). The contents of the two registers are to be compared using a long-precision COMPARE instruction.

Machine Format

| Op Code | R1 | R2 |
|---|---|---|
| 29 | 4 | 6 |

Assembler Format

```
Op Code  R1,R2
-------------
  CDR    4,6
```

The number with the smaller characteristic, which is in register FPR6, is right-shifted 43 - 35 hex (67 - 53 decimal) or 14 digit positions, so that the two characteristics agree. The shifted number is 43 00 00 00 00 00 00 00, with a guard digit of one. Therefore, when the two numbers are compared, condition code 1 is set, indicating that operand 1 in FPR4 is less than operand 2 in FPR6.

If the example is changed to a second operand with a characteristic of 34 instead of 35, so that FPR6 contains 34 12 34 56 78 9A BC DE, the operand is right-shifted 15 positions, leaving all fraction digits and the guard digit as zeros. Condition code 0 is set, indicating equality. This example shows that two HFP numbers with different characteristics or fractions may compare equal if the numbers are unnormalized or zero.

As another example of comparing unnormalized HFP numbers, 41 00 12 34 56 78 9A BC compares equal to all numbers of the form 3F 12 34 56 78 9A BC 0X (X represents any hexadecimal digit). When the COMPARE instruction is executed, the two rightmost digits are shifted right two places, the 0 becomes the guard digit, and the X does not participate in the comparison.

However, when two normalized HFP numbers are compared, the relationship between numbers that compare equal is unique: each digit in one number must be the same as the corresponding digit in the other number.

## DIVIDE (DD, DDR, DE, DER)

Assume that the first operand (the dividend) is in FPR2 and the second operand (the divisor) in FPR0. If the operands are in the short-precision format, the resulting quotient is returned to FPR2 by the instruction:

Machine Format

| Op Code | R1 | R2 |
|---|---|---|
| 3D | 2 | 0 |

Assembler Format

```
Op Code  R1,R2
-------------
  DER    2,0
```

Several examples of short-precision HFP division, with the dividend in FPR2 and the divisor in FPR0, are shown below. For case A, the result, which replaces the dividend, is obtained in the following steps.



LzLabs GmbH. Ex. 1008-888

```
                  7.2522F
              ┌──────────
.123400 │.821000
          7F6C00
          ──────
           2A400 0
           24680 0
           ──────
            5D80 00
            5B04 00
            ───────
             27C 000
             246 800
             ───────
              35 8000
              24 6800
              ───────
              11 18000
              11 10C00
              ────────
                  7400
```

| Case | FPR2 Before (Dividend) | FPR0 (Divisor) | FPR2 After (Quotient) |
|---|---|---|---|
| A | -43 082100 | +43 001234 | -42 72522F |
| B | +42 101010 | +45 111111 | +3D F0F0F0 |
| C | +48 30000F | +41 400000 | +47 C0003C |
| D | +48 30000F | +41 200000 | +48 180007 |
| E | +48 180007 | +41 200000 | +47 C00038 |

Case C shows a number being divided by 4.0. Case D divides the same number by 2.0, and case E divides the result of case D again by 2.0. The results of cases C and E differ in the rightmost hexadecimal digit position, which illustrates an effect of result truncation.

## HALVE (HDR, HER)

HALVE produces the same result as HFP DIVIDE with a divisor of 2.0. Assume FPR2 contains the long-precision number +48 30 00 00 00 00 00 0F. The following HALVE instruction produces the result +48 18 00 00 00 00 00 07 in FPR2:

Machine Format

| Op Code | $R_1$ | $R_2$ |
|---|---|---|
| 24 | 2 | 2 |

Assembler Format

| Op Code | $R_1$,$R_2$ |
|---|---|
| HDR | 2,2 |

## MULTIPLY (MD, MDR, MDE, MDER, MXD, MXDR, MXR)

For this example, the following long-precision operands are in FPR0 and FPR2:

```
FPR0:  -33 606060 60606060
FPR2:  -5A 200000 20000020
```

A long-precision product is generated by the instruction:

Machine Format

| Op Code | $R_1$ | $R_2$ |
|---|---|---|
| 2C | 0 | 2 |

Assembler Format

| Op Code | $R_1$,$R_2$ |
|---|---|
| MDR | 0,2 |

If the operands were not already normalized, the instruction would first normalize them. It then generates an intermediate result consisting of the full 28-digit hexadecimal product fraction obtained by multiplying the 14-digit hexadecimal operand fractions, together with the appropriate sign and a characteristic that is the sum of the operand characteristics less 64 (40 hex):

The fraction multiplication is performed as follows:

```
                .60606060606060
                .20000020000020
                ──────────────
               C0C0C0C0C0C0C00
          C0C0C0C0C0C0C0
     C0C0C0C0C0C0C0
──────────────────────────────
.0C0C0C181818241818180C0C0C00
```

Attaching the sign and characteristic to the fraction gives:

```
+4D 0C0C0C 18181824 1818180C 0C0C00
```

Because this intermediate product has a leading zero, it is then normalized. The truncated final result placed in FPR0 is:

```
+4C C0C0C1 81818241
```



LzLabs GmbH. Ex. 1008-889

## Hexadecimal-Floating-Point-Number Conversion

The following examples illustrate one method of converting between binary fixed-point numbers (32-bit signed binary integers) and normalized HFP numbers. Conversion must provide for the different representations used with negative numbers: the two's-complement form for signed binary integers, and the signed-absolute-value form for the fractions of HFP numbers.

### Fixed Point to Hexadecimal Floating Point

The method used here inverts the leftmost bit of the 32-bit signed binary integer, which is equivalent to adding $2^{31}$ to the number and considering the result to be positive. This changes the number from a signed integer in the range $2^{31} - 1$ through $-2^{31}$ to an unsigned integer in the range $2^{32} - 1$ through 0. After conversion to the long HFP format, the value $2^{31}$ is subtracted again.

Assume that general register 9 (GR9) contains the integer -59 in two's-complement form:

```
GR9:      FF FF FF C5
```

Further, assume two eight-byte fields in storage: TEMP, for use as temporary storage, and TWO31, which contains the floating-point constant $2^{31}$ in the following format:

```
TWO31:   4E 00 00 00 80 00 00 00
```

This is an unnormalized long HFP number with the characteristic 4E, which corresponds to a radix point (hexadecimal point) to the right of the number.

The following instruction sequence performs the conversion:

| | | **Result** |
|---|---|---|
| X | 9,TWO31+4 | GR9:<br>7FFF FFC5 |
| ST | 9,TEMP+4 | TEMP:<br>xxxx xxxx 7FFF FFC5 |
| MVC | TEMP(4),TWO31 | TEMP:<br>4E00 0000 7FFF FFC5 |
| LD | 2,TEMP | FPR2:<br>4E00 0000 7FFF FFC5 |
| SD | 2,TWO31 | FPR2:<br>C23B 0000 0000 0000 |

The EXCLUSIVE OR (X) instruction inverts the leftmost bit in general register 9, using the right half of the constant as the source for a leftmost one bit. The next two instructions assemble the modified number in an unnormalized long HFP format, using the left half of the constant as the plus sign, the characteristic, and the leading zeros of the fraction. LOAD (LD) places the number unchanged in floating-point register 2. The SUBTRACT NORMALIZED (SD) instruction performs the final two steps by subtracting $2^{31}$ in HFP form and normalizing the result.

### Hexadecimal Floating Point to Fixed Point

The procedure described here consists basically in reversing the steps of the previous procedure. Two additional considerations must be taken into account. First: the HFP number may not be an exact integer. Truncating the excess hexadecimal digits on the right requires shifting the number one digit position farther to the right than desired for the final result, so that the units digit occupies the position of the guard digit. Second: the HFP number may have to be tested as to whether it is outside the range of numbers representable as a 32-bit signed binary integer.

Assume that floating-point register 6 contains the number $59.25_{10} = 3B.4_{16}$ in normalized form:

```
FPR6:     42 3B 40 00 00 00 00 00
```

Further, assume three eight-byte fields in storage: TEMP, for use as temporary storage, and the constants $2^{32}$ (TWO32) and $2^{31}$ (TWO31R) in the following formats:

```
TWO32:    4E 00 00 01 00 00 00 00
TWO31R:   4F 00 00 00 08 00 00 00
```

The constant TWO31R is shifted right one more position than the constant TWO31 of the previous example, so as to force the units digit into the guard-digit position.

The following instruction sequence performs the integer truncation, range tests, and conversion to a signed binary integer in general register 8 (GR8):



LzLabs GmbH. Ex. 1008-890

| | | Result |
|---|---|---|
| SD | 6,TWO31R | FPR6:<br>C87F FFFF C500 0000 |
| BC | 11,OVERFLOW | Branch to overflow routine if result is greater than or equal to zero |
| AW | 6,TWO32 | FPR6:<br>4E00 0000 8000 003B |
| BC | 4,OVERFLOW | Branch to overflow routine if result is less than zero |
| STD | 6,TEMP | TEMP:<br>4E00 0000 8000 003B |
| XI | TEMP+4,X'80' | TEMP:<br>4E00 0000 0000 003B |
| L | 8,TEMP+4 | GR8:<br>0000 003B |

The SUBTRACT NORMALIZED (SD) instruction shifts the fraction of the number to the right until it lines up with TWO31R, which causes the fraction digit 4 to fall to the right of the guard digit and be lost; the result of subtracting $2^{31}$ from the remaining digits is renormalized. The result should be less than zero; if not, the original number was too large in the positive direction. The first BRANCH ON CONDITION (BC) performs this test.

The ADD UNNORMALIZED (AW) instruction adds $2^{32}$: $2^{31}$ to correct for the previous subtraction and another $2^{31}$ to change to an all-positive range. The second BC tests for a result less than zero, showing that the original number was too large in the negative direction. The unnormalized result is placed in temporary storage by the STORE (STD) instruction. There the leftmost bit of the binary integer is inverted by the EXCLUSIVE OR (XI) instruction to subtract $2^{31}$ and thus convert the unsigned number to the signed format. The final result is loaded into GR8.

## Multiprogramming and Multiprocessing Examples

When two or more programs sharing common storage locations are being executed concurrently in a multiprogramming or multiprocessing environment, one program may, for example, set a flag bit in the common-storage area for testing by another program. It should be noted that the instructions AND (NI or NC), EXCLUSIVE OR (XI or XC), and OR (OI or OC) could be used to set flag bits in a multiprogramming environment; but the same instructions may cause program logic errors in a multiprocessing configuration where two or more CPUs can fetch, modify, and store data in the same storage locations simultaneously.

## Example of a Program Failure Using OR Immediate

Assume that two independent programs try to set different bits to one in a common byte in storage. The following example shows how the use of the instruction OR immediate (OI) can fail to accomplish this, if the programs are executed simultaneously on two different CPUs. One of the possible error situations is depicted.

| Execution of instruction OI FLAGS,X'01' on CPU A | FLAGS | Execution of instruction OI FLAGS,X'80' on CPU B |
|---|---|---|
| | X'00' | Fetch FLAGS X'00' |
| Fetch FLAGS X'00' | X'00' | |
| | X'00' | OR X'80' into X'00' |
| OR X'01' into X'00' | X'00' | |
| | X'80' | Store X'80' into FLAGS |
| Store X'01' into FLAGS | X'01' | |
| FLAGS should have value of X'81' following both updates. | | |

The problem shown here is that the value stored by the OI instruction executed on CPU A overlays the value that was stored by CPU B. The X'80' flag bit was erroneously turned off, and the data is now invalid.

The COMPARE AND SWAP instruction has been provided to overcome this and similar problems.


LzLabs GmbH. Ex. 1008-891

## Conditional Swapping Instructions (CS, CDS)

The COMPARE AND SWAP (CS) and COMPARE DOUBLE AND SWAP (CDS) instructions can be used in multiprogramming or multiprocessing environments to serialize access to counters, flags, control words, and other common storage areas.

The following examples of the use of the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions illustrate the applications for which the instructions are intended. It is important to note that these are examples of functions that can be performed by programs while the CPU is enabled for interruption (multiprogramming) or by programs that are being executed in a multiprocessing configuration. That is, the routine allows a program to modify the contents of a storage location while the CPU is enabled, even though the routine may be interrupted by another program on the same CPU that will update the location, and even though the possibility exists that another CPU may simultaneously update the same location.

The COMPARE AND SWAP instruction first checks the value of a storage location and then modifies it only if the value is what the program expects; normally this would be a previously fetched value. If the value in storage is not what the program expects, then the location is not modified; instead, the current value of the location is loaded into a general register, in preparation for the program to loop back and try again. During the execution of COMPARE AND SWAP, no other CPU can perform a store access or interlocked-update access at the specified location.

To ensure successful updating of a common storage field by two or more CPUs, all updates must be done by means of an interlocked-update reference. See the programming notes of COMPARE AND SWAP for an example of how COMPARE AND SWAP can be unsuccessful due to an OR IMMEDIATE instruction executed by another CPU.

### Setting a Single Bit

The following instruction sequence shows how the COMPARE AND SWAP instruction can be used to set a single bit in storage to one. Assume that the first byte of a word in storage called “WORD” contains eight flag bits.

```
      LA  6,X'80'   Put bit to be ORed into GR6
      SLL 6,24      Shift left 24 places to
                      align the byte to be ORed
                      with the location of the
                      flag bits within WORD
      L   7,WORD    Fetch current flag values
RETRY LR  8,7       Load flags into GR8
      OR  8,6       Set bit to one
      CS  7,8,WORD  Store new flags if current
                      flags unchanged, or re-
                      fetch current flag values
                      if changed
      BC  4,RETRY   If new flags are not stored,
                      try again
```

The format of the COMPARE AND SWAP instruction is:

Machine Format

| Op Code | $R_1$ | $R_3$ | $S_2$ |
|---|---|---|---|
| BA | 7 | 8 | **** |

Assembler Format

```
Op Code  R1,R3,S2
-------------------
   CS    7,8,WORD
```

The COMPARE AND SWAP instruction compares the first operand (general register 7 containing the current flag values) to the second operand in storage (WORD) while no CPU other than the one executing the COMPARE AND SWAP instruction is permitted to perform a store access or interlocked-update access at the specified storage location.

If the comparison is successful, indicating that the flag bits have not been changed since they were fetched, the modified copy in general register 8 is stored into WORD. If the flags have been changed, the compare will not be successful, and their new values are loaded into general register 7.

The conditional branch (BC) instruction tests the condition code and reexecutes the flag-modifying instructions if the COMPARE AND SWAP instruction indicated an unsuccessful comparison (condi-



LzLabs GmbH. Ex. 1008-892

tion code 1). When the COMPARE AND SWAP instruction is successful (condition code 0), the flags contain valid data, and the program exits from the loop.

The branch to RETRY will be taken only if some other program modifies the contents of WORD. This type of a loop differs from the typical "bit-spin" loop. In a bit-spin loop, the program continues to loop until the bit changes. In this example, the program continues to loop only if the value does change during each iteration. If a number of CPUs simultaneously attempt to modify a single location by using the sample instruction sequence, one CPU will fall through on the first try, another will loop once, and so on until all CPUs have succeeded.

### Updating Counters

In this example, a 32-bit counter is updated by a program using the COMPARE AND SWAP instruction to ensure that the counter will be correctly updated. The original value of the counter is obtained by loading the word containing the counter into general register 7. This value is moved into general register 8 to provide a modifiable copy, and general register 6 (containing an increment to the counter) is added to the modifiable copy to provide the updated counter value. The COMPARE AND SWAP instruction is used to ensure valid storing of the counter.

The program updating the counter checks the result by examining the condition code. The condition code 0 indicates a successful update, and the program can proceed. If the counter had been changed between the time that the program loaded its original value and the time that it executed the COMPARE AND SWAP instruction, the execution would have loaded the new counter value into general register 7 and set the condition code to 1, indicating an unsuccessful update. The program must then repeat the update sequence until the execution of the COMPARE AND SWAP instruction results in a successful update.

The following instruction sequence performs the above procedure:

```
     LA  6,1      Put increment (1) into GR6
     L   7,CNTR   Put original counter value
                    into GR7
LOOP LR  8,7      Set up copy in GR8 to modify
     AR  8,6      Increment copy
     CS  7,8,CNTR Update counter in storage
     BC  4,LOOP   If original value had changed,
                    update new value
```

The following shows two CPUs, A and B, executing this instruction sequence simultaneously: both CPUs attempt to add one to CNTR.

```
 CPU A            CPU B    Comments
GR7 GR8   CNTR   GR7 GR8

            16
16  16                     CPU A loads GR7 and
                             GR8 from CNTR
                  16  16   CPU B loads GR7 and
                             GR8 from CNTR
                      17   CPU B adds one to GR8
    17                     CPU A adds one to GR8
            17             CPU A executes CS;
                             successful match,
                             store
                  17       CPU B executes CS; no
                             match, GR7 changed
                             to CNTR value
                      18   CPU B loads GR8 from
                             GR7, adds one to GR8
            18             CPU B executes CS;
                             successful match,
                             store
```

## Bypassing Post and Wait

### Bypass Post Routine

The following routine allows the SVC "POST" as used in MVS/ESA to be bypassed whenever the corresponding WAIT has not yet been executed, provided that the supervisor WAIT and POST routines use COMPARE AND SWAP to manipulate event control blocks (ECBs).

Initial Conditions:

GR0 contains the POST code.

GR1 contains the address of the ECB.

GR5 contains 40 00 00 00$_{16}$



LzLabs GmbH. Ex. 1008-893

```
HSPOST  OR    0,5         Set bit 1 of GR0 to
                            one
        L     3,0(1)      GR3 = contents of ECB
        LTR   3,3         ECB marked 'waiting'?
        BC    4,PSVC      Yes, execute post
                            SVC
        CS    3,0,0(1)    No, store post code
        BC    8,EXITHP    Continue
PSVC    POST  (1),(0)     ECB address is in GR1,
                            post code in GR0
EXITHP  [Any instruction]
```

The following routine may be used in place of the previous HSPOST routine if it is assumed that bit 1 of the contents of GR0 is already set to one and if the ECB is assumed to contain zeros when it is not marked "WAITING."

```
HSPOST  SR   3,3
        CS   3,0,0(1)
        BC   8,EXITHP
        POST (1),(0)
EXITHP  [Any instruction]
```

### Bypass Wait Routine

A BYPASS WAIT function, corresponding to the BYPASS POST, does not use the CS instruction, but the FIFO LOCK/UNLOCK routines which follow assume its use.

```
HSWAIT  TM   0(1),X'40'
        BC   1,EXITHW  If bit 1 is one, then
                       ECB is already posted;
                       branch to exit
        WAIT ECB=(1)
EXITHW  [Any instruction]
```

## Lock/Unlock

When a common storage area larger than a doubleword is to be updated, it is usually necessary to provide special interlocks to ensure that a single program at a time updates the common area. Such an area is called a serially reusable resource (SRR).

In general, updating a list, or even scanning a list, cannot be safely accomplished without first "freezing" the list. However, the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions can be used in certain restricted situations to perform queuing and list manipulation. Of prime importance is the capability to perform the lock/unlock functions and to provide sufficient queuing to resolve contentions, either in a LIFO or FIFO manner. The lock/unlock functions can then be used as the interlock mechanism for updating an SRR of any complexity.

The lock/unlock functions are based on the use of a "header" associated with the SRR. The header is the common starting point for determining the states of the SRR, either free or in use, and also is used for queuing requests when contentions occur. Contentions are resolved using WAIT and POST. The general programming technique requires that the program that encounters a "locked" SRR must "leave a mark on the wall" indicating the address of an ECB on which it will WAIT. The "unlocking" program sees the mark and posts the ECB, thus permitting the waiting program to continue. In the two examples given, all programs using a particular SRR must use either the LIFO queuing scheme or the FIFO scheme; the two cannot be mixed. When more complex queuing is required, it is suggested that the queue for the SRR be locked using one of the two methods shown.

### Lock/Unlock with LIFO Queuing for Contentions

The header consists of a word, that is, a four-byte field aligned on a word boundary. The word can contain zero, a positive value, or a negative value.

- A zero value indicates that the serially reusable resource (SRR) is free.
- A negative value indicates that the SRR is in use but no additional programs are waiting for the SRR.
- A positive value indicates that the SRR is in use and that one or more additional programs are waiting for the SRR. Each waiting program is identified by an element in a chained list. The positive value in the header is the address of the element most recently added to the list.

Each element consists of two words. The first word is used as an ECB; the second word is used as a pointer to the next element in the list. A negative value in a pointer indicates that the element is the last element in the list. The element is required only if the program finds the SRR locked and desires to be placed in the list.

The following chart describes the action taken for LIFO LOCK and LIFO UNLOCK routines. The routines following the chart allow enabled code to perform the actions described in the chart.



LzLabs GmbH. Ex. 1008-894

| Function | Action | | |
|---|---|---|---|
| | Header Contains Zero | Header Contains Positive Value | Header Contains Negative Value |
| LIFO LOCK (the incoming element is at location A) | SRR is free. Set the header to a negative value. Use the SRR. | SRR is in use. Store the contents of the header into location A+4. Store address A into the header. WAIT; the ECB is at location A. | |
| LIFO UNLOCK | Error | Some program is waiting for the SRR. Move the pointer from the "last in" element into the header. POST; the ECB is in the "last in" element. | The list is empty. Store zeros into the header. The SRR is free. |

***LIFO LOCK Routine:***

Initial Conditions:

GR1 contains the address of the incoming element.

GR2 contains the address of the header.

```
LLOCK  SR     3,3       GR3 = 0
       ST     3,0(1)    Initialize the ECB
       LNR    0,1       GR0 = a negative value
TRYAGN CS     3,0,0(2)  Set the header to a nega-
                          tive value if the header
                          contains zeros
       BC     8,USE     Did the header contain
                          zeros?
       ST     3,4(1)    No, store the value of the
                          header into the pointer
                          in the incoming element
       CS     3,1,0(2)  Store the address of the
                          incoming element into
                          the header
       LA     3,0(0)    GR3 = 0
       BC     7,TRYAGN  Did the header get up-
                          dated?
       WAIT   ECB=(1)   Yes, wait for the re-
                          source; the ECB is in
                          the incoming element
USE    [Any instruction]
```

***LIFO UNLOCK Routine:***

Initial Conditions:

GR2 contains the address of the header.

```
LUNLK L     1,0(2)    GR1 = the contents of the
                        header
A     LTR   1,1       Does the header contain a
      BC    4,B         negative value?
      L     0,4(1)    No, load the pointer from
      CS    1,0,0(2)    the "last in" element and
                        store it in the header
      BC    7,A       Did the header get updated?
      POST  (1)       Yes, post the "last in"
                        element
      BC    15,EXIT   Continue
B     SR    0,0       The header contains a neg-
      CS    1,0,0(2)    ative value; free the
      BC    7,A         header and continue
EXIT  [Any instruction]
```

Note that the LOAD instruction L 1,0(2) at location LUNLK would have to be CS 1,1,0(2) if it were not for the rule concerning storage-operand consistency. This rule requires the LOAD instruction to fetch a four-byte operand aligned on a word boundary such that, if another CPU changes the word being fetched by an operation which is also at least word-consistent, either the entire new or the entire old value of the word is obtained, and not a combination of the two. (See “Storage-Operand Consistency” on page 5-87.)

## Lock/Unlock with FIFO Queuing for Contentions

The header always contains the address of the most recently entered element. The header is originally initialized to contain the address of a posted ECB. Each program using the serially reusable resource (SRR) must provide an element regardless of whether contention occurs. Each program then enters the address of the element which it has provided into the header, while simultaneously it removes the address previously contained in the header. Thus, associated with any particular program attempting to use the SRR are two elements, called the “entered element” and the “removed element.” The “entered element” of one program becomes the “removed element” for the immediately following program. Each program then waits on the removed element, uses the SRR, and then posts the entered element.

When no contention occurs, that is, when the second program does not attempt to use the SRR until after the first program is finished, then the POST of the first program occurs before the WAIT of the second program. In this case, the bypass-post and bypass-wait routines described in the preceding section are applicable. For simplicity,



LzLabs GmbH. Ex. 1008-895

these two routines are shown only by name rather than as individual instructions.

In the example, the element need be only a single word, that is, an ECB. However, in actual practice, the element could be made larger to include a pointer to the previous element, along with a program identification. Such information would be useful in an error situation to permit starting with the header and chaining through the list of elements to find the program currently holding the SRR.

It should be noted that the element provided by the program remains pointed to by the header until the next program attempts to lock. Thus, in general, the entered element cannot be reused by the program. However, the removed element is available, so each program gives up one element and gains a new one. It is expected that the element removed by a particular program during one use of the SRR would then be used by that program as the entry element for the next request to the SRR.

It should be noted that, since the elements are exchanged from one program to the next, the elements cannot be allocated from storage that would be freed and reused when the program ends. It is expected that a program would obtain its first element and release its last element by means of the routines described in "Free-Pool Manipulation."

The following chart describes the action taken for FIFO LOCK and FIFO UNLOCK.

| Function | Action |
|---|---|
| FIFO LOCK<br><br>(the incoming element is at location A) | Store address A into the header.<br>WAIT; the ECB is at the location addressed by the old contents of the header. |
| FIFO UNLOCK | POST; the ECB is at location A. |

The following routines allow enabled code to perform the actions described in the previous chart.

***FIFO Lock Routine:***

Initial conditions:

GR3 contains the address of the header.

GR4 contains the address, A, of the element currently owned by this program. This element becomes the entered element.

```
FLOCK  LR    2,4        GR2 now contains address
                          of element to be
                          entered
       SR    1,1        GR1 = 0
       ST    1,0(2)     Initialize the ECB
       L     1,0(3)     GR1 = contents of the
                          header
TRYAGN CS    1,2,0(3)   Enter address A into
                          header while remember-
       BC    7,TRYAGN     ing old contents of
                          header into GR1; GR1
                          now contains address
                          of removed element
       LR    4,1        Removed element becomes
                          new currently owned
                          element
       HSWAIT           Perform bypass-wait
                          routine; if ECB al-
                          ready posted, con-
                          tinue; if not, wait;
                          GR1 contains the ad-
                          dress of the ECB
USE    [Any instruction]
```

***FIFO Unlock Routine:***

Initial conditions:

GR2 contains the address of the removed element, obtained during the FLOCK routine.

GR5 contains $40\ 00\ 00\ 00_{16}$

```
FUNLK     LR   1,2  Place address of entered
                      element in GR1; GR1 = ad-
                      dress of ECB to be posted
          SR   0,0  GR0 = 0; GR0 has a post code
                      of zero
          OR   0,5  Set bit 1 of GR0 to one
          HSPOST    Perform bypass-post routine;
                      if ECB has not been waited
                      on, then mark posted and
                      continue; if it has been
                      waited on, then post
CONTINUE [Any instruction]
```

## Free-Pool Manipulation

It is anticipated that a program will need to add and delete items from a free list without using the lock/unlock routines. This is especially likely since the lock/unlock routines require storage elements for queuing and may require working storage. The lock/unlock routines discussed previously allow simultaneous lock routines but permit only one unlock routine at a time. In such a situation, multiple additions and a single deletion to the list



LzLabs GmbH. Ex. 1008-896

may all occur simultaneously, but multiple deletions cannot occur at the same time. In the case of a chain of pointers containing free storage buffers, multiple deletions along with additions can occur simultaneously. In this case, the removal cannot be done using the COMPARE AND SWAP instruction without a certain degree of exposure.

Consider a chained list of the type used in the LIFO lock/unlock example. Assume that the first two elements are at locations A and B, respectively. If one program attempted to remove the first element and was interrupted between the fourth and fifth instructions of the LUNLK routine, the list could be changed so that elements A and C are the first two elements when the interrupted program resumes execution. The COMPARE AND SWAP instruction would then succeed in storing the value B into the header, thereby destroying the list.

The probability of the occurrence of such list destruction can be reduced to *near* zero by appending to the header a counter that indicates the number of times elements have been added to the list. The use of a 32-bit counter guarantees that the list will not be destroyed unless the following events occur, in the exact sequence:

1. An unlock routine is interrupted between the fetch of the pointer from the first element and the update of the header.
2. The list is manipulated, including the deletion of the element referenced in 1, and exactly $2^{32}$ (or an integer multiple of $2^{32}$) additions to the list are performed. Note that this takes on the order of days to perform in any practical situation.
3. The element referenced in 1 is added to the list.
4. The unlock routine interrupted in 1 resumes execution.

The following routines use such a counter in order to allow multiple, simultaneous additions and removals at the head of a chain of pointers.

The list consists of a doubleword header and a chain of elements. The first word of the header contains a pointer to the first element in the list. The second word of the header contains a 32-bit counter indicating the number of additions that have been made to the list. Each element contains a pointer to the next element in the list. A zero value indicates the end of the list.

The following chart describes the free-pool-list manipulation.

| | Action | |
|---|---|---|
| Function | Header = 0,Count | Header = A,Count |
| ADD TO LIST (the incoming element is at location A) | Store the first word of the header into location A. Store the address A into the first word of the header. Decrement the second word of the header by one. | |
| DELETE FROM LIST | The list is empty. | Set the first word of the header to the value of the contents of location A. Use element A. |

The following routines allow enabled code to perform the free-pool-list manipulation described in the above chart.

***ADD TO FREE LIST Routine:***

Initial Conditions:

GR2 contains the address of the element to be added.

GR4 contains the address of the header.

```
ADDQ   LM   0,1,0(4) GR0,GR1 = contents of the
                       header
TRYAGN ST   0,0(2)   Point the new element to
                       the top of the list
       LR   3,1      Move the count to GR3
       BCTR 3,0      Decrement the count
       CDS  0,2,0(4) Update the header
       BC   7,TRYAGN
```

***DELETE FROM FREE LIST Routine:***

Initial conditions:

GR4 contains the address of the header.

```
DELETQ LM    2,3,0(4)    GR2,GR3 = contents of
                           the header
TRYAGN LTR   2,2         Is the list empty?
       BC    8,EMPTY     Yes, get help
       L     0,0(2)      No, GR0 = the pointer
                           from the first ele-
                           ment
       LR    1,3         Move the count to GR1
       CDS   2,0,0(4)    Update the header
       BC    7,TRYAGN
USE    [Any instruction] The address of the re-
                           moved element is in
                           GR2
```

Note that the LM (LOAD MULTIPLE) instructions at locations ADDQ and DELETQ would have to be CDS (COMPARE DOUBLE AND SWAP)


LzLabs GmbH. Ex. 1008-897

instructions if it were not for the rule concerning storage-operand consistency. This rule requires the LOAD MULTIPLE instructions to fetch an eight-byte operand aligned on a doubleword boundary such that, if another CPU changes the doubleword being fetched by an operation which is also at least doubleword-consistent, either the entire new or the entire old value of the doubleword is obtained, and not a combination of the two. (See "Storage-Operand Consistency" on page 5-87.)

## PERFORM LOCKED OPERATION (PLO)

The PERFORM LOCKED OPERATION instruction can be used in a multiprogramming or multiprocessing environment to perform compare, load, compare-and-swap, and store operations on two or more discontiguous locations that can be words or doublewords. The operations are performed as an atomic set of operations under the control of a lock that is held only for the duration of the execution of a single PERFORM LOCKED OPERATION instruction, as opposed to across the execution of multiple instructions. Since lock contention is resolved by the CPU and is very brief, the program need not include a method for dealing with the case when the lock to be used is held by a program being executed by another CPU. Also, there need be no concern that the program may be interrupted while it holds a lock, since PERFORM LOCKED OPERATION will complete its operation and release its lock before an interruption can occur.

PERFORM LOCKED OPERATION can be thought of as performing concurrent interlocked updates of multiple operands. However, the instruction does not actually perform any interlocked update, and a serially reusable resource cannot be updated predictably through the use of both PERFORM LOCKED OPERATION and conditional-swapping instructions (CS and CDS).

Following is an example of how PERFORM LOCKED OPERATION can be used to add an element at the beginning of a queue.

Assume the following variables associated with the queue: S, which is a sequence number that is incremented anytime the queue is changed; H (for head), which is the address of the first element on the queue; and C, which is a count of the number of elements on the queue. Assume a queue element contains a variable, F (for forward), which is the address of the next element on the queue. If a new element, N, is to be enqueued at the head of the queue, that can be done by setting F in N to H and then performing the following atomic set of operations:

```
S+1  ⟶ S
A(N) ⟶ H
C+1  ⟶ C
```

where A(N) is the address of N.

The enqueueing of N can be done by means of the following steps:

1. Obtain consistent values of S, H, and C, meaning obtain S and obtain the H and C that are consistent with that value of S.
2. Store H in N.F.
3. By means of PLO.csdst (PERFORM LOCKED OPERATION performing compare and swap and double store), with S as the swap variable and H and C as the store variables, add one to S, set H to A(N), and add one to C, provided that S still has the value obtained in step 1. If S has already been changed, go back to step 1.

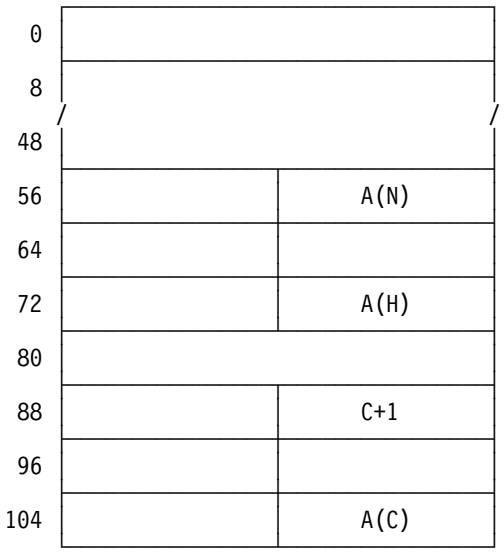
Consistent values of S, H, and C cannot necessarily be obtained simply by using three LOAD instructions because a PERFORM LOCKED OPERATION instruction being executed by another CPU may have completed an update of S but not yet of H or C. In this case, the three LOAD instructions will obtain the new S but the old H or C. However, as will be described, it may be possible to use three LOAD instructions.

If S is obtained while holding the lock, meaning by means of PERFORM LOCKED OPERATION, then H and C can be obtained by LOAD instructions since no other CPU can subsequently change H or C without changing S, as observed when the lock is held.

The parameter list used by the PLO.csdst is as follows, assuming the access-register mode is not used:



LzLabs GmbH. Ex. 1008-898




The program is as follows:

```
      LA    RT,H      Initialize addresses in PL
                        (T = temp)
      ST    RT,PL+76  Op4 address (address of H)
      LA    RT,C
      ST    RT,PL+108 Op6 address (address of C)
      LA    RN,N      Address of N
      ST    RN,PL+60  Initialize op3 in PL
                        (address of N)
      LA    R1,S      PLT address = address of S
-----------------------------------------------
      SR    RS,RS     Dummy S.  CC1 will
                        probably be set
      SR    R0,R0     Function code 0 (compare
                        and load)
      PLO   RS,S,RS,S Obtain S while holding
                        lock
-----------------------------------------------
      LA    R0,16     Function code 16 (csdst)
LOOP  L     RT,H      Consistent H
      ST    RT,OFSTF(,RN)  OFSTF = offset of F
                        in N
      L     RT,C      Consistent C
      LA    RT,1(,RT) C+1
      ST    RT,PL+92  Initialize op5 in PL (C+1)
      LA    RSP,1(,RS)  RS/RSP = even/odd pair.
                        S+1 in RSP
      PLO   RS,S,0,PL
      BNZ   LOOP      Br if S changed (if CC not
                        0)
```

Note the following about the first PERFORM LOCKED OPERATION instruction (PLO.cl). If S is not zero (which is probably true), S (the second operand, op2) is loaded into RS (the first-operand comparison value, op1c). If S is zero, S (the fourth operand, op4) is loaded into RS (the third operand, op3). Either of these loads occurs while the lock is held. It is unnecessary to test the condition code to determine which load occurred.

The above program may be a simplification. If the queue has associated with it a variable, T (for tail), that is the address of the last element on the queue, and the queue is currently empty, T also must be set when N is added to the queue. This would require a different program using a compare-and-swap-and-triple-store operation.

If the queue is added to, deleted from, and rearranged by means of PERFORM LOCKED OPERATION instructions in which the sequence number, S, is always the second operand, then, since the definition of PERFORM LOCKED OPERATION specifies that the second operand is always stored last, the first PERFORM LOCKED OPERATION instruction in the above program can be replaced by a LOAD instruction. The three instructions within the dashed lines would be replaced by L RS,S.

## Sorting Instructions

## Tree Format

Two instructions, COMPARE AND FORM CODEWORD and UPDATE TREE, refer to a tree — a data structure with a specific format. A tree consists of some number (always odd) of consecutively numbered nodes. Node 1 is the root of the tree. Every node except the root has one parent node in the same tree. Every parent node has two son nodes. Every even-numbered node is the *leftson* of its parent node, and every odd-numbered node (except node 1) is the *rightson* of its parent node. Division by two (ignoring remainder) of the node number gives the parent node number. Nodes with sons are also called internal nodes, and nodes without sons are called terminal nodes. Figure A-5 on page A-53 illustrates schematically a 21-node tree with arrows drawn from each parent node to each son node.

A tree is used for merging several sorted sequences of records into a single merged sequence of records. At each step in the merging process, there exists the initial part of the merged sequence and the remaining parts of each of the sorted sequences that are being merged. Each


LzLabs GmbH. Ex. 1008-899

step consists in selecting the lowest record (the record with the lowest key when sorting in ascending sequence) from all of the as yet unmerged parts of the sorted sequences and adding it to the merged sequence. Each terminal node in the tree represents one of the sorted sequences. The number of internal nodes in the tree is one less than the number of sorted sequences. Each internal node conceptually contains one record from each of the sorted sequences but one; these are the lowest records, from all but one of the sorted sequences, that have not yet been added to the merged sequence. In addition, there is the lowest record from the one remaining sorted sequence. This additional record is compared and interchanged with nodes of the tree to select the record to be added next to the merged sequence. This processing begins with the parent of the terminal node that represents the one remaining sorted sequence, and it continues from that node along the path to the root of the tree. The selected record emerges from the root of the tree.

The tree may perhaps be most easily explained by considering each node to represent a comparison operation in an "elimination tournament" to find the lowest record. After the tournament has been completed, each node has an associated "loser" record which had a higher key in the comparison represented by that node. Besides a loser record at each node, there is one record (the "winner") which is not associated with any node since it never compared high. The next step would be to introduce a new record from the same sorted sequence from which the winner record originated and replay the tournament with the new record in place of the former winner. It can be seen that it is unnecessary to do all the comparisons represented by all the nodes in the tree — most of them are unaffected by the new record replacing the former winner. In fact, it is sufficient to redo only those node comparisons in which the former winner record participated. Each new record is inserted into the tree at the terminal node that represents the sorted sequence containing the record. The use of the tree assumes that programming provides a method of remembering at which terminal node each winning record originated. The instruction UPDATE TREE allows for a new record to be inserted at a terminal node and the tree to be updated so that a new winner record is left in the general registers.

Rather than comparing the actual keys of records, much of the merge logic can be performed using "codewords" to represent a record key rather than referring to actual keys. The value of a codeword at a node in the tree depends not only on the record's key but also on the key of the winning record in the last comparison at that node. The codeword consists of two parts:

1. Bits 16-31 contain the one's complement of the first halfword in which the record key differs from that of the node's winning record.
2. Bits 0-15 specify the byte offset of the halfword in this record's key just beyond the halfword value (complemented) in bit positions 16-31.

When comparing records in the path of the last winner record, if the new record is also represented by a codeword resulting from a comparison with the last winner, all codewords in the update path are with respect to the same winner. When comparing such codewords, a high codeword represents a low key and vice versa. Thus, when codewords are unequal, a node entry with a high codeword (representing a low actual key) should move up the tree.

In the case of a tie value of codewords, it is necessary to refer to the actual keys. This is done by the instruction COMPARE AND FORM CODEWORD, which resolves the ambiguity and computes a new codeword for the high-key (loser) record.

The eight bytes at each node of a tree consist of (1) a codeword for this record, computed with respect to the last record which compared low against this record and (2) a parameter usable to locate this record, for example, a direct or indirect address.

The instruction UPDATE TREE is so defined that tree updating stops after equal codewords are detected and the tie-breaking instruction COMPARE AND FORM CODEWORD can be used, after which UPDATE TREE can resume tree updating at the point where equal codewords were previously found.

COMPARE AND FORM CODEWORD may alternatively be used for merging in descending sequence. In that case, bits 16-31 of the codeword at a node contain the true value of the first halfword in which the record key differs from



LzLabs GmbH. Ex. 1008-900

that of the node's winning record. When the descending option of COMPARE AND FORM CODEWORD is used, the higher of two codewords represents the higher key.

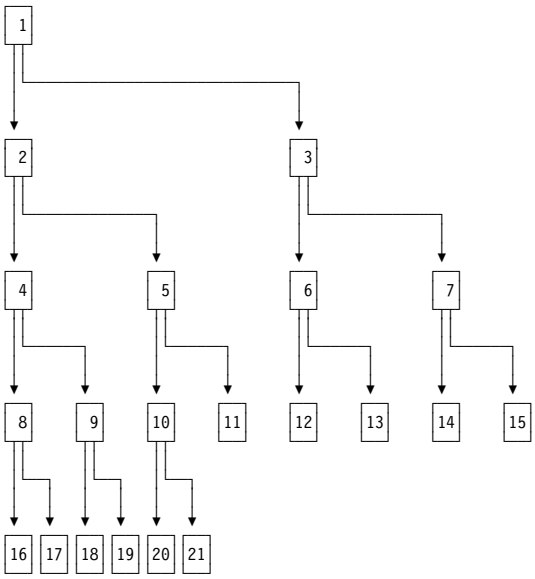



*Figure A-5. Schematic Diagram of Merge Control Tree with 21 Nodes*

## Example of Use of Sort Instructions

An example illustrates how the instructions UPDATE TREE and COMPARE AND FORM CODEWORD may be used in the merge operation within a sort program. A five-way merge requires a tree data structure with four internal nodes and five terminal-node positions. The schematic diagram shown later in this section illustrates such a tree, containing four internal nodes (not counting the dummy node) and five input sequences for a merge, one sequence at each terminal-node position. Each record in an input sequence in the diagram is indicated by its address. The actual record contents are shown in Figure A-7 on page A-57. Each record contains 16 bytes, consisting of the following fields:

| Byte Offset (hexadecimal) | Field |
|---|---|
| 0-5 | Six-byte record key. |
| 6-7 | Halfword node index specifying the input sequence of the next record of this input sequence. |
| 8-B | Address of the next record in the same input sequence. |
| C-F | This chaining field is initially zero. At the completion of the merge, this field is to contain the address of the next record in the merged sequence. |

The merge process forms a single sorted sequence from five input sequences, each of which is in sorted order. This process can be subdivided into three steps:

1. A priming step takes the first record from each of the five input sequences and places them in the tree data structure. For each record to be introduced into the tree, first its codeword value is computed with respect to the lowest possible key value of all zeros. This codeword, with a second word which contains the address of the actual record, forms a doubleword node value that can be placed at the appropriate node. After priming, the node values, one each from each of the five input sequences, will have been placed in the tree so that each of the four internal nodes contains one node value and the node value for a winner record has emerged from the root of the tree.
2. After each winner emerges from the tree, the main merge process is performed repeatedly. Each iteration introduces the node value for one new record into the tree and produces a node value for a new winner record. The tree plus the winner must at all times contain precisely one node value from each input sequence being merged. Therefore, the new node value that is introduced into the tree on each iteration must come from the same input sequence from which the winner node value in the preceding iteration originated.
3. When the node value for the last record of an input sequence emerges as a winner, there is no successor record from that input sequence to be introduced into the tree on the next iteration. Hence, the order of the merge must be reduced by one for each such occurrence.


LzLabs GmbH. Ex. 1008-901

This runout process will consist of one or more iterations for each of a four-way, three-way, two-way, and one-way merge. The onset of runout occurs in the example when it is found that the next input record from a sequence is lower than its predecessor (a sequence break).

The priming process is discussed next, and the state of the tree is shown after priming is complete. Then, a short program that uses the instructions UPDATE TREE and COMPARE AND FORM CODEWORD to perform the main merge is described. An abbreviated trace is then presented to show the status of the tree and certain general registers for 16 iterations of the main merge. The runout process is not discussed in this example.

Priming begins by forming the node value for the first record of each input sequence. The first word of the node value is the codeword formed by executing COMPARE AND FORM CODEWORD on a record key containing all binary zeros. The second word of the node value is the address of the record represented by that node value. The node values for the first record of each input sequence are:

| Sequence Index | Node Values |
|---|---|
| 28 | 0006 FFFC 0000 1030 |
| 30 | 0006 FFFB 0000 1040 |
| 38 | 0006 FFFA 0000 1050 |
| 40 | 0004 FFFE 0000 1080 |
| 48 | 0006 FFF0 0000 1060 |

In the example, the tree data structure is assumed to have base address X'1000', which is kept in general register 4 (to match the expected use in UPDATE TREE). Similarly, internal-node index values and input-sequence index values are always used from general register 5.

Although the tree-priming program is not part of this example, the UPDATE TREE instruction is used in creating it as follows. First, the codeword position for each internal node of the tree is initialized to all ones (X'FFFF FFFF'). This artifice fills the tree with dummy low records. Then, for each record in the table, (1) the sequence index is loaded into general register 5, (2) the node value is loaded into general registers 0 and 1, and (3) UPDATE TREE is executed. At the completion of this priming process, the tree-node contents in the example are as shown on line 0 of Figure A-9 on page A-59. The contents of the general registers are as shown on the first line of Figure A-8 on page A-58.

The figure illustrating the program for the main merge is divided into three groups of columns, containing the absolute program, the general-register trace, and the symbolic program. The first part of the program extends from symbolic locations L1 through L2; it introduces a new record into the tree and executes an UPDATE TREE instruction. If no tied codewords are encountered in UPDATE TREE, then the BRANCH ON CONDITION instruction following UPDATE TREE loops back to L1 to introduce the next record into the tree. This BRANCH ON CONDITION instruction is suitable for use when UPDATE TREE operates in accordance with either its method 1 (setting condition code 1) or its method 2 (setting condition code 3). (The preceding sentence applies to 370-XA. In ESA/370 and ESA/390, UPDATE TREE operates in accordance with only method 2, which is not to say that it cannot set condition code 1. Method 2, but not method 1, tests for the condition that sets condition code 3.)

If UPDATE TREE encounters tied codewords, then the UPDATE TREE instruction is completed, the subsequent BRANCH ON CONDITION instruction does not branch, and control falls through to the second part of the program, which handles entries with tied codewords. This part then branches back to UPDATE TREE at L2, which resumes the tree updating. It is possible for tied codewords to be encountered at any level in the tree (or indeed at all levels), so that the tied-codeword part of the program may be entered up to three times for each record introduced.

The general-register trace for the first part of the main merge shows the contents of the first seven general registers after each instruction is executed during the first iteration. Note that the merged-chain field (at 1140) serves as the anchor for the merged-chain address chain through the records. The trace shows only the lower half of certain general registers, whose upper half is always zero.

Figure A-9 on page A-59 gives an abbreviated trace of the entire main merge of 16 records. For each record introduced into the tree, there are one or more lines (always an odd number) given in the figure to show the tree updating, which results



LzLabs GmbH. Ex. 1008-902

finally in a winner in GR0 and GR1. The first line for each record shows the values of GR5, GR2, and GR3 before the first or only execution of UPDATE TREE. For the even-numbered lines, the storage updating by UPDATE TREE of tree nodes is shown (read left to right to follow the order of swapping). For example, consider line 10 and the corresponding UPDATE TREE: since GR5 contains 28, the first storage node examined is 1010 (refer to the schematic diagram). Since the codeword in GR0 is 0004 FFFE (same as for GR2), which is less than that of the word at 1010 (0006 FFF0), the doubleword at 1010 is swapped with that in GR0 and GR1. A second comparison at 1008 in the same execution of UPDATE TREE causes another register-storage doubleword swap, which leaves the winner (record 1040) in GR1 at the completion of UPDATE TREE (see the column at the far right of Figure A-9 on page A-59).

When a codeword comparison is made which does not result in a tie or a swap (that is, when the storage-codeword value is low), an asterisk appears in the trace for that storage entry.

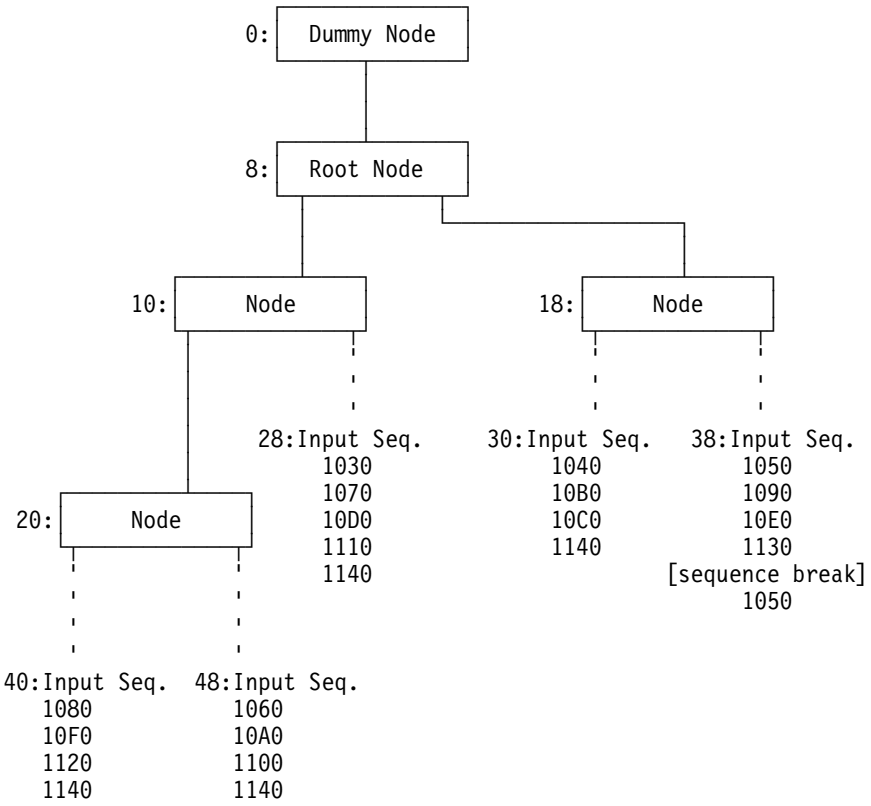
When equal codewords are found, the execution of UPDATE TREE is completed. The following line in each such case shows the result of the tied-codeword routine, which always stores a new codeword and may also store a new record address before branching back to L2 to execute UPDATE TREE again. In this line, the notation "loses" or "wins" means that the node loses or wins, respectively.

The tie-break trace part of Figure A-8 on page A-58 shows the treatment of the third record (that is, the first record for which UPDATE TREE encounters a tied codeword). This corresponds to line 31 in Figure A-9 on page A-59.

The following is a summary of the steps that are needed to use this example for verification purposes:

1. Initialize storage as follows:
   a. 1008 through 102F from line 0 of Figure A-9 on page A-59
   b. 1030 through 114F from Figure A-7 on page A-57
   c. 1150 through 1189 from Figure A-8 on page A-58
2. Initialize GRs per first line in Figure A-8 and trace first record per Figure A-8.
3. Trace to completion of each UPT or BC 15,L2 (once for each line of Figure A-9). A detailed trace of the GRs for the tied-codeword part of line 31 of Figure A-9 is given in the lower part of Figure A-8.
4. Verify that addresses in the chain beginning at 103C and continuing through 114C are as shown in the right-hand column of Figure A-7.


LzLabs GmbH. Ex. 1008-903




**Note**: Each node and input sequence is identified by a number which is the hexadecimal node index. Each input sequence is given as a list of record addresses (also in hexadecimal).

*Figure A-6. Schematic Diagram for Example of Merge to Be Performed*



LzLabs GmbH. Ex. 1008-904

| | Record Key at Hex Byte Offset | | | | | | Successor Record | | | | | | Merged-Chain Address | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | Index | | Location | | | | | | | |
| Location | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 1030 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 2 | 8 | 0 | 0 | 0 | 0 | 1 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 4 | 0 |
| 1040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | B | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 5 | 0 |
| 1050 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 3 | 8 | 0 | 0 | 0 | 0 | 1 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 6 | 0 |
| 1060 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | F | 0 | 0 | 4 | 8 | 0 | 0 | 0 | 0 | 1 | 0 | A | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 8 | 0 |
| 1070 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | F | F | F | F | 0 | 0 | 2 | 8 | 0 | 0 | 0 | 0 | 1 | 0 | D | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 9 | 0 |
| 1080 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | F | F | F | F | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | F | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 7 | 0 |
| 1090 | 0 | 0 | 0 | 0 | F | F | F | F | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 8 | 0 | 0 | 0 | 0 | 1 | 0 | E | 0 | 0 | 0 | 0 | 0 | 1 | 0 | A | 0 |
| 10A0 | 0 | 0 | 0 | 0 | F | F | F | F | 0 | 0 | 0 | 1 | 0 | 0 | 4 | 8 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | B | 0 |
| 10B0 | 0 | 0 | 0 | 0 | F | F | F | F | 0 | 0 | 0 | 2 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | C | 0 | 0 | 0 | 0 | 0 | 1 | 0 | C | 0 |
| 10C0 | 0 | 0 | 0 | 0 | F | F | F | F | 0 | 0 | 0 | 2 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | D | 0 |
| 10D0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 8 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | E | 0 |
| 10E0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 8 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | F | 0 |
| 10F0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 4 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1100 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 5 | 0 | 0 | 0 | 4 | 8 | 0 | 0 | 0 | 0 | 1 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1110 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 8 | 0 | 0 | 0 | 0 | 1 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0 |
| 1120 | 0 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 0 |
| 1130 | F | F | F | F | F | F | F | F | F | F | F | E | 0 | 0 | 3 | 8 | 0 | 0 | 0 | 0 | 1 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1140 | F | F | F | F | F | F | F | F | F | F | F | F | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 0 |

*Figure A-7. Contents of Records to Be Merged*


LzLabs GmbH. Ex. 1008-905

| Absolute | | General-Register Trace | | | | | | | Symbolic Program | |
|---|---|---|---|---|---|---|---|---|---|---|
| Loc | INSTR | GR0 | GR1 | GR2 | GR3 | GR4 | GR5 | GR6 | Loc | Instruction |
| | | 0006FFFC | 1030 | | 0000 | 1000 | 0000 | 1140 | | Using X'1000',4 |
| 1150 | 5010600C | | | | | | | | L1 | ST 1,12(,6) Store merged-chain address |
| 1154 | 48501006 | | | | | | 0028 | | | LH 5,6(,1) Load node index of input sequence of winner |
| 1158 | 58301008 | | | | 1070 | | | | | L 3,8(,1) Load successor-record address |
| 115C | 1861 | | | | | | | 1030 | | LR 6,1 Save old winner address for next merged-chain store |
| 1153 | 1B22 | | | 00000000 | | | | | | SR 2,2 Zero GR2 as initial offset |
| 1160 | B21A0004 | | | 0004FFFE | | | | | | CFC 4 Compute codeword of new record based on last winner |
| 1164 | 4720418A | | | | | | | | | BC 2,L3 Exit on CC=2 (sequence break |
| 1168 | 1813 | | 1070 | | | | | | | LR 1,3 \| |
| | | | | | | | | | | > Move new record entry |
| 116A | 1802 | 0004FFFE | | | | | | | | LR 0,2 \| to GRs 0-1 |
| 116C | 0102 | 0006FFFB | 1040 | | | | 0000 | | L2 | UPT Update tree data structure |
| 116E | 47504150 | | | | | | | | | BC 5,L1 If no codeword tie found, branch to next iteration |
| | | | | | | | | | * | Fall through on tied codewords |
| | | 00040000 | 1090 | 00040000 | 10B0 | | 0018 | 1050 | * | ←GR values for tie-break trace |
| 1172 | 88200010 | | | 00000004 | | | | | | SRL 2,16 Shift codeword offset to initial offset position for CFC |
| 1176 | B21A0004 | | | 0006FFFD | | | | | | CFC 4 Compute loser codeword |
| 117A | 50254000 | | | [CC=1] | | | | | | ST 2,0(5,4) Store loser codeword in current storage node |
| 117E | 47C0416C | | | branch taken | | | | | | BC 12,L2 Resume tree update if old storage-node entry is loser |
| 1182 | 50354004 | | | | | | | | | ST 3,4(5,4) Store loser record address |
| 1186 | 47F0416C | | | | | | | | | BC 15,L2 Resume tree update |
| 118A | ... | | | | | | | | L3 | ... Control reaches here at end |

*Figure A-8. Program for Main Merge*



LzLabs GmbH. Ex. 1008-906

| | General Regs after CFC at Location 1160 | | | | Storage Trace of Node Entries | | | | | | | | General Regs after UPT or BC 15,L2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L# | GR 5 | GR2 | GR3 | Comment | 1020 | | 1018 | | 1010 | | 1008 | | GR0 | GR1 |
| 0[1] | | | | | 0004FFFE | 1080 | 0006FFFA | 1050 | 0006FFF0 | 1060 | 0006FFFB | 1040 | 0006FFFC | 1030 |
| 10 | 28 | 0004FFFE | 1070 | No tie | | | | | 0004FFFE | 1070 | 0006FFF0 | 1060 | 0006FFFB | 1040 |
| 20 | 30 | 00040000 | 10B0 | No tie | | | 00040000 | 10B0 | | | * | | 0006FFFA | 1050 |
| 30<br>31<br>32 | 38 | 00040000 | 1090 | CC = 0<br>Loses<br>No tie | | | Tie<br>0006FFFD | | | | <br><br>00040000 | <br><br>1090 | 00040000<br>00040000<br>0006FFF0 | 1090<br>1090<br>1060 |
| 40<br>41<br>42 | 48 | 00040000 | 10A0 | CC = 0<br>Equal<br>No tie | 00040000 | 10A0 | | | Tie<br>80001070 | | <br><br>* | <br><br> | 0004FFFE<br>0004FFFE<br>0004FFFE | 1080<br>1080<br>1080 |
| 50 | 40 | 0002FF7F | 10F0 | No tie | 0002FF7F | 10F0 | | | 00040000 | 10A0 | ** | | 80001070 | 1070 |
| 60<br>61<br>62 | 28 | 0002FFFE | 10D0 | CC = 0<br>Wins<br>No comp | | | | | 0002FFFE | 10D0 | Tie<br>0006FFFE | <br>10A0 | 00040000<br>00040000<br>00040000 | 10A0<br>1090<br>1090 |
| 70 | 38 | 0002FF7F | 10E0 | No tie | | | 0002FF7F | 10E0 | | | 0006FFFD | 10B0 | 0006FFFE | 10A0 |
| 80<br>81<br>82 | 48 | 0002FF7F | 1100 | CC = 0<br>Wins<br>No tie | Tie<br>0006FFAF | 1100 | | | <br><br>0002FF7F | <br><br>10F0 | <br><br>0002FFFE | <br><br>10D0 | 0002FF7F<br>0002FF7F<br>0006FFFD | 1100<br>10F0<br>10B0 |
| 90 | 30 | 800010C0 | 10C0 | No tie | | | * | | | | * | | 800010C0 | 10C0 |
| 100 | 30 | 00020000 | 1140 | No tie | | | 00020000 | 1140 | | | 0002FF7F | 10E0 | 0002FFFE | 10D0 |
| 110<br>111<br>112<br>113<br>114 | 28 | 0002FF7F | 1110 | CC = 0<br>Wins<br>CC = 0<br>Wins<br>No comp | | | | | Tie<br>0004FFFC | 1110 | <br><br>Tie<br>0004FFFD | <br><br><br>10F0 | 0002FF7F<br>0002FF7F<br>0002FF7F<br>0002FF7F<br>0002FF7F | 1110<br>10F0<br>10F0<br>10E0<br>10E0 |
| 120<br>121<br>122 | 38 | 00020000 | 1130 | CC = 0<br>Loses<br>No tie | | | Tie<br>00060000 | | | | <br><br>00020000 | <br><br>1130 | 00020000<br>00020000<br>0004FFFD | 1130<br>1130<br>10F0 |
| 130 | 40 | 0002FF6F | 1120 | No tie | 0002FF6F | 1120 | | | * | | * | | 0006FFAF | 1100 |
| 140 | 48 | 00020000 | 1140 | No tie | 00020000 | 1140 | | | 0002FF6F | 1120 | * | | 0004FFFC | 1110 |
| 150 | 28 | 00020000 | 1140 | No tie | | | | | 00020000 | 1140 | * | | 0002FF6F | 1120 |
| 160<br>161<br>162<br>163<br>164<br>165<br>166 | 40 | 00020000 | 1140 | CC = 0<br>Equal<br>CC = 0<br>Equal<br>CC = 0<br>Wins<br>No comp | Tie<br>80001140 | | | | <br><br>Tie<br>80001140 | | <br><br><br><br>Tie<br>00060000 | <br><br><br><br><br>1140 | 00020000<br>00020000<br>00020000<br>00020000<br>00020000<br>00020000<br>00020000 | 1140<br>1140<br>1140<br>1140<br>1140<br>1130<br>1130 |
| 170 | 38 | 00020000 | 1050 | Branch | | | | | | | | | | |

*Figure A-9 (Part 1 of 2). Abbreviated Trace of Main Merge Processing*


LzLabs GmbH. Ex. 1008-907

**Explanation:**

| | |
|---|---|
| 1 | Line 0 shows the values in the tree after it is primed. |
| * | Means no swap. |
| ** | Means no swap if UPDATE TREE method 1 is used or no examination if UPDATE TREE method 2 is used. Only method 2 is included in ESA/370 and ESA/390. |
| CC = 0 | UPDATE TREE finds a tie and sets condition code 0. |
| Loses | The tied-codeword routine finds that the node loses. |
| Wins | The tied-codeword routine finds that the node wins. |
| Equal | The tied-codeword routine finds that the keys are equal. |
| Branch | Branches to terminate at 118A on sequence break. |
| No comp | No compare. |

*Figure A-9 (Part 2 of 2). Abbreviated Trace of Main Merge Processing*



LzLabs GmbH. Ex. 1008-908

# Appendix B. Lists of Instructions

The following figures list instructions by name, mnemonic, and operation code. Some models may offer instructions that do not appear in the figures, such as those provided for assists or as part of special or custom features.

The operation codes for the vector facility, compression facility, and interpretive execution are not included in this appendix. See the publications *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207, *IBM Enterprise Systems Architecture/390 Data Compression*, SA22-7208, and *IBM System/370 Extended Architecture Interpretive Execution*, SA22-7095, for operation codes associated with those facilities.

The operation code 00 hex with a two-byte instruction format is allocated for use by the program when an indication of an invalid operation is required. It is improbable that this operation code will ever be assigned to an instruction implemented in the CPU.

***Explanation of Symbols in "Characteristics" and "Page" Columns:***

- ¢ Causes serialization and checkpoint synchronization.
- ¢1 Causes serialization and checkpoint synchronization when the $M_1$ and $R_2$ fields contain all ones and all zeros, respectively.
- ¢2 Causes serialization and checkpoint synchronization when the state entry to be unstacked is a program-call state entry.
- $ Causes serialization.
- A Access exceptions for logical addresses.
- A1 Access exceptions; not all access exceptions may occur; see instruction description for details.
- AI Access exceptions for instruction address.
- AS ASN-translation-specification and special-operation exceptions.
- AT ASN-translation-specification exception.
- B PER branch event.
- $B_1$ $B_1$ field designates an access register in the access-register mode.
- $B_2$ $B_2$ field designates an access register in the access-register mode.
- BF BFP facility.
- BP $B_2$ field designates an access register when PSW bits 16 and 17 have the value 01.
- BS Branch-and-set-authority facility.
- C Condition code is set.
- CA CPU cryptographic assist.
- CI Cancel-I/O facility.
- CK Checksum facility.
- CM Compare-and-move-extended facility.
- Da AFP-register data exception.
- Db BFP-instruction data exception.
- Dd Decimal-operand data exception.
- DF Decimal-overflow exception.
- DK Decimal-divide exception.
- DM Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code.
- E E instruction format.
- EK Extended-TOD-clock facility.
- EO HFP-exponent-overflow exception.
- ES Expanded-storage facility.
- ET Extended-translation facility 1.
- E2 Extended-translation facility 2.
- EU HFP-exponent-underflow exception.
- EX Execute exception.
- FK HFP-floating-point-divide exception.
- FX Floating-point-support extensions facility.
- G0 Instruction execution includes the implied use of general register 0.
- G1 Instruction execution includes the implied use of general register 1.
- G2 Instruction execution includes the implied use of general register 2.
- G4 Instruction execution includes the implied use of general register 4.
- GM Instruction execution includes the implied use of multiple general registers.
- GS Instruction execution includes the implied use of general register 1 as the subsystem-identification word.
- HM HFP-multiply-add/subtract facility.
- HX HFP-extensions facility.
- IF Fixed-point-overflow exception.
- II Interruptible instruction.
- IK Fixed-point-divide exception.
- IR Immediate-and-relative-instruction facility.
- I1 Access register 1 is implicitly designated in the access-register mode.

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-909

| | |
|---|---|
| I4 | Access register 4 is implicitly designated in the access-register mode. |
| L | New condition code is loaded. |
| LS | HFP-significance exception. |
| MD | Designation of access registers in the access-register mode is model-dependent. |
| MO | Monitor event. |
| M1 | Move-page facility 1. |
| M2 | Move-page facility 2. |
| N3 | Instruction is added to ESA/390 from z/Architecture. |
| OP | Operand exception. |
| P | Privileged-operation exception. |
| PC | Program-call-fast facility. |
| PL | Perform-locked-operation facility. |
| Q | Privileged-operation exception for semi-privileged instructions. |
| QR | Square-root facility. |
| R | PER general-register alteration event. |
| $R_1$ | $R_1$ field designates an access register in the access-register mode. |
| $R_2$ | $R_2$ field designates an access register in the access-register mode. |
| RI | RI instruction format. |
| RIL | RIL instruction format. |
| RP | Resume-program facility. |
| RR | RR instruction format. |
| RRE | RRE instruction format. |
| RRF | RRF instruction format. |
| RS | RS instruction format. |
| RSE | RSE instruction format. |
| RSL | RSL instruction format. |
| RX | RX instruction format. |
| RXE | RXE instruction format. |
| RXF | RXF instruction format. |
| S | S instruction format. |
| SA | Set-address-space-control-fast facility. |
| SE | Special operation, stack-empty, stack-specification, and stack-type exceptions. |
| SF | Special-operation, stack-full, and stack-specification exceptions. |
| SG | Subspace-group facility. |
| SI | SI instruction format. |
| SN | Store-system-information facility. |
| SO | Special-operation exception. |
| SP | Specification exception. |
| SQ | HFP-square-root exception. |
| SR | String-instruction facility. |
| SS | SS instruction format. |
| SSE | SSE instruction format. |
| ST | PER storage-alteration event. |
| SU | PER store-using-real-address event. |
| SW | Special-operation exception and space-switch event. |
| T | Trace exceptions (which include trace table, addressing, and low-address protection). |
| TR | Trap facility |
| U | Condition code is unpredictable. |
| $U_1$ | $R_1$ field designates an access register unconditionally. |
| $U_2$ | $R_2$ field designates an access register unconditionally. |
| UB | $R_1$ and $R_3$ fields designate access registers unconditionally, and $B_2$ field designates an access register in the access-register mode. |
| Xi | IEEE invalid-operation condition. |
| Xo | IEEE overflow condition. |
| Xu | IEEE underflow condition. |
| Xx | IEEE inexact condition. |
| Xz | IEEE division-by-zero condition. |
| $Z^1$ | Additional exceptions and events for PROGRAM CALL (which include AFX-translation, ASN-translation-specification, ASX-translation, EX-translation, LX-translation, PC-translation-specification, special-operation, stack-full, and stack-specification exceptions and space-switch event). |
| $Z^2$ | Additional exceptions and events for PROGRAM TRANSFER (which include AFX-translation, ASN-translation-specification, ASX-translation, primary-authority, and special-operation exceptions and space-switch event). |
| $Z^3$ | Additional exceptions for SET SECONDARY ASN (which include AFX translation, ASN translation specification, ASX translation, secondary authority, and special operation). |
| $Z^4$ | Additional exceptions and events for PROGRAM RETURN (which include AFX-translation, ASN-translation-specification, ASX-translation, secondary-authority, special-operation, stack-empty, stack-operation, stack-specification, and stack-type exceptions and space-switch event). |
| $Z^5$ | Additional exceptions and events for PROGRAM CALL FAST (which includ EX-translation, special-operation, stack- |



LzLabs GmbH. Ex. 1008-910

full, and stack-specification exceptions and space-switch event).

| Name | Mnemonic | Characteristics | Op Code | Page No. |
|---|---|---|---|---|
| ADD | AR | RR C IF R | 1A | 7-12 |
| ADD | A | RX C A IF R $B_2$ | 5A | 7-12 |
| ADD (extended BFP) | AXBR | RRE C BF SP Db Xi Xo Xu Xx | B34A | 19-18 |
| ADD (long BFP) | ADBR | RRE C BF Db Xi Xo Xu Xx | B31A | 19-18 |
| ADD (long BFP) | ADB | RXE C BF A Db Xi Xo Xu Xx $B_2$ | ED1A | 19-18 |
| ADD (short BFP) | AEBR | RRE C BF Db Xi Xo Xu Xx | B30A | 19-18 |
| ADD (short BFP) | AEB | RXE C BF A Db Xi Xo Xu Xx $B_2$ | ED0A | 19-18 |
| ADD DECIMAL | AP | SS C A Dd DF ST $B_1$ $B_2$ | FA | 8-6 |
| ADD HALFWORD | AH | RX C A IF R $B_2$ | 4A | 7-12 |
| ADD HALFWORD IMMEDIATE | AHI | RI C IR IF R | A7A | 7-12 |
| ADD LOGICAL | ALR | RR C R | 1E | 7-13 |
| ADD LOGICAL | AL | RX C A R $B_2$ | 5E | 7-13 |
| ADD LOGICAL WITH CARRY | ALCR | RRE C N3 R | B998 | 7-13 |
| ADD LOGICAL WITH CARRY | ALC | RXE C N3 A R $B_2$ | E398 | 7-13 |
| ADD NORMALIZED (extended HFP) | AXR | RR C SP Da EU EO LS | 36 | 18-8 |
| ADD NORMALIZED (long HFP) | ADR | RR C SP Da EU EO LS | 2A | 18-8 |
| ADD NORMALIZED (long HFP) | AD | RX C A SP Da EU EO LS $B_2$ | 6A | 18-8 |
| ADD NORMALIZED (short HFP) | AER | RR C SP Da EU EO LS | 3A | 18-8 |
| ADD NORMALIZED (short HFP) | AE | RX C A SP Da EU EO LS $B_2$ | 7A | 18-8 |
| ADD UNNORMALIZED (long HFP) | AWR | RR C SP Da EO LS | 2E | 18-10 |
| ADD UNNORMALIZED (long HFP) | AW | RX C A SP Da EO LS $B_2$ | 6E | 18-10 |
| ADD UNNORMALIZED (short HFP) | AUR | RR C SP Da EO LS | 3E | 18-10 |
| ADD UNNORMALIZED (short HFP) | AU | RX C A SP Da EO LS $B_2$ | 7E | 18-10 |
| AND | NR | RR C R | 14 | 7-13 |
| AND | N | RX C A R $B_2$ | 54 | 7-13 |
| AND (character) | NC | SS C A ST $B_1$ $B_2$ | D4 | 7-13 |
| AND (immediate) | NI | SI C A ST $B_1$ | 94 | 7-13 |
| BRANCH AND LINK | BALR | RR T B R | 05 | 7-14 |
| BRANCH AND LINK | BAL | RX B R | 45 | 7-14 |
| BRANCH AND SAVE | BASR | RR T B R | 0D | 7-15 |
| BRANCH AND SAVE | BAS | RX B R | 4D | 7-15 |
| BRANCH AND SAVE AND SET MODE | BASSM | RR T B R | 0C | 7-16 |
| BRANCH AND SET AUTHORITY | BSA | RRE BS Q A[1] SO T B R | B25A | 10-7 |
| BRANCH AND SET MODE | BSM | RR B R | 0B | 7-16 |
| BRANCH AND STACK | BAKR | RRE A[1] SF T B ST | B240 | 10-10 |
| BRANCH IN SUBSPACE GROUP | BSG | RRE SG A[1] SO T B R $R_2$ | B258 | 10-13 |
| BRANCH ON CONDITION | BCR | RR ¢[1] B | 07 | 7-17 |
| BRANCH ON CONDITION | BC | RX B | 47 | 7-17 |
| BRANCH ON COUNT | BCTR | RR B R | 06 | 7-18 |
| BRANCH ON COUNT | BCT | RX B R | 46 | 7-18 |
| BRANCH ON INDEX HIGH | BXH | RS B R | 86 | 7-18 |
| BRANCH ON INDEX LOW OR EQUAL | BXLE | RS B R | 87 | 7-18 |
| BRANCH RELATIVE AND SAVE | BRAS | RI IR B R | A75 | 7-19 |
| BRANCH RELATIVE AND SAVE LONG | BRASL | RIL N3 B R | C05 | 7-19 |
| BRANCH RELATIVE ON CONDITION | BRC | RI IR B | A74 | 7-20 |

*Figure B-1 (Part 1 of 10). Instructions Arranged by Name*


LzLabs GmbH. Ex. 1008-911

| Name | Mne-monic | Characteristics | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|
| BRANCH RELATIVE ON CONDITION LONG | BRCL | RIL N3 | | | B | | C04 | 7-20 |
| BRANCH RELATIVE ON COUNT | BRCT | RI IR | | | B R | | A76 | 7-21 |
| BRANCH RELATIVE ON INDEX HIGH | BRXH | RSI IR | | | B R | | 84 | 7-21 |
| BRANCH RELATIVE ON INDEX LOW OR EQ. | BRXLE | RSI IR | | | B R | | 85 | 7-21 |
| CHECKSUM | CKSM | RRE C CK | A SP | | R | $R_2$ | B241 | 7-22 |
| CIPHER MESSAGE | KM | RRE C MS | A SP | GM I1 | ST | $R_1$ $R_2$ | B92E | 7-26 |
| CIPHER MESSAGE WITH CHAINING | KMC | RRE C MS | A SP | GM I1 | ST | $R_1$ $R_2$ | B92F | 7-26 |
| COMPARE | CR | RR C | | | | | 19 | 7-35 |
| COMPARE | C | RX C | A | | | $B_2$ | 59 | 7-35 |
| COMPARE (extended BFP) | CXBR | RRE C BF | SP | Db Xi | | | B349 | 19-23 |
| COMPARE (extended HFP) | CXR | RRE C HX | SP | Da | | | B369 | 18-10 |
| COMPARE (long BFP) | CDBR | RRE C BF | | Db Xi | | | B319 | 19-23 |
| COMPARE (long BFP) | CDB | RXE C BF | A | Db Xi | | $B_2$ | ED19 | 19-23 |
| COMPARE (long HFP) | CDR | RR C | SP | Da | | | 29 | 18-10 |
| COMPARE (long HFP) | CD | RX C | A SP | Da | | $B_2$ | 69 | 18-10 |
| COMPARE (short BFP) | CEBR | RRE C BF | | Db Xi | | | B309 | 19-23 |
| COMPARE (short BFP) | CEB | RXE C BF | A | Db Xi | | $B_2$ | ED09 | 19-23 |
| COMPARE (short HFP) | CER | RR C | SP | Da | | | 39 | 18-10 |
| COMPARE (short HFP) | CE | RX C | A SP | Da | | $B_2$ | 79 | 18-10 |
| COMPARE AND FORM CODEWORD | CFC | S C | A SP | II GM | R | I1 | B21A | 7-36 |
| COMPARE AND SIGNAL (extended BFP) | KXBR | RRE C BF | SP | Db Xi | | | B348 | 19-24 |
| COMPARE AND SIGNAL (long BFP) | KDBR | RRE C BF | | Db Xi | | | B318 | 19-24 |
| COMPARE AND SIGNAL (long BFP) | KDB | RXE C BF | A | Db Xi | | $B_2$ | ED18 | 19-24 |
| COMPARE AND SIGNAL (short BFP) | KEBR | RRE C BF | | Db Xi | | | B308 | 19-24 |
| COMPARE AND SIGNAL (short BFP) | KEB | RXE C BF | A | Db Xi | | $B_2$ | ED08 | 19-24 |
| COMPARE AND SWAP | CS | RS C | A SP | $ | R ST | $B_2$ | BA | 7-40 |
| COMPARE AND SWAP AND PURGE | CSP | RRE C | P $A^1$ SP | $ | R ST | $R_2$ | B250 | 10-17 |
| COMPARE DECIMAL | CP | SS C | A | Dd | | $B_1$ $B_2$ | F9 | 8-6 |
| COMPARE DOUBLE AND SWAP | CDS | RS C | A SP | $ | R ST | $B_2$ | BB | 7-40 |
| COMPARE HALFWORD | CH | RX C | A | | | $B_2$ | 49 | 7-42 |
| COMPARE HALFWORD IMMEDIATE | CHI | RI C IR | | | | | A7E | 7-42 |
| COMPARE LOGICAL | CLR | RR C | | | | | 15 | 7-42 |
| COMPARE LOGICAL | CL | RX C | A | | | $B_2$ | 55 | 7-42 |
| COMPARE LOGICAL (character) | CLC | SS C | A | | | $B_1$ $B_2$ | D5 | 7-42 |
| COMPARE LOGICAL (immediate) | CLI | SI C | A | | | $B_1$ | 95 | 7-42 |
| COMPARE LOGICAL CHARS. UNDER MASK | CLM | RS C | A | | | $B_2$ | BD | 7-43 |
| COMPARE LOGICAL LONG | CLCL | RR C | A SP | II | R | $R_1$ $R_2$ | 0F | 7-43 |
| COMPARE LOGICAL LONG EXTENDED | CLCLE | RS C CM | A SP | | R | $R_1$ $R_3$ | A9 | 7-45 |
| COMPARE LOGICAL LONG UNICODE | CLCLU | RSE C E2 | A SP | | R | $R_1$ $R_2$ | EB8F | 7-47 |
| COMPARE LOGICAL STRING | CLST | RRE C SR | A SP | G0 | R | $R_1$ $R_2$ | B25D | 7-50 |
| COMPARE UNTIL SUBSTRING EQUAL | CUSE | RRE C | A SP | II GM | | $R_1$ $R_2$ | B257 | 7-51 |
| COMPUTE INTERMEDIATE MESSAGE DIGEST | KIMD | RRE C MS | A SP | GM I1 | ST | $R_2$ | B93E | 7-55 |
| COMPUTE LAST MESSAGE DIGEST | KLMD | RRE C MS | A SP | GM I1 | ST | $R_2$ | B93F | 7-55 |
| COMPUTE MESSAGE AUTHENTICATION CODE | KMAC | RRE C MS | A SP | GM I1 | ST | $R_2$ | B91E | 7-61 |
| CONVERT BFP TO HFP (long) | THDR | RRE C FX | | Da | | | B359 | 9-8 |

*Figure B-1 (Part 2 of 10). Instructions Arranged by Name*



LzLabs GmbH. Ex. 1008-912

| Name | Mne-monic | Characteristics | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|
| CONVERT BFP TO HFP (short to long) | THDER | RRE C FX | | Da | | | B358 | 9-8 |
| CONVERT FROM FIXED (32 to ext. BFP) | CXFBR | RRE BF | SP | Db | | | B396 | 19-26 |
| CONVERT FROM FIXED (32 to ext. HFP) | CXFR | RRE HX | SP | Da | | | B3B6 | 18-11 |
| CONVERT FROM FIXED (32 to long BFP) | CDFBR | RRE BF | | Db | | | B395 | 19-26 |
| CONVERT FROM FIXED (32 to long HFP) | CDFR | RRE HX | | Da | | | B3B5 | 18-11 |
| CONVERT FROM FIXED (32 to short BFP) | CEFBR | RRE BF | | Db Xx | | | B394 | 19-26 |
| CONVERT FROM FIXED (32 to short HFP) | CEFR | RRE HX | | Da | | | B3B4 | 18-11 |
| CONVERT HFP TO BFP (long to short) | TBEDR | RRF C FX | SP | Da | | | B350 | 9-9 |
| CONVERT HFP TO BFP (long) | TBDR | RRF C FX | SP | Da | | | B351 | 9-9 |
| CONVERT TO BINARY | CVB | RX | A | Dd IK | R | $B_2$ | 4F | 7-66 |
| CONVERT TO DECIMAL | CVD | RX | A | | ST | $B_2$ | 4E | 7-67 |
| CONVERT TO FIXED (ext. BFP to 32) | CFXBR | RRF C BF | SP | Db Xi Xx | R | | B39A | 19-26 |
| CONVERT TO FIXED (ext. HFP to 32) | CFXR | RRF C HX | SP | Da | R | | B3BA | 18-12 |
| CONVERT TO FIXED (long BFP to 32) | CFDBR | RRF C BF | SP | Db Xi Xx | R | | B399 | 19-26 |
| CONVERT TO FIXED (long HFP to 32) | CFDR | RRF C HX | SP | Da | R | | B3B9 | 18-12 |
| CONVERT TO FIXED (short BFP to 32) | CFEBR | RRF C BF | SP | Db Xi Xx | R | | B398 | 19-26 |
| CONVERT TO FIXED (short HFP to 32) | CFER | RRF C HX | SP | Da | R | | B3B8 | 18-12 |
| CONVERT UNICODE TO UTF-8 | CUUTF | RRE C ET | A SP | | R ST | $R_1$ $R_2$ | B2A6 | 7-67 |
| CONVERT UTF-8 TO UNICODE | CUTFU | RRE C ET | A SP | | R ST | $R_1$ $R_2$ | B2A7 | 7-70 |
| COPY ACCESS | CPYA | RRE | | | | $U_1$ $U_2$ | B24D | 7-72 |
| DIAGNOSE | | DM | P DM | | | MD | 83 | 10-19 |
| DIVIDE | DR | RR | SP | IK | R | | 1D | 7-73 |
| DIVIDE | D | RX | A SP | IK | R | $B_2$ | 5D | 7-73 |
| DIVIDE (extended BFP) | DXBR | RRE BF | SP | Db Xi Xz Xo Xu Xx | | | B34D | 19-29 |
| DIVIDE (extended HFP) | DXR | RRE | SP | Da EU EO FK | | | B22D | 18-12 |
| DIVIDE (long BFP) | DDBR | RRE BF | | Db Xi Xz Xo Xu Xx | | | B31D | 19-29 |
| DIVIDE (long BFP) | DDB | RXE BF | A | Db Xi Xz Xo Xu Xx | | $B_2$ | ED1D | 19-29 |
| DIVIDE (long HFP) | DDR | RR | SP | Da EU EO FK | | | 2D | 18-12 |
| DIVIDE (long HFP) | DD | RX | A SP | Da EU EO FK | | $B_2$ | 6D | 18-12 |
| DIVIDE (short BFP) | DEBR | RRE BF | | Db Xi Xz Xo Xu Xx | | | B30D | 19-29 |
| DIVIDE (short BFP) | DEB | RXE BF | A | Db Xi Xz Xo Xu Xx | | $B_2$ | ED0D | 19-29 |
| DIVIDE (short HFP) | DER | RR | SP | Da EU EO FK | | | 3D | 18-12 |
| DIVIDE (short HFP) | DE | RX | A SP | Da EU EO FK | | $B_2$ | 7D | 18-12 |
| DIVIDE DECIMAL | DP | SS | A SP | Dd DK | ST | $B_1$ $B_2$ | FD | 8-7 |
| DIVIDE LOGICAL | DLR | RRE N3 | SP | IK | R | | B997 | 7-73 |
| DIVIDE LOGICAL | DL | RXE N3 | A SP | IK | R | $B_2$ | E397 | 7-73 |
| DIVIDE TO INTEGER (long BFP) | DIDBR | RRF C BF | SP | Db Xi Xu Xx | | | B35B | 19-30 |
| DIVIDE TO INTEGER (short BFP) | DIEBR | RRF C BF | SP | Db Xi Xu Xx | | | B353 | 19-30 |
| EDIT | ED | SS C | A | Dd | ST | $B_1$ $B_2$ | DE | 8-7 |
| EDIT AND MARK | EDMK | SS C | A | Dd G1 | R ST | $B_1$ $B_2$ | DF | 8-12 |
| EXCLUSIVE OR | XR | RR C | | | R | | 17 | 7-74 |
| EXCLUSIVE OR | X | RX C | A | | R | $B_2$ | 57 | 7-74 |
| EXCLUSIVE OR (character) | XC | SS C | A | | ST | $B_1$ $B_2$ | D7 | 7-74 |
| EXCLUSIVE OR (immediate) | XI | SI C | A | | ST | $B_1$ | 97 | 7-74 |
| EXECUTE | EX | RX | AI SP | EX | | | 44 | 7-74 |

*Figure B-1 (Part 3 of 10). Instructions Arranged by Name*



LzLabs GmbH. Ex. 1008-913

| Name | Mne-monic | Characteristics | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|
| EXTRACT ACCESS | EAR | RRE | | | R | $U_2$ | B24F | 7-75 |
| EXTRACT FPC | EFPC | RRE BF | | Db | | | B38C | 19-34 |
| EXTRACT PRIMARY ASN | EPAR | RRE | Q | SO | R | | B226 | 10-19 |
| EXTRACT PSW | EPSW | RRE N3 | | | R | | B98D | 7-76 |
| EXTRACT SECONDARY ASN | ESAR | RRE | Q | SO | R | | B227 | 10-20 |
| EXTRACT STACKED REGISTERS | EREG | RRE | A[1] | SE | R | $U_1$ $U_2$ | B249 | 10-20 |
| EXTRACT STACKED STATE | ESTA | RRE C | A[1] SP | SE | R | | B24A | 10-21 |
| HALVE (long HFP) | HDR | RR | SP | Da EU | | | 24 | 18-13 |
| HALVE (short HFP) | HER | RR | SP | Da EU | | | 34 | 18-13 |
| INSERT ADDRESS SPACE CONTROL | IAC | RRE C | Q | SO | R | | B224 | 10-23 |
| INSERT CHARACTER | IC | RX | A | | R | $B_2$ | 43 | 7-76 |
| INSERT CHARACTERS UNDER MASK | ICM | RS C | A | | R | $B_2$ | BF | 7-76 |
| INSERT PROGRAM MASK | IPM | RRE | | | R | | B222 | 7-77 |
| INSERT PSW KEY | IPK | S | Q | G2 | R | | B20B | 10-24 |
| INSERT STORAGE KEY EXTENDED | ISKE | RRE | P A[1] | | | | B229 | 10-25 |
| INSERT VIRTUAL STORAGE KEY | IVSK | RRE | Q A[1] | SO | R | $R_2$ | B223 | 10-25 |
| INVALIDATE PAGE TABLE ENTRY | IPTE | RRE | P A[1] | $ | | | B221 | 10-26 |
| LOAD | LR | RR | | | R | | 18 | 7-77 |
| LOAD | L | RX | A | | R | $B_2$ | 58 | 7-77 |
| LOAD (extended) | LXR | RRE FX | SP | Da | | | B365 | 9-10 |
| LOAD (long) | LDR | RR | SP | Da | | | 28 | 9-10 |
| LOAD (long) | LD | RX | A SP | Da | | $B_2$ | 68 | 9-10 |
| LOAD (short) | LER | RR | SP | Da | | | 38 | 9-10 |
| LOAD (short) | LE | RX | A SP | Da | | $B_2$ | 78 | 9-10 |
| LOAD ACCESS MULTIPLE | LAM | RS | A SP | | | UB | 9A | 7-77 |
| LOAD ADDRESS | LA | RX | | | R | | 41 | 7-78 |
| LOAD ADDRESS EXTENDED | LAE | RX | | | R | $U_1$ BP | 51 | 7-78 |
| LOAD ADDRESS RELATIVE LONG | LARL | RIL N3 | | | R | | C00 | 7-79 |
| LOAD ADDRESS SPACE PARAMETERS | LASP | SSE C | P A[1] SP | AS | | $B_1$ | E500 | 10-28 |
| LOAD AND TEST | LTR | RR C | | | R | | 12 | 7-79 |
| LOAD AND TEST (extended BFP) | LTXBR | RRE C BF | SP | Db Xi | | | B342 | 19-35 |
| LOAD AND TEST (extended HFP) | LTXR | RRE C HX | SP | Da | | | B362 | 18-14 |
| LOAD AND TEST (long BFP) | LTDBR | RRE C BF | | Db Xi | | | B312 | 19-35 |
| LOAD AND TEST (long HFP) | LTDR | RR C | SP | Da | | | 22 | 18-14 |
| LOAD AND TEST (short BFP) | LTEBR | RRE C BF | | Db Xi | | | B302 | 19-35 |
| LOAD AND TEST (short HFP) | LTER | RR C | SP | Da | | | 32 | 18-14 |
| LOAD COMPLEMENT | LCR | RR C | | IF | R | | 13 | 7-79 |
| LOAD COMPLEMENT (extended BFP) | LCXBR | RRE C BF | SP | Db | | | B343 | 19-35 |
| LOAD COMPLEMENT (extended HFP) | LCXR | RRE C HX | SP | Da | | | B363 | 18-15 |
| LOAD COMPLEMENT (long BFP) | LCDBR | RRE C BF | | Db | | | B313 | 19-35 |
| LOAD COMPLEMENT (long HFP) | LCDR | RR C | SP | Da | | | 23 | 18-15 |
| LOAD COMPLEMENT (short BFP) | LCEBR | RRE C BF | | Db | | | B303 | 19-35 |
| LOAD COMPLEMENT (short HFP) | LCER | RR C | SP | Da | | | 33 | 18-15 |
| LOAD CONTROL | LCTL | RS | P A SP | | | $B_2$ | B7 | 10-36 |
| LOAD FP INTEGER (extended BFP) | FIXBR | RRF BF | SP | Db Xi Xx | | | B347 | 19-36 |

*Figure B-1 (Part 4 of 10). Instructions Arranged by Name*



LzLabs GmbH. Ex. 1008-914

| Name | Mne-monic | Characteristics | | | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|---|---|
| LOAD FP INTEGER (extended HFP) | FIXR | RRE | HX | | SP | Da | | | B367 | 18-15 |
| LOAD FP INTEGER (long BFP) | FIDBR | RRF | BF | | SP | Db Xi | Xx | | B35F | 19-36 |
| LOAD FP INTEGER (long HFP) | FIDR | RRE | HX | | | Da | | | B37F | 18-15 |
| LOAD FP INTEGER (short BFP) | FIEBR | RRF | BF | | SP | Db Xi | Xx | | B357 | 19-36 |
| LOAD FP INTEGER (short HFP) | FIER | RRE | HX | | | Da | | | B377 | 18-15 |
| LOAD FPC | LFPC | S | BF | A | SP | Db | | $B_2$ | B29D | 19-37 |
| LOAD HALFWORD | LH | RX | | A | | | R | $B_2$ | 48 | 7-80 |
| LOAD HALFWORD IMMEDIATE | LHI | RI | IR | | | | R | | A78 | 7-80 |
| LOAD LENGTHENED (long to ext. BFP) | LXDBR | RRE | BF | | SP | Db Xi | | | B305 | 19-38 |
| LOAD LENGTHENED (long to ext. BFP) | LXDB | RXE | BF | A | SP | Db Xi | | $B_2$ | ED05 | 19-38 |
| LOAD LENGTHENED (long to ext. HFP) | LXDR | RRE | HX | | SP | Da | | | B325 | 18-16 |
| LOAD LENGTHENED (long to ext. HFP) | LXD | RXE | HX | A | SP | Da | | $B_2$ | ED25 | 18-16 |
| LOAD LENGTHENED (short to ext. BFP) | LXEBR | RRE | BF | | SP | Db Xi | | | B306 | 19-38 |
| LOAD LENGTHENED (short to ext. BFP) | LXEB | RXE | BF | A | SP | Db Xi | | $B_2$ | ED06 | 19-38 |
| LOAD LENGTHENED (short to ext. HFP) | LXER | RRE | HX | | SP | Da | | | B326 | 18-16 |
| LOAD LENGTHENED (short to ext. HFP) | LXE | RXE | HX | A | SP | Da | | $B_2$ | ED26 | 18-16 |
| LOAD LENGTHENED (short to long BFP) | LDEBR | RRE | BF | | | Db Xi | | | B304 | 19-38 |
| LOAD LENGTHENED (short to long BFP) | LDEB | RXE | BF | A | | Db Xi | | $B_2$ | ED04 | 19-38 |
| LOAD LENGTHENED (short to long HFP) | LDER | RRE | HX | | | Da | | | B324 | 18-16 |
| LOAD LENGTHENED (short to long HFP) | LDE | RXE | HX | A | | Da | | $B_2$ | ED24 | 18-16 |
| LOAD MULTIPLE | LM | RS | | A | | | R | $B_2$ | 98 | 7-80 |
| LOAD NEGATIVE | LNR | RR | C | | | | R | | 11 | 7-80 |
| LOAD NEGATIVE (extended BFP) | LNXBR | RRE | C BF | | SP | Db | | | B341 | 19-38 |
| LOAD NEGATIVE (extended HFP) | LNXR | RRE | C HX | | SP | Da | | | B361 | 18-16 |
| LOAD NEGATIVE (long BFP) | LNDBR | RRE | C BF | | | Db | | | B311 | 19-38 |
| LOAD NEGATIVE (long HFP) | LNDR | RR | C | | SP | Da | | | 21 | 18-16 |
| LOAD NEGATIVE (short BFP) | LNEBR | RRE | C BF | | | Db | | | B301 | 19-38 |
| LOAD NEGATIVE (short HFP) | LNER | RR | C | | SP | Da | | | 31 | 18-16 |
| LOAD POSITIVE | LPR | RR | C | | | IF | R | | 10 | 7-81 |
| LOAD POSITIVE (extended BFP) | LPXBR | RRE | C BF | | SP | Db | | | B340 | 19-39 |
| LOAD POSITIVE (extended HFP) | LPXR | RRE | C HX | | SP | Da | | | B360 | 18-17 |
| LOAD POSITIVE (long BFP) | LPDBR | RRE | C BF | | | Db | | | B310 | 19-39 |
| LOAD POSITIVE (long HFP) | LPDR | RR | C | | SP | Da | | | 20 | 18-17 |
| LOAD POSITIVE (short BFP) | LPEBR | RRE | C BF | | | Db | | | B300 | 19-39 |
| LOAD POSITIVE (short HFP) | LPER | RR | C | | SP | Da | | | 30 | 18-17 |
| LOAD PSW | LPSW | S | L | P A | SP | ¢ | | $B_2$ | 82 | 10-37 |
| LOAD REAL ADDRESS | LRA | RX | C | P A[1] | | AT | R | BP | B1 | 10-38 |
| LOAD REVERSED | LRVR | RRE | N3 | | | | R | | B91F | 7-81 |
| LOAD REVERSED | LRVH | RXE | N3 | A | | | R | $B_2$ | E31F | 7-81 |
| LOAD REVERSED | LRV | RXE | N3 | A | | | R | $B_2$ | E31E | 7-81 |
| LOAD ROUNDED (extended to long BFP) | LDXBR | RRE | BF | | SP | Db Xi | Xo Xu Xx | | B345 | 19-39 |
| LOAD ROUNDED (extended to long HFP) | LDXR | RR | | | SP | Da | EO | | 25 | 18-18 |
| LOAD ROUNDED (extended to long HFP) | LRDR | RR | | | SP | Da | EO | | 25 | 18-18 |
| LOAD ROUNDED (extended to short BFP) | LEXBR | RRE | BF | | SP | Db Xi | Xo Xu Xx | | B346 | 19-39 |
| LOAD ROUNDED (extended to short HFP) | LEXR | RRE | HX | | SP | Da | EO | | B366 | 18-18 |

*Figure B-1 (Part 5 of 10). Instructions Arranged by Name*


LzLabs GmbH. Ex. 1008-915

| Name | Mnemonic | Characteristics | | | | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LOAD ROUNDED (long to short BFP) | LEDBR | RRE BF | | | Db Xi Xo Xu Xx | | | | | B344 | 19-39 |
| LOAD ROUNDED (long to short HFP) | LEDR | RR | | SP | Da EO | | | | | 35 | 18-18 |
| LOAD ROUNDED (long to short HFP) | LRER | RR | | SP | Da EO | | | | | 35 | 18-18 |
| LOAD USING REAL ADDRESS | LURA | RRE | P A[1] | SP | | R | | | | B24B | 10-40 |
| LOAD ZERO (extended) | LZXR | RRE FX | | SP | Da | | | | | B376 | 9-11 |
| LOAD ZERO (long) | LZDR | RRE FX | | | Da | | | | | B375 | 9-11 |
| LOAD ZERO (short) | LZER | RRE FX | | | Da | | | | | B374 | 9-11 |
| MODIFY STACKED STATE | MSTA | RRE | A[1] | SP | SE | ST | | | | B247 | 10-40 |
| MONITOR CALL | MC | SI | | SP | MO | | | | | AF | 7-82 |
| MOVE (character) | MVC | SS | A | | | ST | $B_1$ | $B_2$ | | D2 | 7-83 |
| MOVE (immediate) | MVI | SI | A | | | ST | $B_1$ | | | 92 | 7-83 |
| MOVE INVERSE | MVCIN | SS | A | | | ST | $B_1$ | $B_2$ | | E8 | 7-83 |
| MOVE LONG | MVCL | RR C | A | SP | II | R ST | $R_1$ | $R_2$ | | 0E | 7-83 |
| MOVE LONG EXTENDED | MVCLE | RS C CM | A | SP | | R ST | $R_1$ | $R_3$ | | A8 | 7-87 |
| MOVE LONG UNICODE | MVCLU | RSE C E2 | A | SP | | R ST | $R_1$ | $R_2$ | | EB8E | 7-90 |
| MOVE NUMERICS | MVN | SS | A | | | ST | $B_1$ | $B_2$ | | D1 | 7-93 |
| MOVE PAGE (facility 1) | MVPG | RRE C M1 | A[1] | SP | G0 | ST | $R_1$ | $R_2$ | | B254 | 7-93 |
| MOVE PAGE (facility 2) | MVPG | RRE C M2 | Q A[1] | SP | G0 | ST | $R_1$ | $R_2$ | | B254 | 7-93 |
| MOVE STRING | MVST | RRE C SR | A | SP | G0 | R ST | $R_1$ | $R_2$ | | B255 | 7-95 |
| MOVE TO PRIMARY | MVCP | SS C | Q A | | SO ¢ | ST | | | | DA | 10-45 |
| MOVE TO SECONDARY | MVCS | SS C | Q A | | SO ¢ | ST | | | | DB | 10-45 |
| MOVE WITH DESTINATION KEY | MVCDK | SSE | Q A | | GM | ST | $B_1$ | $B_2$ | | E50F | 10-47 |
| MOVE WITH KEY | MVCK | SS C | Q A | | | ST | $B_1$ | $B_2$ | | D9 | 10-47 |
| MOVE WITH OFFSET | MVO | SS | A | | | ST | $B_1$ | $B_2$ | | F1 | 7-97 |
| MOVE WITH SOURCE KEY | MVCSK | SSE | Q A | | GM | ST | $B_1$ | $B_2$ | | E50E | 10-48 |
| MOVE ZONES | MVZ | SS | A | | | ST | $B_1$ | $B_2$ | | D3 | 7-97 |
| MULTIPLY | MR | RR | | SP | | R | | | | 1C | 7-98 |
| MULTIPLY | M | RX | A | SP | | R | | $B_2$ | | 5C | 7-98 |
| MULTIPLY (extended BFP) | MXBR | RRE BF | | SP | Db Xi Xo Xu Xx | | | | | B34C | 19-40 |
| MULTIPLY (extended HFP) | MXR | RR | | SP | Da EU EO | | | | | 26 | 18-18 |
| MULTIPLY (long to extended BFP) | MXDBR | RRE BF | | SP | Db Xi | | | | | B307 | 19-40 |
| MULTIPLY (long to extended BFP) | MXDB | RXE BF | A | SP | Db Xi | | | $B_2$ | | ED07 | 19-40 |
| MULTIPLY (long to extended HFP) | MXDR | RR | | SP | Da EU EO | | | | | 27 | 18-18 |
| MULTIPLY (long to extended HFP) | MXD | RX | A | SP | Da EU EO | | | $B_2$ | | 67 | 18-18 |
| MULTIPLY (long BFP) | MDBR | RRE BF | | | Db Xi Xo Xu Xx | | | | | B31C | 19-40 |
| MULTIPLY (long BFP) | MDB | RXE BF | A | | Db Xi Xo Xu Xx | | | $B_2$ | | ED1C | 19-40 |
| MULTIPLY (long HFP) | MDR | RR | | SP | Da EU EO | | | | | 2C | 18-18 |
| MULTIPLY (long HFP) | MD | RX | A | SP | Da EU EO | | | $B_2$ | | 6C | 18-18 |
| MULTIPLY (short to long BFP) | MDEBR | RRE BF | | | Db Xi | | | | | B30C | 19-40 |
| MULTIPLY (short to long BFP) | MDEB | RXE BF | A | | Db Xi | | | $B_2$ | | ED0C | 19-40 |
| MULTIPLY (short to long HFP) | MDER | RR | | SP | Da EU EO | | | | | 3C | 18-18 |
| MULTIPLY (short to long HFP) | MER | RR | | SP | Da EU EO | | | | | 3C | 18-18 |
| MULTIPLY (short to long HFP) | MDE | RX | A | SP | Da EU EO | | | $B_2$ | | 7C | 18-18 |
| MULTIPLY (short to long HFP) | ME | RX | A | SP | Da EU EO | | | $B_2$ | | 7C | 18-18 |
| MULTIPLY (short BFP) | MEEBR | RRE BF | | | Db Xi Xo Xu Xx | | | | | B317 | 19-40 |

*Figure B-1 (Part 6 of 10). Instructions Arranged by Name*



LzLabs GmbH. Ex. 1008-916

| Name | Mnemonic | Characteristics | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|
| MULTIPLY (short BFP) | MEEB | RXE BF | A | Db Xi Xo Xu Xx | B2 | ED17 | 19-40 |
| MULTIPLY (short HFP) | MEER | RRE HX | | Da EU EO | | B337 | 18-18 |
| MULTIPLY (short HFP) | MEE | RXE HX | A | Da EU EO | B2 | ED37 | 18-18 |
| MULTIPLY AND ADD (long BFP) | MADBR | RRF BF | | Db Xi Xo Xu Xx | | B31E | 19-42 |
| MULTIPLY AND ADD (long BFP) | MADB | RXF BF | A | Db Xi Xo Xu Xx | B2 | ED1E | 19-42 |
| MULTIPLY AND ADD (long HFP) | MADR | RRF HM | | Da EU EO | | B33E | 18-20 |
| MULTIPLY AND ADD (long HFP) | MAD | RXF HM | A | Da EU EO | B2 | ED3E | 18-20 |
| MULTIPLY AND ADD (short BFP) | MAEBR | RRF BF | | Db Xi Xo Xu Xx | | B30E | 19-42 |
| MULTIPLY AND ADD (short BFP) | MAEB | RXF BF | A | Db Xi Xo Xu Xx | B2 | ED0E | 19-42 |
| MULTIPLY AND ADD (short HFP) | MAER | RRF HM | | Da EU EO | | B32E | 18-20 |
| MULTIPLY AND ADD (short HFP) | MAE | RXF HM | A | Da EU EO | B2 | ED2E | 18-20 |
| MULTIPLY AND SUBTRACT (long BFP) | MSDBR | RRF BF | | Db Xi Xo Xu Xx | | B31F | 19-42 |
| MULTIPLY AND SUBTRACT (long BFP) | MSDB | RXF BF | A | Db Xi Xo Xu Xx | B2 | ED1F | 19-42 |
| MULTIPLY AND SUBTRACT (long HFP) | MSDR | RRF HM | | Da EU EO | | B33F | 18-20 |
| MULTIPLY AND SUBTRACT (long HFP) | MSD | RXF HM | A | Da EU EO | B2 | ED3F | 18-20 |
| MULTIPLY AND SUBTRACT (short BFP) | MSEBR | RRF BF | | Db Xi Xo Xu Xx | | B30F | 19-42 |
| MULTIPLY AND SUBTRACT (short BFP) | MSEB | RXF BF | A | Db Xi Xo Xu Xx | B2 | ED0F | 19-42 |
| MULTIPLY AND SUBTRACT (short HFP) | MSER | RRF HM | | Da EU EO | | B32F | 18-20 |
| MULTIPLY AND SUBTRACT (short HFP) | MSE | RXF HM | A | Da EU EO | B2 | ED2F | 18-20 |
| MULTIPLY DECIMAL | MP | SS | A SP | Dd | ST B1 B2 | FC | 8-12 |
| MULTIPLY HALFWORD | MH | RX | A | | R B2 | 4C | 7-98 |
| MULTIPLY HALFWORD IMMEDIATE | MHI | RI IR | | | R | A7C | 7-98 |
| MULTIPLY LOGICAL | MLR | RRE N3 | SP | | R | B996 | 7-99 |
| MULTIPLY LOGICAL | ML | RXE N3 | A SP | | R B2 | E396 | 7-99 |
| MULTIPLY SINGLE | MSR | RRE IR | | | R | B252 | 7-99 |
| MULTIPLY SINGLE | MS | RX IR | A | | R B2 | 71 | 7-99 |
| OR | OR | RR C | | | R | 16 | 7-100 |
| OR | O | RX C | A | | R B2 | 56 | 7-100 |
| OR (character) | OC | SS C | A | | ST B1 B2 | D6 | 7-100 |
| OR (immediate) | OI | SI C | A | | ST B1 | 96 | 7-100 |
| PACK | PACK | SS | A | | ST B1 B2 | F2 | 7-100 |
| PACK ASCII | PKA | SS E2 | A SP | | ST B1 B2 | E9 | 7-101 |
| PACK UNICODE | PKU | SS E2 | A SP | | ST B1 B2 | E1 | 7-102 |
| PAGE IN | PGIN | RRE C ES | P A1 | ¢ | | B22E | 10-50 |
| PAGE OUT | PGOUT | RRE C ES | P A1 | ¢ | | B22F | 10-51 |
| PERFORM LOCKED OPERATION | PLO | SS C PL | A SP | $ GM | R ST | FC | EE | 7-103 |
| PROGRAM CALL | PC | S | Q A1 | Z1 T ¢ GM | B R ST | | B218 | 10-52 |
| PROGRAM CALL FAST | PCF | S PC | A1 | Z5 ¢ G4 | B R ST | | B218 | 10-63 |
| PROGRAM RETURN | PR | E U | A1 SP | Z4 T ¢2 | B R ST | | 0101 | 10-67 |
| PROGRAM TRANSFER | PT | RRE | Q A1 SP | Z2 T ¢ | B | | B228 | 10-70 |
| PURGE ALB | PALB | RRE | P | $ | | | B248 | 10-76 |
| PURGE TLB | PTLB | S | P | $ | | | B20D | 10-76 |
| RESET REFERENCE BIT EXTENDED | RRBE | RRE C | P A1 | | | | B22A | 10-76 |
| RESUME PROGRAM | RP | S L RP | Q A SP | SW T | B R | B2 | B277 | 10-77 |
| ROTATE LEFT SINGLE LOGICAL | RLL | RSE N3 | | | R | | EB1D | 7-116 |

*Figure B-1 (Part 7 of 10). Instructions Arranged by Name*



LzLabs GmbH. Ex. 1008-917

| Name | Mne-monic | Characteristics | | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|---|
| SEARCH STRING | SRST | RRE C SR | A SP | G0 | R | R2 | | B25E | 7-116 |
| SET ACCESS | SAR | RRE | | | | U1 | | B24E | 7-117 |
| SET ADDRESS SPACE CONTROL | SAC | S | Q SP | SW ¢ | | | | B219 | 10-79 |
| SET ADDRESS SPACE CONTROL FAST | SACF | S SA | Q SP | SW | | | | B279 | 10-79 |
| SET ADDRESSING MODE | SAM24 | E N3 | SP | T | | | | 010C | 7-117 |
| SET ADDRESSING MODE | SAM31 | E N3 | SP | T | | | | 010D | 7-117 |
| SET CLOCK | SCK | S C | P A SP | | | B2 | | B204 | 10-81 |
| SET CLOCK COMPARATOR | SCKC | S | P A SP | | | B2 | | B206 | 10-82 |
| SET CLOCK PROGRAMMABLE FIELD | SCKPF | E EK | P SP | G0 | | | | 0107 | 10-82 |
| SET CPU TIMER | SPT | S | P A SP | | | B2 | | B208 | 10-82 |
| SET FPC | SFPC | RRE BF | SP | Db | | | | B384 | 19-44 |
| SET PREFIX | SPX | S | P A SP | $ | | B2 | | B210 | 10-83 |
| SET PROGRAM MASK | SPM | RR L | | | | | | 04 | 7-118 |
| SET PSW KEY FROM ADDRESS | SPKA | S | Q | | | | | B20A | 10-83 |
| SET ROUNDING MODE | SRNM | S BF | | Db | | | | B299 | 19-44 |
| SET SECONDARY ASN | SSAR | RRE | A1 | Z3 T ¢ | | | | B225 | 10-84 |
| SET STORAGE KEY EXTENDED | SSKE | RRE | P A1 | ¢ | | | | B22B | 10-87 |
| SET SYSTEM MASK | SSM | S | P A SP | SO | | B2 | | 80 | 10-87 |
| SHIFT AND ROUND DECIMAL | SRP | SS C | A | Dd DF | ST | B1 | | F0 | 8-13 |
| SHIFT LEFT DOUBLE | SLDA | RS C | SP | IF | R | | | 8F | 7-118 |
| SHIFT LEFT DOUBLE LOGICAL | SLDL | RS | SP | | R | | | 8D | 7-119 |
| SHIFT LEFT SINGLE | SLA | RS C | | IF | R | | | 8B | 7-119 |
| SHIFT LEFT SINGLE LOGICAL | SLL | RS | | | R | | | 89 | 7-120 |
| SHIFT RIGHT DOUBLE | SRDA | RS C | SP | | R | | | 8E | 7-120 |
| SHIFT RIGHT DOUBLE LOGICAL | SRDL | RS | SP | | R | | | 8C | 7-121 |
| SHIFT RIGHT SINGLE | SRA | RS C | | | R | | | 8A | 7-121 |
| SHIFT RIGHT SINGLE LOGICAL | SRL | RS | | | R | | | 88 | 7-121 |
| SIGNAL PROCESSOR | SIGP | RS C | P | $ | R | | | AE | 10-88 |
| SQUARE ROOT (extended BFP) | SQXBR | RRE BF | SP | Db Xi Xx | | | | B316 | 19-45 |
| SQUARE ROOT (extended HFP) | SQXR | RRE HX | SP | Da SQ | | | | B336 | 18-21 |
| SQUARE ROOT (long BFP) | SQDBR | RRE BF | | Db Xi Xx | | | | B315 | 19-45 |
| SQUARE ROOT (long BFP) | SQDB | RXE BF | A | Db Xi Xx | | B2 | | ED15 | 19-45 |
| SQUARE ROOT (long HFP) | SQDR | RRE QR | SP | Da SQ | | | | B244 | 18-21 |
| SQUARE ROOT (long HFP) | SQD | RXE HX | A | Da SQ | | B2 | | ED35 | 18-21 |
| SQUARE ROOT (short BFP) | SQEBR | RRE BF | | Db Xi Xx | | | | B314 | 19-45 |
| SQUARE ROOT (short BFP) | SQEB | RXE BF | A | Db Xi Xx | | B2 | | ED14 | 19-45 |
| SQUARE ROOT (short HFP) | SQER | RRE QR | SP | Da SQ | | | | B245 | 18-21 |
| SQUARE ROOT (short HFP) | SQE | RXE HX | A | Da SQ | | B2 | | ED34 | 18-21 |
| STORE | ST | RX | A | | ST | B2 | | 50 | 7-122 |
| STORE (long) | STD | RX | A SP | Da | ST | B2 | | 60 | 9-11 |
| STORE (short) | STE | RX | A SP | Da | ST | B2 | | 70 | 9-11 |
| STORE ACCESS MULTIPLE | STAM | RS | A SP | | ST | UB | | 9B | 7-122 |
| STORE CHARACTER | STC | RX | A | | ST | B2 | | 42 | 7-122 |
| STORE CHARACTERS UNDER MASK | STCM | RS | A | | ST | B2 | | BE | 7-122 |
| STORE CLOCK | STCK | S C | A | $ | ST | B2 | | B205 | 7-123 |

*Figure B-1 (Part 8 of 10). Instructions Arranged by Name*



LzLabs GmbH. Ex. 1008-918

| Name | Mne-monic | Characteristics | | | | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| STORE CLOCK COMPARATOR | STCKC | S | | P A SP | | | ST | $B_2$ | | B207 | 10-89 |
| STORE CLOCK EXTENDED | STCKE | S C EK | | A | | $ | ST | $B_2$ | | B278 | 7-124 |
| STORE CONTROL | STCTL | RS | | P A SP | | | ST | $B_2$ | | B6 | 10-89 |
| STORE CPU ADDRESS | STAP | S | | P A SP | | | ST | $B_2$ | | B212 | 10-90 |
| STORE CPU ID | STIDP | S | | P A SP | | | ST | $B_2$ | | B202 | 10-90 |
| STORE CPU TIMER | STPT | S | | P A SP | | | ST | $B_2$ | | B209 | 10-91 |
| STORE FACILITY LIST | STFL | S N3 | | P | | | | | | B2B1 | 10-91 |
| STORE FPC | STFPC | S BF | | A | | Db | ST | $B_2$ | | B29C | 19-45 |
| STORE HALFWORD | STH | RX | | A | | | ST | $B_2$ | | 40 | 7-126 |
| STORE MULTIPLE | STM | RS | | A | | | ST | $B_2$ | | 90 | 7-126 |
| STORE PREFIX | STPX | S | | P A SP | | | ST | $B_2$ | | B211 | 10-92 |
| STORE REVERSED | STRVH | RXE N3 | | A | | | ST | $B_2$ | | E33F | 7-126 |
| STORE REVERSED | STRV | RXE N3 | | A | | | ST | $B_2$ | | E33E | 7-126 |
| STORE SYSTEM INFORMATION | STSI | S C SN | | P A SP | | GM | R ST | $B_2$ | | B27D | 10-92 |
| STORE THEN AND SYSTEM MASK | STNSM | SI | | P A | | | ST | $B_1$ | | AC | 10-103 |
| STORE THEN OR SYSTEM MASK | STOSM | SI | | P A SP | | | ST | $B_1$ | | AD | 10-103 |
| STORE USING REAL ADDRESS | STURA | RRE | | P $A^1$ SP | | | SU | | | B246 | 10-104 |
| SUBTRACT | SR | RR C | | | | IF | R | | | 1B | 7-127 |
| SUBTRACT | S | RX C | | A | | IF | R | $B_2$ | | 5B | 7-127 |
| SUBTRACT (extended BFP) | SXBR | RRE C BF | | SP | | Db Xi Xo Xu Xx | | | | B34B | 19-45 |
| SUBTRACT (long BFP) | SDBR | RRE C BF | | | | Db Xi Xo Xu Xx | | | | B31B | 19-45 |
| SUBTRACT (long BFP) | SDB | RXE C BF | | A | | Db Xi Xo Xu Xx | | $B_2$ | | ED1B | 19-45 |
| SUBTRACT (short BFP) | SEBR | RRE C BF | | | | Db Xi Xo Xu Xx | | | | B30B | 19-45 |
| SUBTRACT (short BFP) | SEB | RXE C BF | | A | | Db Xi Xo Xu Xx | | $B_2$ | | ED0B | 19-45 |
| SUBTRACT DECIMAL | SP | SS C | | A | | Dd DF | ST | $B_1$ $B_2$ | | FB | 8-14 |
| SUBTRACT HALFWORD | SH | RX C | | A | | IF | R | $B_2$ | | 4B | 7-127 |
| SUBTRACT LOGICAL | SLR | RR C | | | | | R | | | 1F | 7-127 |
| SUBTRACT LOGICAL | SL | RX C | | A | | | R | $B_2$ | | 5F | 7-127 |
| SUBTRACT LOGICAL WITH BORROW | SLBR | RRE C N3 | | | | | R | | | B999 | 7-128 |
| SUBTRACT LOGICAL WITH BORROW | SLB | RXE C N3 | | A | | | R | $B_2$ | | E399 | 7-128 |
| SUBTRACT NORMALIZED (extended HFP) | SXR | RR C | | SP | | Da EU EO LS | | | | 37 | 18-23 |
| SUBTRACT NORMALIZED (long HFP) | SDR | RR C | | SP | | Da EU EO LS | | | | 2B | 18-23 |
| SUBTRACT NORMALIZED (long HFP) | SD | RX C | | A SP | | Da EU EO LS | | $B_2$ | | 6B | 18-23 |
| SUBTRACT NORMALIZED (short HFP) | SER | RR C | | SP | | Da EU EO LS | | | | 3B | 18-23 |
| SUBTRACT NORMALIZED (short HFP) | SE | RX C | | A SP | | Da EU EO LS | | $B_2$ | | 7B | 18-23 |
| SUBTRACT UNNORMALIZED (long HFP) | SWR | RR C | | SP | | Da EO LS | | | | 2F | 18-23 |
| SUBTRACT UNNORMALIZED (long HFP) | SW | RX C | | A SP | | Da EO LS | | $B_2$ | | 6F | 18-23 |
| SUBTRACT UNNORMALIZED (short HFP) | SUR | RR C | | SP | | Da EO LS | | | | 3F | 18-23 |
| SUBTRACT UNNORMALIZED (short HFP) | SU | RX C | | A SP | | Da EO LS | | $B_2$ | | 7F | 18-23 |
| SUPERVISOR CALL | SVC | RR | | | | ¢ | | | | 0A | 7-129 |
| TEST ACCESS | TAR | RRE C | | $A^1$ | | AS | | $U_1$ | | B24C | 10-104 |
| TEST ADDRESSING MODE | TAM | E C N3 | | | | | | | | 010B | 7-129 |
| TEST AND SET | TS | S C | | A | | $ | ST | $B_2$ | | 93 | 7-129 |
| TEST BLOCK | TB | RRE C | | P $A^1$ | | II $ G0 | R | | | B22C | 10-107 |
| TEST DATA CLASS (extended BFP) | TCXB | RXE C BF | | SP | | Db | | | | ED12 | 19-46 |

*Figure B-1 (Part 9 of 10). Instructions Arranged by Name*



LzLabs GmbH. Ex. 1008-919

| Name | Mne-monic | Characteristics | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|
| TEST DATA CLASS (long BFP) | TCDB | RXE C BF | | Db | | | ED11 | 19-46 |
| TEST DATA CLASS (short BFP) | TCEB | RXE C BF | | Db | | | ED10 | 19-46 |
| TEST DECIMAL | TP | RSL C E2 | A | | | $B_1$ | EBC0 | 8-14 |
| TEST PROTECTION | TPROT | SSE C | P $A^1$ | | | $B_1$ | E501 | 10-109 |
| TEST UNDER MASK | TM | SI C | A | | | $B_1$ | 91 | 7-130 |
| TEST UNDER MASK HIGH | TMH | RI C IR | | | | | A70 | 7-130 |
| TEST UNDER MASK LOW | TML | RI C IR | | | | | A71 | 7-130 |
| TRACE | TRACE | RS | P A SP | T ¢ | | $B_2$ | 99 | 10-111 |
| TRANSLATE | TR | SS | A | | ST | $B_1$ $B_2$ | DC | 7-131 |
| TRANSLATE AND TEST | TRT | SS C | A | GM | R | $B_1$ $B_2$ | DD | 7-132 |
| TRANSLATE EXTENDED | TRE | RRE C ET | A SP | | R ST | $R_1$ $R_2$ | B2A5 | 7-132 |
| TRANSLATE ONE TO ONE | TROO | RRE C E2 | A SP | GM | R ST | RM $R_2$ | B993 | 7-134 |
| TRANSLATE ONE TO TWO | TROT | RRE C E2 | A SP | GM | R ST | RM $R_2$ | B992 | 7-135 |
| TRANSLATE TWO TO ONE | TRTO | RRE C E2 | A SP | GM | R ST | RM $R_2$ | B991 | 7-135 |
| TRANSLATE TWO TO TWO | TRTT | RRE C E2 | A SP | GM | R ST | RM $R_2$ | B990 | 7-135 |
| TRAP | TRAP2 | E TR | A | SO T | B R ST | | 01FF | 10-112 |
| TRAP | TRAP4 | S TR | A | SO T | B R ST | | B2FF | 10-112 |
| UNPACK | UNPK | SS | A | | ST | $B_1$ $B_2$ | F3 | 7-139 |
| UNPACK ASCII | UNPKA | SS C E2 | A SP | | ST | $B_1$ $B_2$ | EA | 7-139 |
| UNPACK UNICODE | UNPKU | SS C E2 | A SP | | ST | $B_1$ $B_2$ | E2 | 7-140 |
| UPDATE TREE | UPT | E C | A SP | II GM | R ST | I4 | 0102 | 7-141 |
| ZERO AND ADD | ZAP | SS C | A | Dd DF | ST | $B_1$ $B_2$ | F8 | 8-14 |

*Figure B-1 (Part 10 of 10). Instructions Arranged by Name*



LzLabs GmbH. Ex. 1008-920

| Mne-monic | Name | Characteristics | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|
| | DIAGNOSE | DM | P DM | | | 83 | 10-19 |
| A | ADD | RX C | A | IF | R | $B_2$ 5A | 7-12 |
| AD | ADD NORMALIZED (long HFP) | RX C | A SP | Da EU EO LS | | $B_2$ 6A | 18-8 |
| ADB | ADD (long BFP) | RXE C BF | A | Db Xi Xo Xu Xx | | $B_2$ ED1A | 19-18 |
| ADBR | ADD (long BFP) | RRE C BF | | Db Xi Xo Xu Xx | | B31A | 19-18 |
| ADR | ADD NORMALIZED (long HFP) | RR C | SP | Da EU EO LS | | 2A | 18-8 |
| AE | ADD NORMALIZED (short HFP) | RX C | A SP | Da EU EO LS | | $B_2$ 7A | 18-8 |
| AEB | ADD (short BFP) | RXE C BF | A | Db Xi Xo Xu Xx | | $B_2$ ED0A | 19-18 |
| AEBR | ADD (short BFP) | RRE C BF | | Db Xi Xo Xu Xx | | B30A | 19-18 |
| AER | ADD NORMALIZED (short HFP) | RR C | SP | Da EU EO LS | | 3A | 18-8 |
| AH | ADD HALFWORD | RX C | A | IF | R | $B_2$ 4A | 7-12 |
| AHI | ADD HALFWORD IMMEDIATE | RI C IR | | IF | R | A7A | 7-12 |
| AL | ADD LOGICAL | RX C | A | | R | $B_2$ 5E | 7-13 |
| ALC | ADD LOGICAL WITH CARRY | RXE C N3 | A | | R | $B_2$ E398 | 7-13 |
| ALCR | ADD LOGICAL WITH CARRY | RRE C N3 | | | R | B998 | 7-13 |
| ALR | ADD LOGICAL | RR C | | | R | 1E | 7-13 |
| AP | ADD DECIMAL | SS C | A | Dd DF | ST | $B_1$ $B_2$ FA | 8-6 |
| AR | ADD | RR C | | IF | R | 1A | 7-12 |
| AU | ADD UNNORMALIZED (short HFP) | RX C | A SP | Da EO LS | | $B_2$ 7E | 18-10 |
| AUR | ADD UNNORMALIZED (short HFP) | RR C | SP | Da EO LS | | 3E | 18-10 |
| AW | ADD UNNORMALIZED (long HFP) | RX C | A SP | Da EO LS | | $B_2$ 6E | 18-10 |
| AWR | ADD UNNORMALIZED (long HFP) | RR C | SP | Da EO LS | | 2E | 18-10 |
| AXBR | ADD (extended BFP) | RRE C BF | SP | Db Xi Xo Xu Xx | | B34A | 19-18 |
| AXR | ADD NORMALIZED (extended HFP) | RR C | SP | Da EU EO LS | | 36 | 18-8 |
| BAKR | BRANCH AND STACK | RRE | A[1] | SF T | B ST | B240 | 10-10 |
| BAL | BRANCH AND LINK | RX | | | B R | 45 | 7-14 |
| BALR | BRANCH AND LINK | RR | | T | B R | 05 | 7-14 |
| BAS | BRANCH AND SAVE | RX | | | B R | 4D | 7-15 |
| BASR | BRANCH AND SAVE | RR | | T | B R | 0D | 7-15 |
| BASSM | BRANCH AND SAVE AND SET MODE | RR | | T | B R | 0C | 7-16 |
| BC | BRANCH ON CONDITION | RX | | | B | 47 | 7-17 |
| BCR | BRANCH ON CONDITION | RR | | ¢[1] | B | 07 | 7-17 |
| BCT | BRANCH ON COUNT | RX | | | B R | 46 | 7-18 |
| BCTR | BRANCH ON COUNT | RR | | | B R | 06 | 7-18 |
| BRAS | BRANCH RELATIVE AND SAVE | RI IR | | | B R | A75 | 7-19 |
| BRASL | BRANCH RELATIVE AND SAVE LONG | RIL N3 | | | B R | C05 | 7-19 |
| BRC | BRANCH RELATIVE ON CONDITION | RI IR | | | B | A74 | 7-20 |
| BRCL | BRANCH RELATIVE ON CONDITION LONG | RIL N3 | | | B | C04 | 7-20 |
| BRCT | BRANCH RELATIVE ON COUNT | RI IR | | | B R | A76 | 7-21 |
| BRXH | BRANCH RELATIVE ON INDEX HIGH | RSI IR | | | B R | 84 | 7-21 |
| BRXLE | BRANCH RELATIVE ON INDEX LOW OR EQ. | RSI IR | | | B R | 85 | 7-21 |
| BSA | BRANCH AND SET AUTHORITY | RRE BS | Q A[1] | SO T | B R | B25A | 10-7 |
| BSG | BRANCH IN SUBSPACE GROUP | RRE SG | A[1] | SO T | B R | $R_2$ B258 | 10-13 |
| BSM | BRANCH AND SET MODE | RR | | | B R | 0B | 7-16 |
| BXH | BRANCH ON INDEX HIGH | RS | | | B R | 86 | 7-18 |

*Figure B-2 (Part 1 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-921

| Mne-monic | Name | Characteristics | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|
| BXLE | BRANCH ON INDEX LOW OR EQUAL | RS | | | B R | 87 | 7-18 |
| C | COMPARE | RX C | A | | B$_2$ | 59 | 7-35 |
| CD | COMPARE (long HFP) | RX C | A SP | Da | B$_2$ | 69 | 18-10 |
| CDB | COMPARE (long BFP) | RXE C BF | A | Db Xi | B$_2$ | ED19 | 19-23 |
| CDBR | COMPARE (long BFP) | RRE C BF | | Db Xi | | B319 | 19-23 |
| CDFBR | CONVERT FROM FIXED (32 to long BFP) | RRE BF | | Db | | B395 | 19-26 |
| CDFR | CONVERT FROM FIXED (32 to long HFP) | RRE HX | | Da | | B3B5 | 18-11 |
| CDR | COMPARE (long HFP) | RR C | SP | Da | | 29 | 18-10 |
| CDS | COMPARE DOUBLE AND SWAP | RS C | A SP | $ | R ST | BB | 7-40 |
| CE | COMPARE (short HFP) | RX C | A SP | Da | B$_2$ | 79 | 18-10 |
| CEB | COMPARE (short BFP) | RXE C BF | A | Db Xi | B$_2$ | ED09 | 19-23 |
| CEBR | COMPARE (short BFP) | RRE C BF | | Db Xi | | B309 | 19-23 |
| CEFBR | CONVERT FROM FIXED (32 to short BFP) | RRE BF | | Db Xx | | B394 | 19-26 |
| CEFR | CONVERT FROM FIXED (32 to short HFP) | RRE HX | | Da | | B3B4 | 18-11 |
| CER | COMPARE (short HFP) | RR C | SP | Da | | 39 | 18-10 |
| CFC | COMPARE AND FORM CODEWORD | S C | A SP | II GM | R I1 | B21A | 7-36 |
| CFDBR | CONVERT TO FIXED (long BFP to 32) | RRF C BF | SP | Db Xi Xx | R | B399 | 19-26 |
| CFDR | CONVERT TO FIXED (long HFP to 32) | RRF C HX | SP | Da | R | B3B9 | 18-12 |
| CFEBR | CONVERT TO FIXED (short BFP to 32) | RRF C BF | SP | Db Xi Xx | R | B398 | 19-26 |
| CFER | CONVERT TO FIXED (short HFP to 32) | RRF C HX | SP | Da | R | B3B8 | 18-12 |
| CFXBR | CONVERT TO FIXED (ext. BFP to 32) | RRF C BF | SP | Db Xi Xx | R | B39A | 19-26 |
| CFXR | CONVERT TO FIXED (ext. HFP to 32) | RRF C HX | SP | Da | R | B3BA | 18-12 |
| CH | COMPARE HALFWORD | RX C | A | | B$_2$ | 49 | 7-42 |
| CHI | COMPARE HALFWORD IMMEDIATE | RI C IR | | | | A7E | 7-42 |
| CKSM | CHECKSUM | RRE C CK | A SP | | R R$_2$ | B241 | 7-22 |
| CL | COMPARE LOGICAL | RX C | A | | B$_2$ | 55 | 7-42 |
| CLC | COMPARE LOGICAL (character) | SS C | A | | B$_1$ B$_2$ | D5 | 7-42 |
| CLCL | COMPARE LOGICAL LONG | RR C | A SP | II | R R$_1$ R$_2$ | 0F | 7-43 |
| CLCLE | COMPARE LOGICAL LONG EXTENDED | RS C CM | A SP | | R R$_1$ R$_3$ | A9 | 7-45 |
| CLCLU | COMPARE LOGICAL LONG UNICODE | RSE C E2 | A SP | | R R$_1$ R$_2$ | EB8F | 7-47 |
| CLI | COMPARE LOGICAL (immediate) | SI C | A | | B$_1$ | 95 | 7-42 |
| CLM | COMPARE LOGICAL CHARS. UNDER MASK | RS C | A | | B$_2$ | BD | 7-43 |
| CLR | COMPARE LOGICAL | RR C | | | | 15 | 7-42 |
| CLST | COMPARE LOGICAL STRING | RRE C SR | A SP | G0 | R R$_1$ R$_2$ | B25D | 7-50 |
| CP | COMPARE DECIMAL | SS C | A | Dd | B$_1$ B$_2$ | F9 | 8-6 |
| CPYA | COPY ACCESS | RRE | | | U$_1$ U$_2$ | B24D | 7-72 |
| CR | COMPARE | RR C | | | | 19 | 7-35 |
| CS | COMPARE AND SWAP | RS C | A SP | $ | R ST B$_2$ | BA | 7-40 |
| CSP | COMPARE AND SWAP AND PURGE | RRE C | P A[1] SP | $ | R ST R$_2$ | B250 | 10-17 |
| CUSE | COMPARE UNTIL SUBSTRING EQUAL | RRE C | A SP | II GM | R$_1$ R$_2$ | B257 | 7-51 |
| CUTFU | CONVERT UTF-8 TO UNICODE | RRE C ET | A SP | | R ST R$_1$ R$_2$ | B2A7 | 7-70 |
| CUUTF | CONVERT UNICODE TO UTF-8 | RRE C ET | A SP | | R ST R$_1$ R$_2$ | B2A6 | 7-67 |
| CVB | CONVERT TO BINARY | RX | A | Dd IK | R B$_2$ | 4F | 7-66 |
| CVD | CONVERT TO DECIMAL | RX | A | | ST B$_2$ | 4E | 7-67 |
| CXBR | COMPARE (extended BFP) | RRE C BF | SP | Db Xi | | B349 | 19-23 |

*Figure B-2 (Part 2 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-922

| Mne-monic | Name | Characteristics | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|
| CXFBR | CONVERT FROM FIXED (32 to ext. BFP) | RRE BF | SP | Db | | B396 | 19-26 |
| CXFR | CONVERT FROM FIXED (32 to ext. HFP) | RRE HX | SP | Da | | B3B6 | 18-11 |
| CXR | COMPARE (extended HFP) | RRE C HX | SP | Da | | B369 | 18-10 |
| D | DIVIDE | RX | A SP | IK | R B2 | 5D | 7-73 |
| DD | DIVIDE (long HFP) | RX | A SP | Da EU EO FK | B2 | 6D | 18-12 |
| DDB | DIVIDE (long BFP) | RXE BF | A | Db Xi Xz Xo Xu Xx | B2 | ED1D | 19-29 |
| DDBR | DIVIDE (long BFP) | RRE BF | | Db Xi Xz Xo Xu Xx | | B31D | 19-29 |
| DDR | DIVIDE (long HFP) | RR | SP | Da EU EO FK | | 2D | 18-12 |
| DE | DIVIDE (short HFP) | RX | A SP | Da EU EO FK | B2 | 7D | 18-12 |
| DEB | DIVIDE (short BFP) | RXE BF | A | Db Xi Xz Xo Xu Xx | B2 | ED0D | 19-29 |
| DEBR | DIVIDE (short BFP) | RRE BF | | Db Xi Xz Xo Xu Xx | | B30D | 19-29 |
| DER | DIVIDE (short HFP) | RR | SP | Da EU EO FK | | 3D | 18-12 |
| DIDBR | DIVIDE TO INTEGER (long BFP) | RRF C BF | SP | Db Xi Xu Xx | | B35B | 19-30 |
| DIEBR | DIVIDE TO INTEGER (short BFP) | RRF C BF | SP | Db Xi Xu Xx | | B353 | 19-30 |
| DL | DIVIDE LOGICAL | RXE N3 | A SP | IK | R B2 | E397 | 7-73 |
| DLR | DIVIDE LOGICAL | RRE N3 | SP | IK | R | B997 | 7-73 |
| DP | DIVIDE DECIMAL | SS | A SP | Dd DK | ST B1 B2 | FD | 8-7 |
| DR | DIVIDE | RR | SP | IK | R | 1D | 7-73 |
| DXBR | DIVIDE (extended BFP) | RRE BF | SP | Db Xi Xz Xo Xu Xx | | B34D | 19-29 |
| DXR | DIVIDE (extended HFP) | RRE | SP | Da EU EO FK | | B22D | 18-12 |
| EAR | EXTRACT ACCESS | RRE | | | R U2 | B24F | 7-75 |
| ED | EDIT | SS C | A | Dd | ST B1 B2 | DE | 8-7 |
| EDMK | EDIT AND MARK | SS C | A | Dd G1 | R ST B1 B2 | DF | 8-12 |
| EFPC | EXTRACT FPC | RRE BF | | Db | | B38C | 19-34 |
| EPAR | EXTRACT PRIMARY ASN | RRE | Q | SO | R | B226 | 10-19 |
| EPSW | EXTRACT PSW | RRE N3 | | | R | B98D | 7-76 |
| EREG | EXTRACT STACKED REGISTERS | RRE | A[1] | SE | R U1 U2 | B249 | 10-20 |
| ESAR | EXTRACT SECONDARY ASN | RRE | Q | SO | R | B227 | 10-20 |
| ESTA | EXTRACT STACKED STATE | RRE C | A[1] SP | SE | R | B24A | 10-21 |
| EX | EXECUTE | RX | AI SP | EX | | 44 | 7-74 |
| FIDBR | LOAD FP INTEGER (long BFP) | RRF BF | SP | Db Xi Xx | | B35F | 19-36 |
| FIDR | LOAD FP INTEGER (long HFP) | RRE HX | | Da | | B37F | 18-15 |
| FIEBR | LOAD FP INTEGER (short BFP) | RRF BF | SP | Db Xi Xx | | B357 | 19-36 |
| FIER | LOAD FP INTEGER (short HFP) | RRE HX | | Da | | B377 | 18-15 |
| FIXBR | LOAD FP INTEGER (extended BFP) | RRF BF | SP | Db Xi Xx | | B347 | 19-36 |
| FIXR | LOAD FP INTEGER (extended HFP) | RRE HX | SP | Da | | B367 | 18-15 |
| HDR | HALVE (long HFP) | RR | SP | Da EU | | 24 | 18-13 |
| HER | HALVE (short HFP) | RR | SP | Da EU | | 34 | 18-13 |
| IAC | INSERT ADDRESS SPACE CONTROL | RRE C | Q | SO | R | B224 | 10-23 |
| IC | INSERT CHARACTER | RX | A | | R B2 | 43 | 7-76 |
| ICM | INSERT CHARACTERS UNDER MASK | RS C | A | | R B2 | BF | 7-76 |
| IPK | INSERT PSW KEY | S | Q | G2 | R | B20B | 10-24 |
| IPM | INSERT PROGRAM MASK | RRE | | | R | B222 | 7-77 |
| IPTE | INVALIDATE PAGE TABLE ENTRY | RRE | P A[1] | $ | | B221 | 10-26 |
| ISKE | INSERT STORAGE KEY EXTENDED | RRE | P A[1] | | | B229 | 10-25 |

*Figure B-2 (Part 3 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-923

| Mne-monic | Name | Characteristics | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|
| IVSK | INSERT VIRTUAL STORAGE KEY | RRE | Q A[1] | SO | R | $R_2$ | B223 | 10-25 |
| KDB | COMPARE AND SIGNAL (long BFP) | RXE C BF | A | Db Xi | | $B_2$ | ED18 | 19-24 |
| KDBR | COMPARE AND SIGNAL (long BFP) | RRE C BF | | Db Xi | | | B318 | 19-24 |
| KEB | COMPARE AND SIGNAL (short BFP) | RXE C BF | A | Db Xi | | $B_2$ | ED08 | 19-24 |
| KEBR | COMPARE AND SIGNAL (short BFP) | RRE C BF | | Db Xi | | | B308 | 19-24 |
| KIMD | COMPUTE INTERMEDIATE MESSAGE DIGEST | RRE C MS | A SP | GM I1 | ST | $R_2$ | B93E | 7-55 |
| KLMD | COMPUTE LAST MESSAGE DIGEST | RRE C MS | A SP | GM I1 | ST | $R_2$ | B93F | 7-55 |
| KM | CIPHER MESSAGE | RRE C MS | A SP | GM I1 | ST | $R_1$ $R_2$ | B92E | 7-26 |
| KMAC | COMPUTE MESSAGE AUTHENTICATION CODE | RRE C MS | A SP | GM I1 | ST | $R_2$ | B91E | 7-61 |
| KMC | CIPHER MESSAGE WITH CHAINING | RRE C MS | A SP | GM I1 | ST | $R_1$ $R_2$ | B92F | 7-26 |
| KXBR | COMPARE AND SIGNAL (extended BFP) | RRE C BF | SP | Db Xi | | | B348 | 19-24 |
| L | LOAD | RX | A | | R | $B_2$ | 58 | 7-77 |
| LA | LOAD ADDRESS | RX | | | R | | 41 | 7-78 |
| LAE | LOAD ADDRESS EXTENDED | RX | | | R | $U_1$ BP | 51 | 7-78 |
| LAM | LOAD ACCESS MULTIPLE | RS | A SP | | | UB | 9A | 7-77 |
| LARL | LOAD ADDRESS RELATIVE LONG | RIL N3 | | | R | | C00 | 7-79 |
| LASP | LOAD ADDRESS SPACE PARAMETERS | SSE C | P A[1] SP | AS | | $B_1$ | E500 | 10-28 |
| LCDBR | LOAD COMPLEMENT (long BFP) | RRE C BF | | Db | | | B313 | 19-35 |
| LCDR | LOAD COMPLEMENT (long HFP) | RR C | SP | Da | | | 23 | 18-15 |
| LCEBR | LOAD COMPLEMENT (short BFP) | RRE C BF | | Db | | | B303 | 19-35 |
| LCER | LOAD COMPLEMENT (short HFP) | RR C | SP | Da | | | 33 | 18-15 |
| LCR | LOAD COMPLEMENT | RR C | | IF | R | | 13 | 7-79 |
| LCTL | LOAD CONTROL | RS | P A SP | | | $B_2$ | B7 | 10-36 |
| LCXBR | LOAD COMPLEMENT (extended BFP) | RRE C BF | SP | Db | | | B343 | 19-35 |
| LCXR | LOAD COMPLEMENT (extended HFP) | RRE C HX | SP | Da | | | B363 | 18-15 |
| LD | LOAD (long) | RX | A SP | Da | | $B_2$ | 68 | 9-10 |
| LDE | LOAD LENGTHENED (short to long HFP) | RXE HX | A | Da | | $B_2$ | ED24 | 18-16 |
| LDEB | LOAD LENGTHENED (short to long BFP) | RXE BF | A | Db Xi | | $B_2$ | ED04 | 19-38 |
| LDEBR | LOAD LENGTHENED (short to long BFP) | RRE BF | | Db Xi | | | B304 | 19-38 |
| LDER | LOAD LENGTHENED (short to long HFP) | RRE HX | | Da | | | B324 | 18-16 |
| LDR | LOAD (long) | RR | SP | Da | | | 28 | 9-10 |
| LDXBR | LOAD ROUNDED (extended to long BFP) | RRE BF | SP | Db Xi Xo Xu Xx | | | B345 | 19-39 |
| LDXR | LOAD ROUNDED (extended to long HFP) | RR | SP | Da EO | | | 25 | 18-18 |
| LE | LOAD (short) | RX | A SP | Da | | $B_2$ | 78 | 9-10 |
| LEDBR | LOAD ROUNDED (long to short BFP) | RRE BF | | Db Xi Xo Xu Xx | | | B344 | 19-39 |
| LEDR | LOAD ROUNDED (long to short HFP) | RR | SP | Da EO | | | 35 | 18-18 |
| LER | LOAD (short) | RR | SP | Da | | | 38 | 9-10 |
| LEXBR | LOAD ROUNDED (extended to short BFP) | RRE BF | SP | Db Xi Xo Xu Xx | | | B346 | 19-39 |
| LEXR | LOAD ROUNDED (extended to short HFP) | RRE HX | SP | Da EO | | | B366 | 18-18 |
| LFPC | LOAD FPC | S BF | A SP | Db | | $B_2$ | B29D | 19-37 |
| LH | LOAD HALFWORD | RX | A | | R | $B_2$ | 48 | 7-80 |
| LHI | LOAD HALFWORD IMMEDIATE | RI IR | | | R | | A78 | 7-80 |
| LM | LOAD MULTIPLE | RS | A | | R | $B_2$ | 98 | 7-80 |
| LNDBR | LOAD NEGATIVE (long BFP) | RRE C BF | | Db | | | B311 | 19-38 |
| LNDR | LOAD NEGATIVE (long HFP) | RR C | SP | Da | | | 21 | 18-16 |

*Figure B-2 (Part 4 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-924

| Mne-monic | Name | Characteristics | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|
| LNEBR | LOAD NEGATIVE (short BFP) | RRE C BF | | Db | | | B301 | 19-38 |
| LNER | LOAD NEGATIVE (short HFP) | RR C | SP | Da | | | 31 | 18-16 |
| LNR | LOAD NEGATIVE | RR C | | | R | | 11 | 7-80 |
| LNXBR | LOAD NEGATIVE (extended BFP) | RRE C BF | SP | Db | | | B341 | 19-38 |
| LNXR | LOAD NEGATIVE (extended HFP) | RRE C HX | SP | Da | | | B361 | 18-16 |
| LPDBR | LOAD POSITIVE (long BFP) | RRE C BF | | Db | | | B310 | 19-39 |
| LPDR | LOAD POSITIVE (long HFP) | RR C | SP | Da | | | 20 | 18-17 |
| LPEBR | LOAD POSITIVE (short BFP) | RRE C BF | | Db | | | B300 | 19-39 |
| LPER | LOAD POSITIVE (short HFP) | RR C | SP | Da | | | 30 | 18-17 |
| LPR | LOAD POSITIVE | RR C | | IF | R | | 10 | 7-81 |
| LPSW | LOAD PSW | S L | P A SP | ¢ | | B2 | 82 | 10-37 |
| LPXBR | LOAD POSITIVE (extended BFP) | RRE C BF | SP | Db | | | B340 | 19-39 |
| LPXR | LOAD POSITIVE (extended HFP) | RRE C HX | SP | Da | | | B360 | 18-17 |
| LR | LOAD | RR | | | R | | 18 | 7-77 |
| LRA | LOAD REAL ADDRESS | RX C | P A[1] | AT | R | BP | B1 | 10-38 |
| LRDR | LOAD ROUNDED (extended to long HFP) | RR | SP | Da EO | | | 25 | 18-18 |
| LRER | LOAD ROUNDED (long to short HFP) | RR | SP | Da EO | | | 35 | 18-18 |
| LRV | LOAD REVERSED | RXE N3 | A | | R | B2 | E31E | 7-81 |
| LRVH | LOAD REVERSED | RXE N3 | A | | R | B2 | E31F | 7-81 |
| LRVR | LOAD REVERSED | RRE N3 | | | R | | B91F | 7-81 |
| LTDBR | LOAD AND TEST (long BFP) | RRE C BF | | Db Xi | | | B312 | 19-35 |
| LTDR | LOAD AND TEST (long HFP) | RR C | SP | Da | | | 22 | 18-14 |
| LTEBR | LOAD AND TEST (short BFP) | RRE C BF | | Db Xi | | | B302 | 19-35 |
| LTER | LOAD AND TEST (short HFP) | RR C | SP | Da | | | 32 | 18-14 |
| LTR | LOAD AND TEST | RR C | | | R | | 12 | 7-79 |
| LTXBR | LOAD AND TEST (extended BFP) | RRE C BF | SP | Db Xi | | | B342 | 19-35 |
| LTXR | LOAD AND TEST (extended HFP) | RRE C HX | SP | Da | | | B362 | 18-14 |
| LURA | LOAD USING REAL ADDRESS | RRE | P A[1] SP | | R | | B24B | 10-40 |
| LXD | LOAD LENGTHENED (long to ext. HFP) | RXE HX | A SP | Da | | B2 | ED25 | 18-16 |
| LXDB | LOAD LENGTHENED (long to ext. BFP) | RXE BF | A SP | Db Xi | | B2 | ED05 | 19-38 |
| LXDBR | LOAD LENGTHENED (long to ext. BFP) | RRE BF | SP | Db Xi | | | B305 | 19-38 |
| LXDR | LOAD LENGTHENED (long to ext. HFP) | RRE HX | SP | Da | | | B325 | 18-16 |
| LXE | LOAD LENGTHENED (short to ext. HFP) | RXE HX | A SP | Da | | B2 | ED26 | 18-16 |
| LXEB | LOAD LENGTHENED (short to ext. BFP) | RXE BF | A SP | Db Xi | | B2 | ED06 | 19-38 |
| LXEBR | LOAD LENGTHENED (short to ext. BFP) | RRE BF | SP | Db Xi | | | B306 | 19-38 |
| LXER | LOAD LENGTHENED (short to ext. HFP) | RRE HX | SP | Da | | | B326 | 18-16 |
| LXR | LOAD (extended) | RRE FX | SP | Da | | | B365 | 9-10 |
| LZDR | LOAD ZERO (long) | RRE FX | | Da | | | B375 | 9-11 |
| LZER | LOAD ZERO (short) | RRE FX | | Da | | | B374 | 9-11 |
| LZXR | LOAD ZERO (extended) | RRE FX | SP | Da | | | B376 | 9-11 |
| M | MULTIPLY | RX | A SP | | R | B2 | 5C | 7-98 |
| MAD | MULTIPLY AND ADD (long HFP) | RXF HM | A | Da EU EO | | B2 | ED3E | 18-20 |
| MADB | MULTIPLY AND ADD (long BFP) | RXF BF | A | Db Xi Xo Xu Xx | | B2 | ED1E | 19-42 |
| MADBR | MULTIPLY AND ADD (long BFP) | RRF BF | | Db Xi Xo Xu Xx | | | B31E | 19-42 |
| MADR | MULTIPLY AND ADD (long HFP) | RRF HM | | Da EU EO | | | B33E | 18-20 |

*Figure B-2 (Part 5 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-925

| Mne-monic | Name | Characteristics | | | | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MAE | MULTIPLY AND ADD (short HFP) | RXF | HM | A | | Da EU EO | | | $B_2$ | ED2E | 18-20 |
| MAEB | MULTIPLY AND ADD (short BFP) | RXF | BF | A | | Db Xi | Xo Xu Xx | | $B_2$ | ED0E | 19-42 |
| MAEBR | MULTIPLY AND ADD (short BFP) | RRF | BF | | | Db Xi | Xo Xu Xx | | | B30E | 19-42 |
| MAER | MULTIPLY AND ADD (short HFP) | RRF | HM | | | Da EU EO | | | | B32E | 18-20 |
| MC | MONITOR CALL | SI | | | SP | | MO | | | AF | 7-82 |
| MD | MULTIPLY (long HFP) | RX | | A | SP | Da EU EO | | | $B_2$ | 6C | 18-18 |
| MDB | MULTIPLY (long BFP) | RXE | BF | A | | Db Xi | Xo Xu Xx | | $B_2$ | ED1C | 19-40 |
| MDBR | MULTIPLY (long BFP) | RRE | BF | | | Db Xi | Xo Xu Xx | | | B31C | 19-40 |
| MDE | MULTIPLY (short to long HFP) | RX | | A | SP | Da EU EO | | | $B_2$ | 7C | 18-18 |
| MDEB | MULTIPLY (short to long BFP) | RXE | BF | A | | Db Xi | | | $B_2$ | ED0C | 19-40 |
| MDEBR | MULTIPLY (short to long BFP) | RRE | BF | | | Db Xi | | | | B30C | 19-40 |
| MDER | MULTIPLY (short to long HFP) | RR | | | SP | Da EU EO | | | | 3C | 18-18 |
| MDR | MULTIPLY (long HFP) | RR | | | SP | Da EU EO | | | | 2C | 18-18 |
| ME | MULTIPLY (short to long HFP) | RX | | A | SP | Da EU EO | | | $B_2$ | 7C | 18-18 |
| MEE | MULTIPLY (short HFP) | RXE | HX | A | | Da EU EO | | | $B_2$ | ED37 | 18-18 |
| MEEB | MULTIPLY (short BFP) | RXE | BF | A | | Db Xi | Xo Xu Xx | | $B_2$ | ED17 | 19-40 |
| MEEBR | MULTIPLY (short BFP) | RRE | BF | | | Db Xi | Xo Xu Xx | | | B317 | 19-40 |
| MEER | MULTIPLY (short HFP) | RRE | HX | | | Da EU EO | | | | B337 | 18-18 |
| MER | MULTIPLY (short to long HFP) | RR | | | SP | Da EU EO | | | | 3C | 18-18 |
| MH | MULTIPLY HALFWORD | RX | | A | | | | R | $B_2$ | 4C | 7-98 |
| MHI | MULTIPLY HALFWORD IMMEDIATE | RI | IR | | | | | R | | A7C | 7-98 |
| ML | MULTIPLY LOGICAL | RXE | N3 | A | SP | | | R | $B_2$ | E396 | 7-99 |
| MLR | MULTIPLY LOGICAL | RRE | N3 | | SP | | | R | | B996 | 7-99 |
| MP | MULTIPLY DECIMAL | SS | | A | SP | Dd | | ST | $B_1$ $B_2$ | FC | 8-12 |
| MR | MULTIPLY | RR | | | SP | | | R | | 1C | 7-98 |
| MS | MULTIPLY SINGLE | RX | IR | A | | | | R | $B_2$ | 71 | 7-99 |
| MSD | MULTIPLY AND SUBTRACT (long HFP) | RXF | HM | A | | Da EU EO | | | $B_2$ | ED3F | 18-20 |
| MSDB | MULTIPLY AND SUBTRACT (long BFP) | RXF | BF | A | | Db Xi | Xo Xu Xx | | $B_2$ | ED1F | 19-42 |
| MSDBR | MULTIPLY AND SUBTRACT (long BFP) | RRF | BF | | | Db Xi | Xo Xu Xx | | | B31F | 19-42 |
| MSDR | MULTIPLY AND SUBTRACT (long HFP) | RRF | HM | | | Da EU EO | | | | B33F | 18-20 |
| MSE | MULTIPLY AND SUBTRACT (short HFP) | RXF | HM | A | | Da EU EO | | | $B_2$ | ED2F | 18-20 |
| MSEB | MULTIPLY AND SUBTRACT (short BFP) | RXF | BF | A | | Db Xi | Xo Xu Xx | | $B_2$ | ED0F | 19-42 |
| MSEBR | MULTIPLY AND SUBTRACT (short BFP) | RRF | BF | | | Db Xi | Xo Xu Xx | | | B30F | 19-42 |
| MSER | MULTIPLY AND SUBTRACT (short HFP) | RRF | HM | | | Da EU EO | | | | B32F | 18-20 |
| MSR | MULTIPLY SINGLE | RRE | IR | | | | | R | | B252 | 7-99 |
| MSTA | MODIFY STACKED STATE | RRE | | $A^1$ | SP | SE | | ST | | B247 | 10-40 |
| MVC | MOVE (character) | SS | | A | | | | ST | $B_1$ $B_2$ | D2 | 7-83 |
| MVCDK | MOVE WITH DESTINATION KEY | SSE | | Q A | | | GM | ST | $B_1$ $B_2$ | E50F | 10-47 |
| MVCIN | MOVE INVERSE | SS | | A | | | | ST | $B_1$ $B_2$ | E8 | 7-83 |
| MVCK | MOVE WITH KEY | SS | C | Q A | | | | ST | $B_1$ $B_2$ | D9 | 10-47 |
| MVCL | MOVE LONG | RR | C | A | SP | II | | R ST | $R_1$ $R_2$ | 0E | 7-83 |
| MVCLE | MOVE LONG EXTENDED | RS | C CM | A | SP | | | R ST | $R_1$ $R_3$ | A8 | 7-87 |
| MVCLU | MOVE LONG UNICODE | RSE | C E2 | A | SP | | | R ST | $R_1$ $R_2$ | EB8E | 7-90 |
| MVCP | MOVE TO PRIMARY | SS | C | Q A | | SO ¢ | | ST | | DA | 10-45 |
| MVCS | MOVE TO SECONDARY | SS | C | Q A | | SO ¢ | | ST | | DB | 10-45 |

*Figure B-2 (Part 6 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-926

| Mne-monic | Name | Characteristics | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|
| MVCSK | MOVE WITH SOURCE KEY | SSE | Q A | GM | ST B1 B2 | E50E | 10-48 |
| MVI | MOVE (immediate) | SI | A | | ST B1 | 92 | 7-83 |
| MVN | MOVE NUMERICS | SS | A | | ST B1 B2 | D1 | 7-93 |
| MVO | MOVE WITH OFFSET | SS | A | | ST B1 B2 | F1 | 7-97 |
| MVPG | MOVE PAGE (facility 1) | RRE C M1 | A1 SP | G0 | ST R1 R2 | B254 | 7-93 |
| MVPG | MOVE PAGE (facility 2) | RRE C M2 | Q A1 SP | G0 | ST R1 R2 | B254 | 7-93 |
| MVST | MOVE STRING | RRE C SR | A SP | G0 | R ST R1 R2 | B255 | 7-95 |
| MVZ | MOVE ZONES | SS | A | | ST B1 B2 | D3 | 7-97 |
| MXBR | MULTIPLY (extended BFP) | RRE BF | SP | Db Xi Xo Xu Xx | | B34C | 19-40 |
| MXD | MULTIPLY (long to extended HFP) | RX | A SP | Da EU EO | B2 | 67 | 18-18 |
| MXDB | MULTIPLY (long to extended BFP) | RXE BF | A SP | Db Xi | B2 | ED07 | 19-40 |
| MXDBR | MULTIPLY (long to extended BFP) | RRE BF | SP | Db Xi | | B307 | 19-40 |
| MXDR | MULTIPLY (long to extended HFP) | RR | SP | Da EU EO | | 27 | 18-18 |
| MXR | MULTIPLY (extended HFP) | RR | SP | Da EU EO | | 26 | 18-18 |
| N | AND | RX C | A | | R B2 | 54 | 7-13 |
| NC | AND (character) | SS C | A | | ST B1 B2 | D4 | 7-13 |
| NI | AND (immediate) | SI C | A | | ST B1 | 94 | 7-13 |
| NR | AND | RR C | | | R | 14 | 7-13 |
| O | OR | RX C | A | | R B2 | 56 | 7-100 |
| OC | OR (character) | SS C | A | | ST B1 B2 | D6 | 7-100 |
| OI | OR (immediate) | SI C | A | | ST B1 | 96 | 7-100 |
| OR | OR | RR C | | | R | 16 | 7-100 |
| PACK | PACK | SS | A | | ST B1 B2 | F2 | 7-100 |
| PALB | PURGE ALB | RRE | P | $ | | B248 | 10-76 |
| PC | PROGRAM CALL | S | Q A1 | Z1 T ¢ GM | B R ST | B218 | 10-52 |
| PCF | PROGRAM CALL FAST | S PC | A1 | Z5 ¢ G4 | B R ST | B218 | 10-63 |
| PGIN | PAGE IN | RRE C ES | P A1 | ¢ | | B22E | 10-50 |
| PGOUT | PAGE OUT | RRE C ES | P A1 | ¢ | | B22F | 10-51 |
| PKA | PACK ASCII | SS E2 | A SP | | ST B1 B2 | E9 | 7-101 |
| PKU | PACK UNICODE | SS E2 | A SP | | ST B1 B2 | E1 | 7-102 |
| PLO | PERFORM LOCKED OPERATION | SS C PL | A SP | $ GM | R ST FC | EE | 7-103 |
| PR | PROGRAM RETURN | E U | A1 SP | Z4 T ¢2 | B R ST | 0101 | 10-67 |
| PT | PROGRAM TRANSFER | RRE | Q A1 SP | Z2 T ¢ | B | B228 | 10-70 |
| PTLB | PURGE TLB | S | P | $ | | B20D | 10-76 |
| RLL | ROTATE LEFT SINGLE LOGICAL | RSE N3 | | | R | EB1D | 7-116 |
| RP | RESUME PROGRAM | S L RP | Q A SP | SW T | B R B2 | B277 | 10-77 |
| RRBE | RESET REFERENCE BIT EXTENDED | RRE C | P A1 | | | B22A | 10-76 |
| S | SUBTRACT | RX C | A | IF | R B2 | 5B | 7-127 |
| SAC | SET ADDRESS SPACE CONTROL | S | Q SP | SW ¢ | | B219 | 10-79 |
| SACF | SET ADDRESS SPACE CONTROL FAST | S SA | Q SP | SW | | B279 | 10-79 |
| SAM24 | SET ADDRESSING MODE | E N3 | SP | T | | 010C | 7-117 |
| SAM31 | SET ADDRESSING MODE | E N3 | SP | T | | 010D | 7-117 |
| SAR | SET ACCESS | RRE | | | U1 | B24E | 7-117 |
| SCK | SET CLOCK | S C | P A SP | | B2 | B204 | 10-81 |
| SCKC | SET CLOCK COMPARATOR | S | P A SP | | B2 | B206 | 10-82 |

*Figure B-2 (Part 7 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-927

| Mne-monic | Name | Characteristics | | | | | | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SCKPF | SET CLOCK PROGRAMMABLE FIELD | E EK | P SP | G0 | | | | | | | | 0107 | 10-82 |
| SD | SUBTRACT NORMALIZED (long HFP) | RX C | A SP | Da EU EO LS | | | | | | | $B_2$ | 6B | 18-23 |
| SDB | SUBTRACT (long BFP) | RXE C BF | A | Db Xi Xo Xu Xx | | | | | | | $B_2$ | ED1B | 19-45 |
| SDBR | SUBTRACT (long BFP) | RRE C BF | | Db Xi Xo Xu Xx | | | | | | | | B31B | 19-45 |
| SDR | SUBTRACT NORMALIZED (long HFP) | RR C | SP | Da EU EO LS | | | | | | | | 2B | 18-23 |
| SE | SUBTRACT NORMALIZED (short HFP) | RX C | A SP | Da EU EO LS | | | | | | | $B_2$ | 7B | 18-23 |
| SEB | SUBTRACT (short BFP) | RXE C BF | A | Db Xi Xo Xu Xx | | | | | | | $B_2$ | ED0B | 19-45 |
| SEBR | SUBTRACT (short BFP) | RRE C BF | | Db Xi Xo Xu Xx | | | | | | | | B30B | 19-45 |
| SER | SUBTRACT NORMALIZED (short HFP) | RR C | SP | Da EU EO LS | | | | | | | | 3B | 18-23 |
| SFPC | SET FPC | RRE BF | SP | Db | | | | | | | | B384 | 19-44 |
| SH | SUBTRACT HALFWORD | RX C | A | IF | | | | | R | | $B_2$ | 4B | 7-127 |
| SIGP | SIGNAL PROCESSOR | RS C | P | $ | | | | | R | | | AE | 10-88 |
| SL | SUBTRACT LOGICAL | RX C | A | | | | | | R | | $B_2$ | 5F | 7-127 |
| SLA | SHIFT LEFT SINGLE | RS C | | IF | | | | | R | | | 8B | 7-119 |
| SLB | SUBTRACT LOGICAL WITH BORROW | RXE C N3 | A | | | | | | R | | $B_2$ | E399 | 7-128 |
| SLBR | SUBTRACT LOGICAL WITH BORROW | RRE C N3 | | | | | | | R | | | B999 | 7-128 |
| SLDA | SHIFT LEFT DOUBLE | RS C | SP | IF | | | | | R | | | 8F | 7-118 |
| SLDL | SHIFT LEFT DOUBLE LOGICAL | RS | SP | | | | | | R | | | 8D | 7-119 |
| SLL | SHIFT LEFT SINGLE LOGICAL | RS | | | | | | | R | | | 89 | 7-120 |
| SLR | SUBTRACT LOGICAL | RR C | | | | | | | R | | | 1F | 7-127 |
| SP | SUBTRACT DECIMAL | SS C | A | Dd DF | | | | | ST | $B_1$ | $B_2$ | FB | 8-14 |
| SPKA | SET PSW KEY FROM ADDRESS | S | Q | | | | | | | | | B20A | 10-83 |
| SPM | SET PROGRAM MASK | RR L | | | | | | | | | | 04 | 7-118 |
| SPT | SET CPU TIMER | S | P A SP | | | | | | | | $B_2$ | B208 | 10-82 |
| SPX | SET PREFIX | S | P A SP | $ | | | | | | | $B_2$ | B210 | 10-83 |
| SQD | SQUARE ROOT (long HFP) | RXE HX | A | Da SQ | | | | | | | $B_2$ | ED35 | 18-21 |
| SQDB | SQUARE ROOT (long BFP) | RXE BF | A | Db Xi Xx | | | | | | | $B_2$ | ED15 | 19-45 |
| SQDBR | SQUARE ROOT (long BFP) | RRE BF | | Db Xi Xx | | | | | | | | B315 | 19-45 |
| SQDR | SQUARE ROOT (long HFP) | RRE QR | SP | Da SQ | | | | | | | | B244 | 18-21 |
| SQE | SQUARE ROOT (short HFP) | RXE HX | A | Da SQ | | | | | | | $B_2$ | ED34 | 18-21 |
| SQEB | SQUARE ROOT (short BFP) | RXE BF | A | Db Xi Xx | | | | | | | $B_2$ | ED14 | 19-45 |
| SQEBR | SQUARE ROOT (short BFP) | RRE BF | | Db Xi Xx | | | | | | | | B314 | 19-45 |
| SQER | SQUARE ROOT (short HFP) | RRE QR | SP | Da SQ | | | | | | | | B245 | 18-21 |
| SQXBR | SQUARE ROOT (extended BFP) | RRE BF | SP | Db Xi Xx | | | | | | | | B316 | 19-45 |
| SQXR | SQUARE ROOT (extended HFP) | RRE HX | SP | Da SQ | | | | | | | | B336 | 18-21 |
| SR | SUBTRACT | RR C | | IF | | | | | R | | | 1B | 7-127 |
| SRA | SHIFT RIGHT SINGLE | RS C | | | | | | | R | | | 8A | 7-121 |
| SRDA | SHIFT RIGHT DOUBLE | RS C | SP | | | | | | R | | | 8E | 7-120 |
| SRDL | SHIFT RIGHT DOUBLE LOGICAL | RS | SP | | | | | | R | | | 8C | 7-121 |
| SRL | SHIFT RIGHT SINGLE LOGICAL | RS | | | | | | | R | | | 88 | 7-121 |
| SRNM | SET ROUNDING MODE | S BF | | Db | | | | | | | | B299 | 19-44 |
| SRP | SHIFT AND ROUND DECIMAL | SS C | A | Dd DF | | | | | ST | $B_1$ | | F0 | 8-13 |
| SRST | SEARCH STRING | RRE C SR | A SP | G0 | | | | | R | | $R_2$ | B25E | 7-116 |
| SSAR | SET SECONDARY ASN | RRE | A[1] | Z[3] T ¢ | | | | | | | | B225 | 10-84 |
| SSKE | SET STORAGE KEY EXTENDED | RRE | P A[1] | ¢ | | | | | | | | B22B | 10-87 |

*Figure B-2 (Part 8 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-928

| Mne-monic | Name | Characteristics | | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|---|
| SSM | SET SYSTEM MASK | S | P A SP | SO | | | B2 | 80 | 10-87 |
| ST | STORE | RX | A | | | ST | B2 | 50 | 7-122 |
| STAM | STORE ACCESS MULTIPLE | RS | A SP | | | ST | UB | 9B | 7-122 |
| STAP | STORE CPU ADDRESS | S | P A SP | | | ST | B2 | B212 | 10-90 |
| STC | STORE CHARACTER | RX | A | | | ST | B2 | 42 | 7-122 |
| STCK | STORE CLOCK | S C | A | $ | | ST | B2 | B205 | 7-123 |
| STCKC | STORE CLOCK COMPARATOR | S | P A SP | | | ST | B2 | B207 | 10-89 |
| STCKE | STORE CLOCK EXTENDED | S C EK | A | $ | | ST | B2 | B278 | 7-124 |
| STCM | STORE CHARACTERS UNDER MASK | RS | A | | | ST | B2 | BE | 7-122 |
| STCTL | STORE CONTROL | RS | P A SP | | | ST | B2 | B6 | 10-89 |
| STD | STORE (long) | RX | A SP | Da | | ST | B2 | 60 | 9-11 |
| STE | STORE (short) | RX | A SP | Da | | ST | B2 | 70 | 9-11 |
| STFL | STORE FACILITY LIST | S N3 | P | | | | | B2B1 | 10-91 |
| STFPC | STORE FPC | S BF | A | Db | | ST | B2 | B29C | 19-45 |
| STH | STORE HALFWORD | RX | A | | | ST | B2 | 40 | 7-126 |
| STIDP | STORE CPU ID | S | P A SP | | | ST | B2 | B202 | 10-90 |
| STM | STORE MULTIPLE | RS | A | | | ST | B2 | 90 | 7-126 |
| STNSM | STORE THEN AND SYSTEM MASK | SI | P A | | | ST | B1 | AC | 10-103 |
| STOSM | STORE THEN OR SYSTEM MASK | SI | P A SP | | | ST | B1 | AD | 10-103 |
| STPT | STORE CPU TIMER | S | P A SP | | | ST | B2 | B209 | 10-91 |
| STPX | STORE PREFIX | S | P A SP | | | ST | B2 | B211 | 10-92 |
| STRV | STORE REVERSED | RXE N3 | A | | | ST | B2 | E33E | 7-126 |
| STRVH | STORE REVERSED | RXE N3 | A | | | ST | B2 | E33F | 7-126 |
| STSI | STORE SYSTEM INFORMATION | S C SN | P A SP | GM | | R ST | B2 | B27D | 10-92 |
| STURA | STORE USING REAL ADDRESS | RRE | P A1 SP | | | SU | | B246 | 10-104 |
| SU | SUBTRACT UNNORMALIZED (short HFP) | RX C | A SP | Da EO LS | | | B2 | 7F | 18-23 |
| SUR | SUBTRACT UNNORMALIZED (short HFP) | RR C | SP | Da EO LS | | | | 3F | 18-23 |
| SVC | SUPERVISOR CALL | RR | | ¢ | | | | 0A | 7-129 |
| SW | SUBTRACT UNNORMALIZED (long HFP) | RX C | A SP | Da EO LS | | | B2 | 6F | 18-23 |
| SWR | SUBTRACT UNNORMALIZED (long HFP) | RR C | SP | Da EO LS | | | | 2F | 18-23 |
| SXBR | SUBTRACT (extended BFP) | RRE C BF | SP | Db Xi Xo Xu Xx | | | | B34B | 19-45 |
| SXR | SUBTRACT NORMALIZED (extended HFP) | RR C | SP | Da EU EO LS | | | | 37 | 18-23 |
| TAM | TEST ADDRESSING MODE | E C N3 | | | | | | 010B | 7-129 |
| TAR | TEST ACCESS | RRE C | A1 | AS | | | U1 | B24C | 10-104 |
| TB | TEST BLOCK | RRE C | P A1 | II $ G0 | | R | | B22C | 10-107 |
| TBDR | CONVERT HFP TO BFP (long) | RRF C FX | SP | Da | | | | B351 | 9-9 |
| TBEDR | CONVERT HFP TO BFP (long to short) | RRF C FX | SP | Da | | | | B350 | 9-9 |
| TCDB | TEST DATA CLASS (long BFP) | RXE C BF | | Db | | | | ED11 | 19-46 |
| TCEB | TEST DATA CLASS (short BFP) | RXE C BF | | Db | | | | ED10 | 19-46 |
| TCXB | TEST DATA CLASS (extended BFP) | RXE C BF | SP | Db | | | | ED12 | 19-46 |
| THDER | CONVERT BFP TO HFP (short to long) | RRE C FX | | Da | | | | B358 | 9-8 |
| THDR | CONVERT BFP TO HFP (long) | RRE C FX | | Da | | | | B359 | 9-8 |
| TM | TEST UNDER MASK | SI C | A | | | | B1 | 91 | 7-130 |
| TMH | TEST UNDER MASK HIGH | RI C IR | | | | | | A70 | 7-130 |
| TML | TEST UNDER MASK LOW | RI C IR | | | | | | A71 | 7-130 |

*Figure B-2 (Part 9 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-929

| Mne-monic | Name | Characteristics | | | | | | Op Code | Page No. |
|---|---|---|---|---|---|---|---|---|---|
| TP | TEST DECIMAL | RSL C E2 | A | | | | $B_1$ | EBC0 | 8-14 |
| TPROT | TEST PROTECTION | SSE C | P $A^1$ | | | | $B_1$ | E501 | 10-109 |
| TR | TRANSLATE | SS | A | | | ST | $B_1$ $B_2$ | DC | 7-131 |
| TRACE | TRACE | RS | P A SP | T ¢ | | | $B_2$ | 99 | 10-111 |
| TRAP2 | TRAP | E TR | A | SO T | | B R ST | | 01FF | 10-112 |
| TRAP4 | TRAP | S TR | A | SO T | | B R ST | | B2FF | 10-112 |
| TRE | TRANSLATE EXTENDED | RRE C ET | A SP | | | R ST | $R_1$ $R_2$ | B2A5 | 7-132 |
| TROO | TRANSLATE ONE TO ONE | RRE C E2 | A SP | GM | | R ST | RM $R_2$ | B993 | 7-134 |
| TROT | TRANSLATE ONE TO TWO | RRE C E2 | A SP | GM | | R ST | RM $R_2$ | B992 | 7-135 |
| TRT | TRANSLATE AND TEST | SS C | A | GM | | R | $B_1$ $B_2$ | DD | 7-132 |
| TRTO | TRANSLATE TWO TO ONE | RRE C E2 | A SP | GM | | R ST | RM $R_2$ | B991 | 7-135 |
| TRTT | TRANSLATE TWO TO TWO | RRE C E2 | A SP | GM | | R ST | RM $R_2$ | B990 | 7-135 |
| TS | TEST AND SET | S C | A | $ | | ST | $B_2$ | 93 | 7-129 |
| UNPK | UNPACK | SS | A | | | ST | $B_1$ $B_2$ | F3 | 7-139 |
| UNPKA | UNPACK ASCII | SS C E2 | A SP | | | ST | $B_1$ $B_2$ | EA | 7-139 |
| UNPKU | UNPACK UNICODE | SS C E2 | A SP | | | ST | $B_1$ $B_2$ | E2 | 7-140 |
| UPT | UPDATE TREE | E C | A SP | II GM | | R ST | I4 | 0102 | 7-141 |
| X | EXCLUSIVE OR | RX C | A | | | R | $B_2$ | 57 | 7-74 |
| XC | EXCLUSIVE OR (character) | SS C | A | | | ST | $B_1$ $B_2$ | D7 | 7-74 |
| XI | EXCLUSIVE OR (immediate) | SI C | A | | | ST | $B_1$ | 97 | 7-74 |
| XR | EXCLUSIVE OR | RR C | | | | R | | 17 | 7-74 |
| ZAP | ZERO AND ADD | SS C | A | Dd DF | | ST | $B_1$ $B_2$ | F8 | 8-14 |

*Figure B-2 (Part 10 of 10). Instructions Arranged by Mnemonic*



LzLabs GmbH. Ex. 1008-930

| Op Code | Name | Mne-monic | Characteristics | Page No. |
|---|---|---|---|---|
| 0101 | PROGRAM RETURN | PR | E U A[1] SP Z[4] T ¢[2] B R ST | 10-67 |
| 0102 | UPDATE TREE | UPT | E C A SP II GM R ST I4 | 7-141 |
| 0107 | SET CLOCK PROGRAMMABLE FIELD | SCKPF | E EK P SP G0 | 10-82 |
| 010B | TEST ADDRESSING MODE | TAM | E C N3 | 7-129 |
| 010C | SET ADDRESSING MODE | SAM24 | E N3 SP T | 7-117 |
| 010D | SET ADDRESSING MODE | SAM31 | E N3 SP T | 7-117 |
| 01FF | TRAP | TRAP2 | E TR A SO T B R ST | 10-112 |
| 04 | SET PROGRAM MASK | SPM | RR L | 7-118 |
| 05 | BRANCH AND LINK | BALR | RR T B R | 7-14 |
| 06 | BRANCH ON COUNT | BCTR | RR B R | 7-18 |
| 07 | BRANCH ON CONDITION | BCR | RR ¢[1] B | 7-17 |
| 0A | SUPERVISOR CALL | SVC | RR ¢ | 7-129 |
| 0B | BRANCH AND SET MODE | BSM | RR B R | 7-16 |
| 0C | BRANCH AND SAVE AND SET MODE | BASSM | RR T B R | 7-16 |
| 0D | BRANCH AND SAVE | BASR | RR T B R | 7-15 |
| 0E | MOVE LONG | MVCL | RR C A SP II R ST R1 R2 | 7-83 |
| 0F | COMPARE LOGICAL LONG | CLCL | RR C A SP II R R1 R2 | 7-43 |
| 10 | LOAD POSITIVE | LPR | RR C IF R | 7-81 |
| 11 | LOAD NEGATIVE | LNR | RR C R | 7-80 |
| 12 | LOAD AND TEST | LTR | RR C R | 7-79 |
| 13 | LOAD COMPLEMENT | LCR | RR C IF R | 7-79 |
| 14 | AND | NR | RR C R | 7-13 |
| 15 | COMPARE LOGICAL | CLR | RR C | 7-42 |
| 16 | OR | OR | RR C R | 7-100 |
| 17 | EXCLUSIVE OR | XR | RR C R | 7-74 |
| 18 | LOAD | LR | RR R | 7-77 |
| 19 | COMPARE | CR | RR C | 7-35 |
| 1A | ADD | AR | RR C IF R | 7-12 |
| 1B | SUBTRACT | SR | RR C IF R | 7-127 |
| 1C | MULTIPLY | MR | RR SP R | 7-98 |
| 1D | DIVIDE | DR | RR SP IK R | 7-73 |
| 1E | ADD LOGICAL | ALR | RR C R | 7-13 |
| 1F | SUBTRACT LOGICAL | SLR | RR C R | 7-127 |
| 20 | LOAD POSITIVE (long HFP) | LPDR | RR C SP Da | 18-17 |
| 21 | LOAD NEGATIVE (long HFP) | LNDR | RR C SP Da | 18-16 |
| 22 | LOAD AND TEST (long HFP) | LTDR | RR C SP Da | 18-14 |
| 23 | LOAD COMPLEMENT (long HFP) | LCDR | RR C SP Da | 18-15 |
| 24 | HALVE (long HFP) | HDR | RR SP Da EU | 18-13 |
| 25 | LOAD ROUNDED (extended to long HFP) | LDXR | RR SP Da EO | 18-18 |
| 25 | LOAD ROUNDED (extended to long HFP) | LRDR | RR SP Da EO | 18-18 |
| 26 | MULTIPLY (extended HFP) | MXR | RR SP Da EU EO | 18-18 |
| 27 | MULTIPLY (long to extended HFP) | MXDR | RR SP Da EU EO | 18-18 |
| 28 | LOAD (long) | LDR | RR SP Da | 9-10 |
| 29 | COMPARE (long HFP) | CDR | RR C SP Da | 18-10 |
| 2A | ADD NORMALIZED (long HFP) | ADR | RR C SP Da EU EO LS | 18-8 |

*Figure B-3 (Part 1 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-931

| Op Code | Name | Mne-monic | Characteristics | | | | | Page No. |
|---|---|---|---|---|---|---|---|---|
| 2B | SUBTRACT NORMALIZED (long HFP) | SDR | RR C | SP | Da EU EO LS | | | 18-23 |
| 2C | MULTIPLY (long HFP) | MDR | RR | SP | Da EU EO | | | 18-18 |
| 2D | DIVIDE (long HFP) | DDR | RR | SP | Da EU EO FK | | | 18-12 |
| 2E | ADD UNNORMALIZED (long HFP) | AWR | RR C | SP | Da EO LS | | | 18-10 |
| 2F | SUBTRACT UNNORMALIZED (long HFP) | SWR | RR C | SP | Da EO LS | | | 18-23 |
| 30 | LOAD POSITIVE (short HFP) | LPER | RR C | SP | Da | | | 18-17 |
| 31 | LOAD NEGATIVE (short HFP) | LNER | RR C | SP | Da | | | 18-16 |
| 32 | LOAD AND TEST (short HFP) | LTER | RR C | SP | Da | | | 18-14 |
| 33 | LOAD COMPLEMENT (short HFP) | LCER | RR C | SP | Da | | | 18-15 |
| 34 | HALVE (short HFP) | HER | RR | SP | Da EU | | | 18-13 |
| 35 | LOAD ROUNDED (long to short HFP) | LEDR | RR | SP | Da EO | | | 18-18 |
| 35 | LOAD ROUNDED (long to short HFP) | LRER | RR | SP | Da EO | | | 18-18 |
| 36 | ADD NORMALIZED (extended HFP) | AXR | RR C | SP | Da EU EO LS | | | 18-8 |
| 37 | SUBTRACT NORMALIZED (extended HFP) | SXR | RR C | SP | Da EU EO LS | | | 18-23 |
| 38 | LOAD (short) | LER | RR | SP | Da | | | 9-10 |
| 39 | COMPARE (short HFP) | CER | RR C | SP | Da | | | 18-10 |
| 3A | ADD NORMALIZED (short HFP) | AER | RR C | SP | Da EU EO LS | | | 18-8 |
| 3B | SUBTRACT NORMALIZED (short HFP) | SER | RR C | SP | Da EU EO LS | | | 18-23 |
| 3C | MULTIPLY (short to long HFP) | MDER | RR | SP | Da EU EO | | | 18-18 |
| 3C | MULTIPLY (short to long HFP) | MER | RR | SP | Da EU EO | | | 18-18 |
| 3D | DIVIDE (short HFP) | DER | RR | SP | Da EU EO FK | | | 18-12 |
| 3E | ADD UNNORMALIZED (short HFP) | AUR | RR C | SP | Da EO LS | | | 18-10 |
| 3F | SUBTRACT UNNORMALIZED (short HFP) | SUR | RR C | SP | Da EO LS | | | 18-23 |
| 40 | STORE HALFWORD | STH | RX | A | | ST | $B_2$ | 7-126 |
| 41 | LOAD ADDRESS | LA | RX | | | R | | 7-78 |
| 42 | STORE CHARACTER | STC | RX | A | | ST | $B_2$ | 7-122 |
| 43 | INSERT CHARACTER | IC | RX | A | | R | $B_2$ | 7-76 |
| 44 | EXECUTE | EX | RX | AI SP | EX | | | 7-74 |
| 45 | BRANCH AND LINK | BAL | RX | | | B R | | 7-14 |
| 46 | BRANCH ON COUNT | BCT | RX | | | B R | | 7-18 |
| 47 | BRANCH ON CONDITION | BC | RX | | | B | | 7-17 |
| 48 | LOAD HALFWORD | LH | RX | A | | R | $B_2$ | 7-80 |
| 49 | COMPARE HALFWORD | CH | RX C | A | | | $B_2$ | 7-42 |
| 4A | ADD HALFWORD | AH | RX C | A | IF | R | $B_2$ | 7-12 |
| 4B | SUBTRACT HALFWORD | SH | RX C | A | IF | R | $B_2$ | 7-127 |
| 4C | MULTIPLY HALFWORD | MH | RX | A | | R | $B_2$ | 7-98 |
| 4D | BRANCH AND SAVE | BAS | RX | | | B R | | 7-15 |
| 4E | CONVERT TO DECIMAL | CVD | RX | A | | ST | $B_2$ | 7-67 |
| 4F | CONVERT TO BINARY | CVB | RX | A | Dd IK | R | $B_2$ | 7-66 |
| 50 | STORE | ST | RX | A | | ST | $B_2$ | 7-122 |
| 51 | LOAD ADDRESS EXTENDED | LAE | RX | | | R | $U_1$ BP | 7-78 |
| 54 | AND | N | RX C | A | | R | $B_2$ | 7-13 |
| 55 | COMPARE LOGICAL | CL | RX C | A | | | $B_2$ | 7-42 |
| 56 | OR | O | RX C | A | | R | $B_2$ | 7-100 |
| 57 | EXCLUSIVE OR | X | RX C | A | | R | $B_2$ | 7-74 |

*Figure B-3 (Part 2 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-932

| Op Code | Name | Mne-monic | Characteristics | | | | | Page No. |
|---|---|---|---|---|---|---|---|---|
| 58 | LOAD | L | RX | A | | R | $B_2$ | 7-77 |
| 59 | COMPARE | C | RX C | A | | | $B_2$ | 7-35 |
| 5A | ADD | A | RX C | A | IF | R | $B_2$ | 7-12 |
| 5B | SUBTRACT | S | RX C | A | IF | R | $B_2$ | 7-127 |
| 5C | MULTIPLY | M | RX | A SP | | R | $B_2$ | 7-98 |
| 5D | DIVIDE | D | RX | A SP | IK | R | $B_2$ | 7-73 |
| 5E | ADD LOGICAL | AL | RX C | A | | R | $B_2$ | 7-13 |
| 5F | SUBTRACT LOGICAL | SL | RX C | A | | R | $B_2$ | 7-127 |
| 60 | STORE (long) | STD | RX | A SP | Da | ST | $B_2$ | 9-11 |
| 67 | MULTIPLY (long to extended HFP) | MXD | RX | A SP | Da EU EO | | $B_2$ | 18-18 |
| 68 | LOAD (long) | LD | RX | A SP | Da | | $B_2$ | 9-10 |
| 69 | COMPARE (long HFP) | CD | RX C | A SP | Da | | $B_2$ | 18-10 |
| 6A | ADD NORMALIZED (long HFP) | AD | RX C | A SP | Da EU EO LS | | $B_2$ | 18-8 |
| 6B | SUBTRACT NORMALIZED (long HFP) | SD | RX C | A SP | Da EU EO LS | | $B_2$ | 18-23 |
| 6C | MULTIPLY (long HFP) | MD | RX | A SP | Da EU EO | | $B_2$ | 18-18 |
| 6D | DIVIDE (long HFP) | DD | RX | A SP | Da EU EO FK | | $B_2$ | 18-12 |
| 6E | ADD UNNORMALIZED (long HFP) | AW | RX C | A SP | Da EO LS | | $B_2$ | 18-10 |
| 6F | SUBTRACT UNNORMALIZED (long HFP) | SW | RX C | A SP | Da EO LS | | $B_2$ | 18-23 |
| 70 | STORE (short) | STE | RX | A SP | Da | ST | $B_2$ | 9-11 |
| 71 | MULTIPLY SINGLE | MS | RX IR | A | | R | $B_2$ | 7-99 |
| 78 | LOAD (short) | LE | RX | A SP | Da | | $B_2$ | 9-10 |
| 79 | COMPARE (short HFP) | CE | RX C | A SP | Da | | $B_2$ | 18-10 |
| 7A | ADD NORMALIZED (short HFP) | AE | RX C | A SP | Da EU EO LS | | $B_2$ | 18-8 |
| 7B | SUBTRACT NORMALIZED (short HFP) | SE | RX C | A SP | Da EU EO LS | | $B_2$ | 18-23 |
| 7C | MULTIPLY (short to long HFP) | MDE | RX | A SP | Da EU EO | | $B_2$ | 18-18 |
| 7C | MULTIPLY (short to long HFP) | ME | RX | A SP | Da EU EO | | $B_2$ | 18-18 |
| 7D | DIVIDE (short HFP) | DE | RX | A SP | Da EU EO FK | | $B_2$ | 18-12 |
| 7E | ADD UNNORMALIZED (short HFP) | AU | RX C | A SP | Da EO LS | | $B_2$ | 18-10 |
| 7F | SUBTRACT UNNORMALIZED (short HFP) | SU | RX C | A SP | Da EO LS | | $B_2$ | 18-23 |
| 80 | SET SYSTEM MASK | SSM | S | P A SP | SO | | $B_2$ | 10-87 |
| 82 | LOAD PSW | LPSW | S L | P A SP | ¢ | | $B_2$ | 10-37 |
| 83 | DIAGNOSE | | DM | P DM | | | MD | 10-19 |
| 84 | BRANCH RELATIVE ON INDEX HIGH | BRXH | RSI IR | | | B R | | 7-21 |
| 85 | BRANCH RELATIVE ON INDEX LOW OR EQ. | BRXLE | RSI IR | | | B R | | 7-21 |
| 86 | BRANCH ON INDEX HIGH | BXH | RS | | | B R | | 7-18 |
| 87 | BRANCH ON INDEX LOW OR EQUAL | BXLE | RS | | | B R | | 7-18 |
| 88 | SHIFT RIGHT SINGLE LOGICAL | SRL | RS | | | R | | 7-121 |
| 89 | SHIFT LEFT SINGLE LOGICAL | SLL | RS | | | R | | 7-120 |
| 8A | SHIFT RIGHT SINGLE | SRA | RS C | | | R | | 7-121 |
| 8B | SHIFT LEFT SINGLE | SLA | RS C | | IF | R | | 7-119 |
| 8C | SHIFT RIGHT DOUBLE LOGICAL | SRDL | RS | SP | | R | | 7-121 |
| 8D | SHIFT LEFT DOUBLE LOGICAL | SLDL | RS | SP | | R | | 7-119 |
| 8E | SHIFT RIGHT DOUBLE | SRDA | RS C | SP | | R | | 7-120 |
| 8F | SHIFT LEFT DOUBLE | SLDA | RS C | SP | IF | R | | 7-118 |
| 90 | STORE MULTIPLE | STM | RS | A | | ST | $B_2$ | 7-126 |

*Figure B-3 (Part 3 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-933

| Op Code | Name | Mne-monic | Characteristics | | | | | Page No. |
|---|---|---|---|---|---|---|---|---|
| 91 | TEST UNDER MASK | TM | SI C | A | | | $B_1$ | 7-130 |
| 92 | MOVE (immediate) | MVI | SI | A | | ST | $B_1$ | 7-83 |
| 93 | TEST AND SET | TS | S C | A | $ | ST | $B_2$ | 7-129 |
| 94 | AND (immediate) | NI | SI C | A | | ST | $B_1$ | 7-13 |
| 95 | COMPARE LOGICAL (immediate) | CLI | SI C | A | | | $B_1$ | 7-42 |
| 96 | OR (immediate) | OI | SI C | A | | ST | $B_1$ | 7-100 |
| 97 | EXCLUSIVE OR (immediate) | XI | SI C | A | | ST | $B_1$ | 7-74 |
| 98 | LOAD MULTIPLE | LM | RS | A | | R | $B_2$ | 7-80 |
| 99 | TRACE | TRACE | RS | P A SP | T ¢ | | $B_2$ | 10-111 |
| 9A | LOAD ACCESS MULTIPLE | LAM | RS | A SP | | | UB | 7-77 |
| 9B | STORE ACCESS MULTIPLE | STAM | RS | A SP | | ST | UB | 7-122 |
| A70 | TEST UNDER MASK HIGH | TMH | RI C IR | | | | | 7-130 |
| A71 | TEST UNDER MASK LOW | TML | RI C IR | | | | | 7-130 |
| A74 | BRANCH RELATIVE ON CONDITION | BRC | RI IR | | | B | | 7-20 |
| A75 | BRANCH RELATIVE AND SAVE | BRAS | RI IR | | | B R | | 7-19 |
| A76 | BRANCH RELATIVE ON COUNT | BRCT | RI IR | | | B R | | 7-21 |
| A78 | LOAD HALFWORD IMMEDIATE | LHI | RI IR | | | R | | 7-80 |
| A7A | ADD HALFWORD IMMEDIATE | AHI | RI C IR | | IF | R | | 7-12 |
| A7C | MULTIPLY HALFWORD IMMEDIATE | MHI | RI IR | | | R | | 7-98 |
| A7E | COMPARE HALFWORD IMMEDIATE | CHI | RI C IR | | | | | 7-42 |
| A8 | MOVE LONG EXTENDED | MVCLE | RS C CM | A SP | | R ST | $R_1$ $R_3$ | 7-87 |
| A9 | COMPARE LOGICAL LONG EXTENDED | CLCLE | RS C CM | A SP | | R | $R_1$ $R_3$ | 7-45 |
| AC | STORE THEN AND SYSTEM MASK | STNSM | SI | P A | | ST | $B_1$ | 10-103 |
| AD | STORE THEN OR SYSTEM MASK | STOSM | SI | P A SP | | ST | $B_1$ | 10-103 |
| AE | SIGNAL PROCESSOR | SIGP | RS C | P | $ | R | | 10-88 |
| AF | MONITOR CALL | MC | SI | SP | MO | | | 7-82 |
| B1 | LOAD REAL ADDRESS | LRA | RX C | P A[1] | AT | R | BP | 10-38 |
| B202 | STORE CPU ID | STIDP | S | P A SP | | ST | $B_2$ | 10-90 |
| B204 | SET CLOCK | SCK | S C | P A SP | | | $B_2$ | 10-81 |
| B205 | STORE CLOCK | STCK | S C | A | $ | ST | $B_2$ | 7-123 |
| B206 | SET CLOCK COMPARATOR | SCKC | S | P A SP | | | $B_2$ | 10-82 |
| B207 | STORE CLOCK COMPARATOR | STCKC | S | P A SP | | ST | $B_2$ | 10-89 |
| B208 | SET CPU TIMER | SPT | S | P A SP | | | $B_2$ | 10-82 |
| B209 | STORE CPU TIMER | STPT | S | P A SP | | ST | $B_2$ | 10-91 |
| B20A | SET PSW KEY FROM ADDRESS | SPKA | S | Q | | | | 10-83 |
| B20B | INSERT PSW KEY | IPK | S | Q | G2 | R | | 10-24 |
| B20D | PURGE TLB | PTLB | S | P | $ | | | 10-76 |
| B210 | SET PREFIX | SPX | S | P A SP | $ | | $B_2$ | 10-83 |
| B211 | STORE PREFIX | STPX | S | P A SP | | ST | $B_2$ | 10-92 |
| B212 | STORE CPU ADDRESS | STAP | S | P A SP | | ST | $B_2$ | 10-90 |
| B218 | PROGRAM CALL | PC | S | Q A[1] | Z[1] T ¢ GM | B R ST | | 10-52 |
| B218 | PROGRAM CALL FAST | PCF | S PC | A[1] | Z[5] ¢ G4 | B R ST | | 10-63 |
| B219 | SET ADDRESS SPACE CONTROL | SAC | S | Q SP | SW ¢ | | | 10-79 |
| B21A | COMPARE AND FORM CODEWORD | CFC | S C | A SP | II GM | R | I1 | 7-36 |
| B221 | INVALIDATE PAGE TABLE ENTRY | IPTE | RRE | P A[1] | $ | | | 10-26 |

*Figure B-3 (Part 4 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-934

| Op Code | Name | Mne-monic | Characteristics | | | | Page No. |
|---|---|---|---|---|---|---|---|
| B222 | INSERT PROGRAM MASK | IPM | RRE | | | R | 7-77 |
| B223 | INSERT VIRTUAL STORAGE KEY | IVSK | RRE | Q A[1] | SO | R R[2] | 10-25 |
| B224 | INSERT ADDRESS SPACE CONTROL | IAC | RRE C | Q | SO | R | 10-23 |
| B225 | SET SECONDARY ASN | SSAR | RRE | A[1] | Z[3] T ¢ | | 10-84 |
| B226 | EXTRACT PRIMARY ASN | EPAR | RRE | Q | SO | R | 10-19 |
| B227 | EXTRACT SECONDARY ASN | ESAR | RRE | Q | SO | R | 10-20 |
| B228 | PROGRAM TRANSFER | PT | RRE | Q A[1] SP | Z[2] T ¢ | B | 10-70 |
| B229 | INSERT STORAGE KEY EXTENDED | ISKE | RRE | P A[1] | | | 10-25 |
| B22A | RESET REFERENCE BIT EXTENDED | RRBE | RRE C | P A[1] | | | 10-76 |
| B22B | SET STORAGE KEY EXTENDED | SSKE | RRE | P A[1] | ¢ | | 10-87 |
| B22C | TEST BLOCK | TB | RRE C | P A[1] | II $ G0 | R | 10-107 |
| B22D | DIVIDE (extended HFP) | DXR | RRE | SP | Da EU EO FK | | 18-12 |
| B22E | PAGE IN | PGIN | RRE C ES | P A[1] | ¢ | | 10-50 |
| B22F | PAGE OUT | PGOUT | RRE C ES | P A[1] | ¢ | | 10-51 |
| B240 | BRANCH AND STACK | BAKR | RRE | A[1] | SF T | B ST | 10-10 |
| B241 | CHECKSUM | CKSM | RRE C CK | A SP | | R R[2] | 7-22 |
| B244 | SQUARE ROOT (long HFP) | SQDR | RRE QR | SP | Da SQ | | 18-21 |
| B245 | SQUARE ROOT (short HFP) | SQER | RRE QR | SP | Da SQ | | 18-21 |
| B246 | STORE USING REAL ADDRESS | STURA | RRE | P A[1] SP | | SU | 10-104 |
| B247 | MODIFY STACKED STATE | MSTA | RRE | A[1] SP | SE | ST | 10-40 |
| B248 | PURGE ALB | PALB | RRE | P | $ | | 10-76 |
| B249 | EXTRACT STACKED REGISTERS | EREG | RRE | A[1] | SE | R U[1] U[2] | 10-20 |
| B24A | EXTRACT STACKED STATE | ESTA | RRE C | A[1] SP | SE | R | 10-21 |
| B24B | LOAD USING REAL ADDRESS | LURA | RRE | P A[1] SP | | R | 10-40 |
| B24C | TEST ACCESS | TAR | RRE C | A[1] | AS | U[1] | 10-104 |
| B24D | COPY ACCESS | CPYA | RRE | | | U[1] U[2] | 7-72 |
| B24E | SET ACCESS | SAR | RRE | | | U[1] | 7-117 |
| B24F | EXTRACT ACCESS | EAR | RRE | | | R U[2] | 7-75 |
| B250 | COMPARE AND SWAP AND PURGE | CSP | RRE C | P A[1] SP | $ | R ST R[2] | 10-17 |
| B252 | MULTIPLY SINGLE | MSR | RRE IR | | | R | 7-99 |
| B254 | MOVE PAGE (facility 1) | MVPG | RRE C M1 | A[1] SP | G0 | ST R[1] R[2] | 7-93 |
| B254 | MOVE PAGE (facility 2) | MVPG | RRE C M2 | Q A[1] SP | G0 | ST R[1] R[2] | 7-93 |
| B255 | MOVE STRING | MVST | RRE C SR | A SP | G0 | R ST R[1] R[2] | 7-95 |
| B257 | COMPARE UNTIL SUBSTRING EQUAL | CUSE | RRE C | A SP | II G0 | R[1] R[2] | 7-51 |
| B258 | BRANCH IN SUBSPACE GROUP | BSG | RRE SG | A[1] | SO T | B R R[2] | 10-13 |
| B25A | BRANCH AND SET AUTHORITY | BSA | RRE BS | Q A[1] | SO T | B R | 10-7 |
| B25D | COMPARE LOGICAL STRING | CLST | RRE C SR | A SP | G0 | R R[1] R[2] | 7-50 |
| B25E | SEARCH STRING | SRST | RRE C SR | A SP | G0 | R R[2] | 7-116 |
| B277 | RESUME PROGRAM | RP | S L RP | Q A SP | SW T | B R B[2] | 10-77 |
| B278 | STORE CLOCK EXTENDED | STCKE | S C EK | A | $ | ST B[2] | 7-124 |
| B279 | SET ADDRESS SPACE CONTROL FAST | SACF | S SA | Q SP | SW | | 10-79 |
| B27D | STORE SYSTEM INFORMATION | STSI | S C SN | P A SP | GM | R ST B[2] | 10-92 |
| B299 | SET ROUNDING MODE | SRNM | S BF | | Db | | 19-44 |
| B29C | STORE FPC | STFPC | S BF | A | Db | ST B[2] | 19-45 |
| B29D | LOAD FPC | LFPC | S BF | A SP | Db | B[2] | 19-37 |

*Figure B-3 (Part 5 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-935

| Op Code | Name | Mne-monic | Characteristics | | | | | Page No. |
|---|---|---|---|---|---|---|---|---|
| B2A5 | TRANSLATE EXTENDED | TRE | RRE C ET | A SP | | R ST | R1 R2 | 7-132 |
| B2A6 | CONVERT UNICODE TO UTF-8 | CUUTF | RRE C ET | A SP | | R ST | R1 R2 | 7-67 |
| B2A7 | CONVERT UTF-8 TO UNICODE | CUTFU | RRE C ET | A SP | | R ST | R1 R2 | 7-70 |
| B2B1 | STORE FACILITY LIST | STFL | S N3 | P | | | | 10-91 |
| B2FF | TRAP | TRAP4 | S TR | A | SO T | B R ST | | 10-112 |
| B300 | LOAD POSITIVE (short BFP) | LPEBR | RRE C BF | | Db | | | 19-39 |
| B301 | LOAD NEGATIVE (short BFP) | LNEBR | RRE C BF | | Db | | | 19-38 |
| B302 | LOAD AND TEST (short BFP) | LTEBR | RRE C BF | | Db Xi | | | 19-35 |
| B303 | LOAD COMPLEMENT (short BFP) | LCEBR | RRE C BF | | Db | | | 19-35 |
| B304 | LOAD LENGTHENED (short to long BFP) | LDEBR | RRE BF | | Db Xi | | | 19-38 |
| B305 | LOAD LENGTHENED (long to ext. BFP) | LXDBR | RRE BF | SP | Db Xi | | | 19-38 |
| B306 | LOAD LENGTHENED (short to ext. BFP) | LXEBR | RRE BF | SP | Db Xi | | | 19-38 |
| B307 | MULTIPLY (long to extended BFP) | MXDBR | RRE BF | SP | Db Xi | | | 19-40 |
| B308 | COMPARE AND SIGNAL (short BFP) | KEBR | RRE C BF | | Db Xi | | | 19-24 |
| B309 | COMPARE (short BFP) | CEBR | RRE C BF | | Db Xi | | | 19-23 |
| B30A | ADD (short BFP) | AEBR | RRE C BF | | Db Xi Xo Xu Xx | | | 19-18 |
| B30B | SUBTRACT (short BFP) | SEBR | RRE C BF | | Db Xi Xo Xu Xx | | | 19-45 |
| B30C | MULTIPLY (short to long BFP) | MDEBR | RRE BF | | Db Xi | | | 19-40 |
| B30D | DIVIDE (short BFP) | DEBR | RRE BF | | Db Xi Xz Xo Xu Xx | | | 19-29 |
| B30E | MULTIPLY AND ADD (short BFP) | MAEBR | RRF BF | | Db Xi Xo Xu Xx | | | 19-42 |
| B30F | MULTIPLY AND SUBTRACT (short BFP) | MSEBR | RRF BF | | Db Xi Xo Xu Xx | | | 19-42 |
| B310 | LOAD POSITIVE (long BFP) | LPDBR | RRE C BF | | Db | | | 19-39 |
| B311 | LOAD NEGATIVE (long BFP) | LNDBR | RRE C BF | | Db | | | 19-38 |
| B312 | LOAD AND TEST (long BFP) | LTDBR | RRE C BF | | Db Xi | | | 19-35 |
| B313 | LOAD COMPLEMENT (long BFP) | LCDBR | RRE C BF | | Db | | | 19-35 |
| B314 | SQUARE ROOT (short BFP) | SQEBR | RRE BF | | Db Xi Xx | | | 19-45 |
| B315 | SQUARE ROOT (long BFP) | SQDBR | RRE BF | | Db Xi Xx | | | 19-45 |
| B316 | SQUARE ROOT (extended BFP) | SQXBR | RRE BF | SP | Db Xi Xx | | | 19-45 |
| B317 | MULTIPLY (short BFP) | MEEBR | RRE BF | | Db Xi Xo Xu Xx | | | 19-40 |
| B318 | COMPARE AND SIGNAL (long BFP) | KDBR | RRE C BF | | Db Xi | | | 19-24 |
| B319 | COMPARE (long BFP) | CDBR | RRE C BF | | Db Xi | | | 19-23 |
| B31A | ADD (long BFP) | ADBR | RRE C BF | | Db Xi Xo Xu Xx | | | 19-18 |
| B31B | SUBTRACT (long BFP) | SDBR | RRE C BF | | Db Xi Xo Xu Xx | | | 19-45 |
| B31C | MULTIPLY (long BFP) | MDBR | RRE BF | | Db Xi Xo Xu Xx | | | 19-40 |
| B31D | DIVIDE (long BFP) | DDBR | RRE BF | | Db Xi Xz Xo Xu Xx | | | 19-29 |
| B31E | MULTIPLY AND ADD (long BFP) | MADBR | RRF BF | | Db Xi Xo Xu Xx | | | 19-42 |
| B31F | MULTIPLY AND SUBTRACT (long BFP) | MSDBR | RRF BF | | Db Xi Xo Xu Xx | | | 19-42 |
| B324 | LOAD LENGTHENED (short to long HFP) | LDER | RRE HX | | Da | | | 18-16 |
| B325 | LOAD LENGTHENED (long to ext. HFP) | LXDR | RRE HX | SP | Da | | | 18-16 |
| B326 | LOAD LENGTHENED (short to ext. HFP) | LXER | RRE HX | SP | Da | | | 18-16 |
| B32E | MULTIPLY AND ADD (short HFP) | MAER | RRF HM | | Da EU EO | | | 18-20 |
| B32F | MULTIPLY AND SUBTRACT (short HFP) | MSER | RRF HM | | Da EU EO | | | 18-20 |
| B336 | SQUARE ROOT (extended HFP) | SQXR | RRE HX | SP | Da SQ | | | 18-21 |
| B337 | MULTIPLY (short HFP) | MEER | RRE HX | | Da EU EO | | | 18-18 |
| B33E | MULTIPLY AND ADD (long HFP) | MADR | RRF HM | | Da EU EO | | | 18-20 |

*Figure B-3 (Part 6 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-936

| Op Code | Name | Mne-monic | Characteristics | | | | | | Page No. |
|---|---|---|---|---|---|---|---|---|---|
| B33F | MULTIPLY AND SUBTRACT (long HFP) | MSDR | RRF | HM | | Da EU EO | | | 18-20 |
| B340 | LOAD POSITIVE (extended BFP) | LPXBR | RRE C | BF | SP | Db | | | 19-39 |
| B341 | LOAD NEGATIVE (extended BFP) | LNXBR | RRE C | BF | SP | Db | | | 19-38 |
| B342 | LOAD AND TEST (extended BFP) | LTXBR | RRE C | BF | SP | Db Xi | | | 19-35 |
| B343 | LOAD COMPLEMENT (extended BFP) | LCXBR | RRE C | BF | SP | Db | | | 19-35 |
| B344 | LOAD ROUNDED (long to short BFP) | LEDBR | RRE | BF | | Db Xi Xo Xu Xx | | | 19-39 |
| B345 | LOAD ROUNDED (extended to long BFP) | LDXBR | RRE | BF | SP | Db Xi Xo Xu Xx | | | 19-39 |
| B346 | LOAD ROUNDED (extended to short BFP) | LEXBR | RRE | BF | SP | Db Xi Xo Xu Xx | | | 19-39 |
| B347 | LOAD FP INTEGER (extended BFP) | FIXBR | RRF | BF | SP | Db Xi Xx | | | 19-36 |
| B348 | COMPARE AND SIGNAL (extended BFP) | KXBR | RRE C | BF | SP | Db Xi | | | 19-24 |
| B349 | COMPARE (extended BFP) | CXBR | RRE C | BF | SP | Db Xi | | | 19-23 |
| B34A | ADD (extended BFP) | AXBR | RRE C | BF | SP | Db Xi Xo Xu Xx | | | 19-18 |
| B34B | SUBTRACT (extended BFP) | SXBR | RRE C | BF | SP | Db Xi Xo Xu Xx | | | 19-45 |
| B34C | MULTIPLY (extended BFP) | MXBR | RRE | BF | SP | Db Xi Xo Xu Xx | | | 19-40 |
| B34D | DIVIDE (extended BFP) | DXBR | RRE | BF | SP | Db Xi Xz Xo Xu Xx | | | 19-29 |
| B350 | CONVERT HFP TO BFP (long to short) | TBEDR | RRF C | FX | SP | Da | | | 9-9 |
| B351 | CONVERT HFP TO BFP (long) | TBDR | RRF C | FX | SP | Da | | | 9-9 |
| B353 | DIVIDE TO INTEGER (short BFP) | DIEBR | RRF C | BF | SP | Db Xi Xu Xx | | | 19-30 |
| B357 | LOAD FP INTEGER (short BFP) | FIEBR | RRF | BF | SP | Db Xi Xx | | | 19-36 |
| B358 | CONVERT BFP TO HFP (short to long) | THDER | RRE C | FX | | Da | | | 9-8 |
| B359 | CONVERT BFP TO HFP (long) | THDR | RRE C | FX | | Da | | | 9-8 |
| B35B | DIVIDE TO INTEGER (long BFP) | DIDBR | RRF C | BF | SP | Db Xi Xu Xx | | | 19-30 |
| B35F | LOAD FP INTEGER (long BFP) | FIDBR | RRF | BF | SP | Db Xi Xx | | | 19-36 |
| B360 | LOAD POSITIVE (extended HFP) | LPXR | RRE C | HX | SP | Da | | | 18-17 |
| B361 | LOAD NEGATIVE (extended HFP) | LNXR | RRE C | HX | SP | Da | | | 18-16 |
| B362 | LOAD AND TEST (extended HFP) | LTXR | RRE C | HX | SP | Da | | | 18-14 |
| B363 | LOAD COMPLEMENT (extended HFP) | LCXR | RRE C | HX | SP | Da | | | 18-15 |
| B365 | LOAD (extended) | LXR | RRE | FX | SP | Da | | | 9-10 |
| B366 | LOAD ROUNDED (extended to short HFP) | LEXR | RRE | HX | SP | Da EO | | | 18-18 |
| B367 | LOAD FP INTEGER (extended HFP) | FIXR | RRE | HX | SP | Da | | | 18-15 |
| B369 | COMPARE (extended HFP) | CXR | RRE C | HX | SP | Da | | | 18-10 |
| B374 | LOAD ZERO (short) | LZER | RRE | FX | | Da | | | 9-11 |
| B375 | LOAD ZERO (long) | LZDR | RRE | FX | | Da | | | 9-11 |
| B376 | LOAD ZERO (extended) | LZXR | RRE | FX | SP | Da | | | 9-11 |
| B377 | LOAD FP INTEGER (short HFP) | FIER | RRE | HX | | Da | | | 18-15 |
| B37F | LOAD FP INTEGER (long HFP) | FIDR | RRE | HX | | Da | | | 18-15 |
| B384 | SET FPC | SFPC | RRE | BF | SP | Db | | | 19-44 |
| B38C | EXTRACT FPC | EFPC | RRE | BF | | Db | | | 19-34 |
| B394 | CONVERT FROM FIXED (32 to short BFP) | CEFBR | RRE | BF | | Db Xx | | | 19-26 |
| B395 | CONVERT FROM FIXED (32 to long BFP) | CDFBR | RRE | BF | | Db | | | 19-26 |
| B396 | CONVERT FROM FIXED (32 to ext. BFP) | CXFBR | RRE | BF | SP | Db | | | 19-26 |
| B398 | CONVERT TO FIXED (short BFP to 32) | CFEBR | RRF C | BF | SP | Db Xi Xx | R | | 19-26 |
| B399 | CONVERT TO FIXED (long BFP to 32) | CFDBR | RRF C | BF | SP | Db Xi Xx | R | | 19-26 |
| B39A | CONVERT TO FIXED (ext. BFP to 32) | CFXBR | RRF C | BF | SP | Db Xi Xx | R | | 19-26 |
| B3B4 | CONVERT FROM FIXED (32 to short HFP) | CEFR | RRE | HX | | Da | | | 18-11 |

*Figure B-3 (Part 7 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-937

| Op Code | Name | Mne-monic | Characteristics | | | | | Page No. |
|---|---|---|---|---|---|---|---|---|
| B3B5 | CONVERT FROM FIXED (32 to long HFP) | CDFR | RRE HX | | Da | | | 18-11 |
| B3B6 | CONVERT FROM FIXED (32 to ext. HFP) | CXFR | RRE HX SP | | Da | | | 18-11 |
| B3B8 | CONVERT TO FIXED (short HFP to 32) | CFER | RRF C HX SP | | Da | R | | 18-12 |
| B3B9 | CONVERT TO FIXED (long HFP to 32) | CFDR | RRF C HX SP | | Da | R | | 18-12 |
| B3BA | CONVERT TO FIXED (ext. HFP to 32) | CFXR | RRF C HX SP | | Da | R | | 18-12 |
| B6 | STORE CONTROL | STCTL | RS | P A SP | | ST | $B_2$ | 10-89 |
| B7 | LOAD CONTROL | LCTL | RS | P A SP | | | $B_2$ | 10-36 |
| B91E | COMPUTE MESSAGE AUTHENTICATION CODE | KMAC | RRE C MS | A SP | GM I1 | ST | $R_2$ | 7-61 |
| B91F | LOAD REVERSED | LRVR | RRE N3 | | | R | | 7-81 |
| B92E | CIPHER MESSAGE | KM | RRE C MS | A SP | GM I1 | ST | $R_1$ $R_2$ | 7-26 |
| B92F | CIPHER MESSAGE WITH CHAINING | KMC | RRE C MS | A SP | GM I1 | ST | $R_1$ $R_2$ | 7-26 |
| B93E | COMPUTE INTERMEDIATE MESSAGE DIGEST | KIMD | RRE C MS | A SP | GM I1 | ST | $R_2$ | 7-55 |
| B93F | COMPUTE LAST MESSAGE DIGEST | KLMD | RRE C MS | A SP | GM I1 | ST | $R_2$ | 7-55 |
| B98D | EXTRACT PSW | EPSW | RRE N3 | | | R | | 7-76 |
| B990 | TRANSLATE TWO TO TWO | TRTT | RRE C E2 | A SP | GM | R ST | RM $R_2$ | 7-135 |
| B991 | TRANSLATE TWO TO ONE | TRTO | RRE C E2 | A SP | GM | R ST | RM $R_2$ | 7-135 |
| B992 | TRANSLATE ONE TO TWO | TROT | RRE C E2 | A SP | GM | R ST | RM $R_2$ | 7-135 |
| B993 | TRANSLATE ONE TO ONE | TROO | RRE C E2 | A SP | GM | R ST | RM $R_2$ | 7-134 |
| B996 | MULTIPLY LOGICAL | MLR | RRE N3 | SP | | R | | 7-99 |
| B997 | DIVIDE LOGICAL | DLR | RRE N3 | SP | IK | R | | 7-73 |
| B998 | ADD LOGICAL WITH CARRY | ALCR | RRE C N3 | | | R | | 7-13 |
| B999 | SUBTRACT LOGICAL WITH BORROW | SLBR | RRE C N3 | | | R | | 7-128 |
| BA | COMPARE AND SWAP | CS | RS C | A SP | $ | R ST | $B_2$ | 7-40 |
| BB | COMPARE DOUBLE AND SWAP | CDS | RS C | A SP | $ | R ST | $B_2$ | 7-40 |
| BD | COMPARE LOGICAL CHARS. UNDER MASK | CLM | RS C | A | | | $B_2$ | 7-43 |
| BE | STORE CHARACTERS UNDER MASK | STCM | RS | A | | ST | $B_2$ | 7-122 |
| BF | INSERT CHARACTERS UNDER MASK | ICM | RS C | A | | R | $B_2$ | 7-76 |
| C00 | LOAD ADDRESS RELATIVE LONG | LARL | RIL N3 | | | R | | 7-79 |
| C04 | BRANCH RELATIVE ON CONDITION LONG | BRCL | RIL N3 | | | B | | 7-20 |
| C05 | BRANCH RELATIVE AND SAVE LONG | BRASL | RIL N3 | | | B R | | 7-19 |
| D1 | MOVE NUMERICS | MVN | SS | A | | ST | $B_1$ $B_2$ | 7-93 |
| D2 | MOVE (character) | MVC | SS | A | | ST | $B_1$ $B_2$ | 7-83 |
| D3 | MOVE ZONES | MVZ | SS | A | | ST | $B_1$ $B_2$ | 7-97 |
| D4 | AND (character) | NC | SS C | A | | ST | $B_1$ $B_2$ | 7-13 |
| D5 | COMPARE LOGICAL (character) | CLC | SS C | A | | | $B_1$ $B_2$ | 7-42 |
| D6 | OR (character) | OC | SS C | A | | ST | $B_1$ $B_2$ | 7-100 |
| D7 | EXCLUSIVE OR (character) | XC | SS C | A | | ST | $B_1$ $B_2$ | 7-74 |
| D9 | MOVE WITH KEY | MVCK | SS C | Q A | | ST | $B_1$ $B_2$ | 10-47 |
| DA | MOVE TO PRIMARY | MVCP | SS C | Q A | SO ¢ | ST | | 10-45 |
| DB | MOVE TO SECONDARY | MVCS | SS C | Q A | SO ¢ | ST | | 10-45 |
| DC | TRANSLATE | TR | SS | A | | ST | $B_1$ $B_2$ | 7-131 |
| DD | TRANSLATE AND TEST | TRT | SS C | A | GM | R | $B_1$ $B_2$ | 7-132 |
| DE | EDIT | ED | SS C | A | Dd | ST | $B_1$ $B_2$ | 8-7 |
| DF | EDIT AND MARK | EDMK | SS C | A | Dd G1 | R ST | $B_1$ $B_2$ | 8-12 |
| E1 | PACK UNICODE | PKU | SS E2 | A SP | | ST | $B_1$ $B_2$ | 7-102 |

*Figure B-3 (Part 8 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-938

| Op Code | Name | Mne-monic | Characteristics | | | | | Page No. |
|---|---|---|---|---|---|---|---|---|
| E2 | UNPACK UNICODE | UNPKU | SS C E2 | A SP | | ST | B1 B2 | 7-140 |
| E31E | LOAD REVERSED | LRV | RXE N3 | A | | R | B2 | 7-81 |
| E31F | LOAD REVERSED | LRVH | RXE N3 | A | | R | B2 | 7-81 |
| E33E | STORE REVERSED | STRV | RXE N3 | A | | ST | B2 | 7-126 |
| E33F | STORE REVERSED | STRVH | RXE N3 | A | | ST | B2 | 7-126 |
| E396 | MULTIPLY LOGICAL | ML | RXE N3 | A SP | | R | B2 | 7-99 |
| E397 | DIVIDE LOGICAL | DL | RXE N3 | A SP | IK | R | B2 | 7-73 |
| E398 | ADD LOGICAL WITH CARRY | ALC | RXE C N3 | A | | R | B2 | 7-13 |
| E399 | SUBTRACT LOGICAL WITH BORROW | SLB | RXE C N3 | A | | R | B2 | 7-128 |
| E500 | LOAD ADDRESS SPACE PARAMETERS | LASP | SSE C | P A[1] SP | AS | | B1 | 10-28 |
| E501 | TEST PROTECTION | TPROT | SSE C | P A[1] | | | B1 | 10-109 |
| E50E | MOVE WITH SOURCE KEY | MVCSK | SSE | Q A | GM | ST | B1 B2 | 10-48 |
| E50F | MOVE WITH DESTINATION KEY | MVCDK | SSE | Q A | GM | ST | B1 B2 | 10-47 |
| E8 | MOVE INVERSE | MVCIN | SS | A | | ST | B1 B2 | 7-83 |
| E9 | PACK ASCII | PKA | SS E2 | A SP | | ST | B1 B2 | 7-101 |
| EA | UNPACK ASCII | UNPKA | SS C E2 | A SP | | ST | B1 B2 | 7-139 |
| EB1D | ROTATE LEFT SINGLE LOGICAL | RLL | RSE N3 | | | R | | 7-116 |
| EB8E | MOVE LONG UNICODE | MVCLU | RSE C E2 | A SP | | R ST | R1 R2 | 7-90 |
| EB8F | COMPARE LOGICAL LONG UNICODE | CLCLU | RSE C E2 | A SP | | R | R1 R2 | 7-47 |
| EBC0 | TEST DECIMAL | TP | RSL C E2 | A | | | B1 | 8-14 |
| ED04 | LOAD LENGTHENED (short to long BFP) | LDEB | RXE BF | A | Db Xi | | B2 | 19-38 |
| ED05 | LOAD LENGTHENED (long to ext. BFP) | LXDB | RXE BF | A SP | Db Xi | | B2 | 19-38 |
| ED06 | LOAD LENGTHENED (short to ext. BFP) | LXEB | RXE BF | A SP | Db Xi | | B2 | 19-38 |
| ED07 | MULTIPLY (long to extended BFP) | MXDB | RXE BF | A SP | Db Xi | | B2 | 19-40 |
| ED08 | COMPARE AND SIGNAL (short BFP) | KEB | RXE C BF | A | Db Xi | | B2 | 19-24 |
| ED09 | COMPARE (short BFP) | CEB | RXE C BF | A | Db Xi | | B2 | 19-23 |
| ED0A | ADD (short BFP) | AEB | RXE C BF | A | Db Xi Xo Xu Xx | | B2 | 19-18 |
| ED0B | SUBTRACT (short BFP) | SEB | RXE C BF | A | Db Xi Xo Xu Xx | | B2 | 19-45 |
| ED0C | MULTIPLY (short to long BFP) | MDEB | RXE BF | A | Db Xi | | B2 | 19-40 |
| ED0D | DIVIDE (short BFP) | DEB | RXE BF | A | Db Xi Xz Xo Xu Xx | | B2 | 19-29 |
| ED0E | MULTIPLY AND ADD (short BFP) | MAEB | RXF BF | A | Db Xi Xo Xu Xx | | B2 | 19-42 |
| ED0F | MULTIPLY AND SUBTRACT (short BFP) | MSEB | RXF BF | A | Db Xi Xo Xu Xx | | B2 | 19-42 |
| ED10 | TEST DATA CLASS (short BFP) | TCEB | RXE C BF | | Db | | | 19-46 |
| ED11 | TEST DATA CLASS (long BFP) | TCDB | RXE C BF | | Db | | | 19-46 |
| ED12 | TEST DATA CLASS (extended BFP) | TCXB | RXE C BF | SP | Db | | | 19-46 |
| ED14 | SQUARE ROOT (short BFP) | SQEB | RXE BF | A | Db Xi Xx | | B2 | 19-45 |
| ED15 | SQUARE ROOT (long BFP) | SQDB | RXE BF | A | Db Xi Xx | | B2 | 19-45 |
| ED17 | MULTIPLY (short BFP) | MEEB | RXE BF | A | Db Xi Xo Xu Xx | | B2 | 19-40 |
| ED18 | COMPARE AND SIGNAL (long BFP) | KDB | RXE C BF | A | Db Xi | | B2 | 19-24 |
| ED19 | COMPARE (long BFP) | CDB | RXE C BF | A | Db Xi | | B2 | 19-23 |
| ED1A | ADD (long BFP) | ADB | RXE C BF | A | Db Xi Xo Xu Xx | | B2 | 19-18 |
| ED1B | SUBTRACT (long BFP) | SDB | RXE C BF | A | Db Xi Xo Xu Xx | | B2 | 19-45 |
| ED1C | MULTIPLY (long BFP) | MDB | RXE BF | A | Db Xi Xo Xu Xx | | B2 | 19-40 |
| ED1D | DIVIDE (long BFP) | DDB | RXE BF | A | Db Xi Xz Xo Xu Xx | | B2 | 19-29 |
| ED1E | MULTIPLY AND ADD (long BFP) | MADB | RXF BF | A | Db Xi Xo Xu Xx | | B2 | 19-42 |

*Figure B-3 (Part 9 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-939

| Op Code | Name | Mne-monic | Characteristics | | | | | | | | Page No. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ED1F | MULTIPLY AND SUBTRACT (long BFP) | MSDB | RXF | BF | A | | Db Xi | Xo Xu Xx | | $B_2$ | 19-42 |
| ED24 | LOAD LENGTHENED (short to long HFP) | LDE | RXE | HX | A | | Da | | | $B_2$ | 18-16 |
| ED25 | LOAD LENGTHENED (long to ext. HFP) | LXD | RXE | HX | A | SP | Da | | | $B_2$ | 18-16 |
| ED26 | LOAD LENGTHENED (short to ext. HFP) | LXE | RXE | HX | A | SP | Da | | | $B_2$ | 18-16 |
| ED2E | MULTIPLY AND ADD (short HFP) | MAE | RXF | HM | A | | Da EU EO | | | $B_2$ | 18-20 |
| ED2F | MULTIPLY AND SUBTRACT (short HFP) | MSE | RXF | HM | A | | Da EU EO | | | $B_2$ | 18-20 |
| ED34 | SQUARE ROOT (short HFP) | SQE | RXE | HX | A | | Da | SQ | | $B_2$ | 18-21 |
| ED35 | SQUARE ROOT (long HFP) | SQD | RXE | HX | A | | Da | SQ | | $B_2$ | 18-21 |
| ED37 | MULTIPLY (short HFP) | MEE | RXE | HX | A | | Da EU EO | | | $B_2$ | 18-18 |
| ED3E | MULTIPLY AND ADD (long HFP) | MAD | RXF | HM | A | | Da EU EO | | | $B_2$ | 18-20 |
| ED3F | MULTIPLY AND SUBTRACT (long HFP) | MSD | RXF | HM | A | | Da EU EO | | | $B_2$ | 18-20 |
| EE | PERFORM LOCKED OPERATION | PLO | SS | C PL | A | SP | | $ GM | R ST | FC | 7-103 |
| F0 | SHIFT AND ROUND DECIMAL | SRP | SS | C | A | | Dd DF | | ST | $B_1$ | 8-13 |
| F1 | MOVE WITH OFFSET | MVO | SS | | A | | | | ST | $B_1$ $B_2$ | 7-97 |
| F2 | PACK | PACK | SS | | A | | | | ST | $B_1$ $B_2$ | 7-100 |
| F3 | UNPACK | UNPK | SS | | A | | | | ST | $B_1$ $B_2$ | 7-139 |
| F8 | ZERO AND ADD | ZAP | SS | C | A | | Dd DF | | ST | $B_1$ $B_2$ | 8-14 |
| F9 | COMPARE DECIMAL | CP | SS | C | A | | Dd | | | $B_1$ $B_2$ | 8-6 |
| FA | ADD DECIMAL | AP | SS | C | A | | Dd DF | | ST | $B_1$ $B_2$ | 8-6 |
| FB | SUBTRACT DECIMAL | SP | SS | C | A | | Dd DF | | ST | $B_1$ $B_2$ | 8-14 |
| FC | MULTIPLY DECIMAL | MP | SS | | A | SP | Dd | | ST | $B_1$ $B_2$ | 8-12 |
| FD | DIVIDE DECIMAL | DP | SS | | A | SP | Dd | DK | ST | $B_1$ $B_2$ | 8-7 |

*Figure B-3 (Part 10 of 10). Instructions Arranged by Operation Code*



LzLabs GmbH. Ex. 1008-940

# Appendix C. Condition-Code Settings

This appendix lists the condition-code setting for instructions in ESA/390 which set the condition code. In addition to those instructions listed which set the condition code, the condition code is set unpredictably by PROGRAM RETURN, and it may be changed by DIAGNOSE and the target of EXECUTE. The condition code is loaded by LOAD PSW, RESUME PROGRAM, and SET PROGRAM MASK and by an interruption. The condition code is set to zero by initial CPU reset and is loaded by the successful conclusion of the initial-program-loading sequence.

The condition codes for the vector facility are not included in this appendix. See the publication *IBM Enterprise Systems Architecture/390 Vector Operations*, SA22-7207 for the condition codes set by vector instructions.

Some models may offer instructions which set the condition code and do not appear in this document, such as those provided for assists or as part of special or custom features.

| Instruction | Condition Code | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| ADD (gen) | Zero | < zero | > zero | Overflow |
| ADD (BFP) | Zero | < zero | > zero | NaN |
| ADD DECIMAL | Zero | < zero | > zero | Overflow |
| ADD HALFWORD | Zero | < zero | > zero | Overflow |
| ADD HALFWORD IMMEDIATE | Zero | < zero | > zero | Overflow |
| ADD LOGICAL | Zero, no carry | Not zero, no carry | Zero, carry | Not zero, carry |
| ADD LOGICAL WITH CARRY | Zero, no carry | Not zero, no carry | Zero, carry | Not zero, carry |
| ADD NORMALIZED | Zero | < zero | > zero | -- |
| ADD UNNORMALIZED | Zero | < zero | > zero | -- |
| AND | Zero | Not zero | -- | -- |
| CANCEL SUBCHANNEL | Function initiated | -- | -- | Not operational |
| CHECKSUM | Checksum complete | -- | -- | CPU-determined completion |
| CIPHER MESSAGE | Normal completion | -- | -- | Partial completion |
| CIPHER MESSAGE WITH CHAINING | Normal completion | -- | -- | Partial completion |
| CLEAR SUBCHANNEL | Function initiated | -- | -- | Not operational |
| COMPARE (gen, HFP) | Equal | Low | High | -- |
| COMPARE (BFP) | Equal | Low | High | Unordered |
| COMPARE AND FORM CODEWORD | Equal | OCB=0: low<br>OCB=1: high | OCB=0: high<br>OCB=1: low | -- |
| COMPARE AND SIGNAL | Equal | Low | High | Unordered |
| COMPARE AND SWAP | Equal | Not equal | -- | -- |

*Figure C-1 (Part 1 of 6). Summary of Condition-Code Settings*

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-941

| Instruction | Condition Code | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| COMPARE AND SWAP AND PURGE | Equal | Not equal | -- | -- |
| COMPARE DECIMAL | Equal | Low | High | -- |
| COMPARE DOUBLE AND SWAP | Equal | Not equal | -- | -- |
| COMPARE HALFWORD | Equal | Low | High | -- |
| COMPARE HALFWORD IMMEDIATE | Equal | Low | High | -- |
| COMPARE LOGICAL | Equal | Low | High | -- |
| COMPARE LOGICAL CHARACTERS UNDER MASK | Equal | Low | High | -- |
| COMPARE LOGICAL LONG | Equal | Low | High | -- |
| COMPARE LOGICAL LONG EXTENDED | Equal | Low | High | CPU-determined completion |
| COMPARE LOGICAL LONG UNICODE | Equal | Low | High | CPU-determined completion |
| COMPARE LOGICAL STRING | Equal | Low | High | CPU-determined completion |
| COMPARE UNTIL SUBSTRING EQUAL | Equal substrings | Last bytes equal | Last bytes unequal | CPU-determined completion |
| COMPUTE INTERMEDIATE MESSAGE DIGEST | Normal completion | -- | -- | Partial completion |
| COMPUTE LAST MESSAGE DIGEST | Normal completion | -- | -- | Partial completion |
| COMPUTE MESSAGE AUTHENTICATION CODE | Normal completion | -- | -- | Partial completion |
| CONVERT BFP TO HFP | Zero | < zero | > zero | Special case |
| CONVERT HFP TO BFP | Zero | < zero | > zero | Special case |
| CONVERT TO FIXED | Zero | < zero | > zero | Special case |
| CONVERT UNICODE TO UTF-8 | Data processed | Op1 full | -- | CPU-determined completion |
| CONVERT UTF-8 TO UNICODE | Data processed | Op1 full | -- | CPU-determined completion |
| DIVIDE TO INTEGER | Remainder complete; normal quotient | Remainder complete; quotient overflow or NaN | Remainder incomplete; normal quotient | Remainder incomplete; quotient overflow or NaN |
| EDIT | Zero | < zero | > zero | -- |
| EDIT AND MARK | Zero | < zero | > zero | -- |
| EXCLUSIVE OR | Zero | Not zero | -- | -- |
| EXTRACT STACKED STATE | Branch state entry | Program-call state entry | -- | -- |

*Figure C-1 (Part 2 of 6). Summary of Condition-Code Settings*



LzLabs GmbH. Ex. 1008-942

| Instruction | Condition Code | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| HALT SUBCHANNEL | Function initiated | Status-pending with other than interme-diate status | Busy | Not operational |
| INSERT ADDRESS SPACE CONTROL | Primary-space mode | Secondary-space mode | Access-register mode | Home-space mode |
| INSERT CHARACTERS UNDER MASK | All zeros | First bit one | First bit zero | -- |
| LOAD ADDRESS SPACE PARAMETERS | Parameters loaded | Primary ASN not available | Secondary ASN not available or not authorized | Space-switch event |
| LOAD AND TEST (gen, HFP) | Zero | < zero | > zero | -- |
| LOAD AND TEST (BFP) | Zero | < zero | > zero | NaN |
| LOAD COMPLEMENT (gen) | Zero | < zero | > zero | Overflow |
| LOAD COMPLEMENT (BFP) | Zero | < zero | > zero | NaN |
| LOAD COMPLEMENT (HFP) | Zero | < zero | > zero | -- |
| LOAD NEGATIVE (gen, HFP) | Zero | < zero | -- | -- |
| LOAD NEGATIVE (BFP) | Zero | < zero | -- | NaN |
| LOAD POSITIVE (gen) | Zero | -- | > zero | Overflow |
| | available | invalid | invalid | not available or length violation |
| LOAD POSITIVE (BFP) | Zero | -- | > zero | NaN |
| LOAD POSITIVE (HFP) | Zero | -- | > zero | -- |
| LOAD REAL ADDRESS | Translation | ST entry | PT entry | ST designation |
| MODIFY SUBCHANNEL | SCHIB informa-tion placed in subchannel | Status-pending | Busy | Not operational |
| MOVE LONG | Length equal | Length low | Length high | Destructive overlap |
| MOVE LONG EXTENDED | Length equal | Length low | Length high | CPU-determined completion |
| MOVE LONG UNICODE | Length equal | Length low | Length high | CPU-determined completion |
| MOVE PAGE | Data moved | Operand 1 invalid, both valid in ES, locked, or ES error | Operand 2 invalid | -- |
| MOVE STRING | -- | Data moved | -- | CPU-determined completion |
| MOVE TO PRIMARY | Length =< 256 | -- | -- | Length > 256 |
| MOVE TO SECONDARY | Length =< 256 | -- | -- | Length > 256 |
| MOVE WITH KEY | Length =< 256 | -- | -- | Length > 256 |
| OR | Zero | Not zero | -- | -- |

*Figure C-1 (Part 3 of 6). Summary of Condition-Code Settings*


LzLabs GmbH. Ex. 1008-943

| Instruction | Condition Code | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| PAGE IN | Page-in operation completed | Expanded-storage data error | -- | Expanded-storage block not available |
| PAGE OUT | Page-out operation completed | Expanded-storage data error | -- | Expanded-storage block not available |
| PERFORM LOCKED OPERATION if test bit zero | Equal | Op1 not equal | Op1 equal, op3 not equal (dcs only) | -- |
| PERFORM LOCKED OPERATION if test bit one | Function code valid | -- | -- | Function code invalid |
| RESET CHANNEL PATH | Function initiated | -- | Busy | Not operational |
| RESET REFERENCE BIT EXTENDED | R bit zero, C bit zero | R bit zero, C bit one | R bit one, C bit zero | R bit one, C bit one |
| RESUME SUBCHANNEL | Function initiated | Status pending | Function not applicable | Not operational |
| SEARCH STRING | -- | Found | Not found | CPU-determined completion |
| SET CLOCK | Set | Secure | -- | Not operational |
| SHIFT AND ROUND DECIMAL | Zero | < zero | > zero | Overflow |
| SHIFT LEFT (DOUBLE/SINGLE) | Zero | < zero | > zero | Overflow |
| SHIFT RIGHT (DOUBLE/SINGLE) | Zero | < zero | > zero | -- |
| SIGNAL PROCESSOR | Order accepted | Status stored | Busy | Not operational |
| START SUBCHANNEL | Function initiated | Status-pending | Busy | Not operational |
| STORE CHANNEL REPORT WORD | CRW stored | Zeros stored | -- | -- |
| STORE CLOCK | Set | Not set | Error | Stopped or not operational |
| STORE CLOCK EXTENDED | Set | Not set | Error | Stopped or not operational |
| STORE SUBCHANNEL | SCHIB stored | -- | -- | Not operational |
| STORE SYSTEM INFORMATION | Information provided | -- | -- | Information not available |
| SUBTRACT (gen) | Zero | < zero | > zero | Overflow |
| SUBTRACT (BFP) | Zero | < zero | > zero | NaN |
| SUBTRACT DECIMAL | Zero | < zero | > zero | Overflow |
| SUBTRACT HALFWORD | Zero | < zero | > zero | Overflow |
| SUBTRACT LOGICAL | -- | Not zero, borrow | Zero, no borrow | Not zero, no borrow |
| SUBTRACT LOGICAL WITH BORROW | Zero, borrow | Not zero, borrow | Zero, no borrow | Not zero, no borrow |

*Figure C-1 (Part 4 of 6). Summary of Condition-Code Settings*



LzLabs GmbH. Ex. 1008-944

| Instruction | Condition Code | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| SUBTRACT NORMALIZED (HFP) | Zero | < zero | > zero | -- |
| SUBTRACT UNNORMALIZED (HFP) | Zero | < zero | > zero | -- |
| TEST ACCESS | ALET 0 | DU access list, no exceptions | PS access list, no exceptions | ALET 1 or exceptions |
| TEST ADDRESSING MODE | Twenty-four bit mode | Thirty-one bit mode | -- | -- |
| TEST AND SET | Left bit zero | Left bit one | -- | -- |
| TEST BLOCK | Usable | Not usable | -- | -- |
| TEST DATA CLASS | Zero (no match) | One (match) | -- | -- |
| TEST DECIMAL | Digits and sign valid | Sign invalid | Digit invalid | Sign and digit invalid |
| TEST PENDING INTERRUPTION | Interruption code not stored | Interruption code stored | -- | -- |
| TEST PROTECTION | Can fetch, can store | Can fetch, cannot store | Cannot fetch, cannot store | Translation not available |
| TEST SUBCHANNEL | IRB stored; subchannel status-pending | IRB stored; subchannel not status-pending | -- | Not operational |
| TEST UNDER MASK | All zeros | Mixed | -- | All ones |
| TEST UNDER MASK (HIGH/LOW) | All zeros | Mixed, left bit zero | Mixed, left bit one | All ones |
| TRANSLATE AND TEST | All zeros | Incomplete | Complete | -- |
| TRANSLATE EXTENDED | Data processed | Op1 byte equal test byte | -- | CPU-determined completion |
| TRANSLATE ONE TO ONE, ONE TO TWO, TWO TO ONE, TWO TO TWO | Character equal test character not found | Character equal test character found | -- | CPU-determined completion |
| UNPACK ASCII | Sign plus | Sign minus | -- | Sign invalid |
| UNPACK UNICODE | Sign plus | Sign minus | -- | Sign invalid |
| UPDATE TREE | Equal | Not equal or no comparison | -- | GR5 nonzero, GR0 negative |
| ZERO AND ADD | Zero | < zero | > zero | Overflow |

*Figure C-1 (Part 5 of 6). Summary of Condition-Code Settings*

**Explanation:**

```
 >  zero  Result greater than zero.
 <  zero  Result less than zero.
 =< 256   Equal to, or less than, 256.
 >  256   Greater than 256.
 gen      General instruction.
 BFP      Binary-floating-point instruction.
 High     First operand high.
 HFP      Hexadecimal-floating-point instruction.
 Low      First operand low.
 Length   Length of first operand.
 NaN      Not-a-number.
 OCB      Operand-control bit.
```

*Figure C-1 (Part 6 of 6). Summary of Condition-Code Settings*


LzLabs GmbH. Ex. 1008-945

LzLabs GmbH. Ex. 1008-946

# Appendix D. Comparison between ESA/370 and ESA/390

This appendix provides (1) a list of the facilities that are new in ESA/390 and not provided in ESA/370, and (2) a description of the handling in ESA/390 of the facilities available in ESA/370. This appendix applies to only the facilities that are described in detail in this publication. A summary of other facilities that are new in ESA/390 is in "Highlights of ESA/390" on page 1-1.

## New Facilities in ESA/390

The following facilities are new in ESA/390 and are not provided in ESA/370. Access-list-controlled protection is provided by all ESA/390 models. Concurrent sense, PER 2, storage-protection override, move-page facility 2, square root, string instruction, suppression on protection with virtual-address enhancement, set address space control fast, subspace group, called-space identification, checksum, compare and move extended, immediate and relative instruction, branch and set authority, perform locked operation, additional floating-point, program call fast, resume program, trap, extended TOD clock, TOD-clock-control override, store system information, extended translation 1, extended translation 2, z/Architecture (certain instructions), and enhanced input/output are provided by some ESA/390 models. A model provides move-page facility 1 if it does not provide move-page facility 2.

## Access-List-Controlled Protection

Bit 6 in the access-list entry is assigned as the fetch-only bit. If the fetch-only bit is one when a store-type storage reference is attempted using the access-list entry, a protection exception for access-list-controlled protection is recognized.

## Additional Floating-Point

"Additional floating-point" is an informal name referring to a set of four facilities related to hexadecimal floating point (HFP) and binary floating point (BFP). The four facilities are:

- **Basic floating-point extensions,** which includes:
  - Twelve additional floating-point (AFP) registers, to make a total of 16 floating-point registers.
  - A 32-bit floating-point-control (FPC) register.
  - An AFP-register-control bit, bit 13 of control register 0, which controls whether the new registers and the binary-floating-point instructions can be used.
  - The storing of a data-exception code (DXC) at real locations 144-147 during a program interruption for a data exception.
  - An extended-save-area control, bit 2 of control register 14, and an extended-save-area address at real and absolute locations 212-215. All 16 of the floating-point registers and the FPC register are saved in the extended save area during a store-status operation or a machine-check program interruption.
  - A new SIGNAL PROCESSOR order, store-extended-status-at-address, that performs the store-status-at-address operation and also saves the contents of the 16 floating-point registers and the FPC register.
- **Floating-point-support (FPS) extensions,** which provides eight new instructions, including four to convert data between the HFP and BFP formats.
- **Hexadecimal-floating-point (HFP) extensions,** which provides 26 new instructions to operate on data in the HFP format. All of these are counterparts to new instructions provided by the BFP facility, including conversion between floating-point and fixed-point formats, and a more complete set of operations on the extended format.
- **Binary floating-point (BFP),** which defines short, long, and extended BFP data formats and provides 87 new instructions to operate on data in these formats. The BFP formats and operations provide everything necessary to conform to the IEEE standard, except for binary-decimal conversion, which must be provided in software.

© Copyright IBM Corp. 1990-2003

LzLabs GmbH. Ex. 1008-947

The new floating-point instructions are listed in "Summary of All Floating-Point Instructions" on page 9-11.

## Additional Input/Output

"Additional input/output" in an informal name referring to a set of functions and facilities that are available on a model in the ESA/390 architectural mode when the z/Architecture architectural mode is installed. The functions and facilities are as follows:

- The ORB for a channel program can specify the use of format-2 IDAWs and either 2K-byte or 4K-byte data blocks.
- The FICON-channel facility provides the capability of attaching FICON-I/O-interface and FICON-converted-I/O-interface channel paths.
- The ORB-extension facility expands the size of the ORB from three words to eight words.
- The channel-subsystem-I/O-priority facility allows the program to establish a priority of operations at subchannels and at fibre-channel-attached control units.

The additional input/output functions and facilities are introduced more fully on page 1-6 in Chapter 1, "Introduction."

## Branch and Set Authority

When the BRANCH AND SET AUTHORITY (BSA) instruction is executed in the base authority state, bits 32-63 of the current PSW, including the updated instruction address, are saved in word 8 of the dispatchable-unit control table (DUCT), the PSW key mask (PKM), PSW key, and problem-state bit are saved in word 9 of the DUCT, and bit 28 in word 9 is set to one to indicate the reduced-authority state. The PKM and PSW key are replaced from general register $R_1$, PSW bits 32-63 are replaced from general register $R_2$, and the problem-state bit is set to one.

When BSA is executed in the reduced-authority state, bits 32-63 of the PSW and the PKM, PSW key, and problem-state bit are replaced with the values saved in the DUCT, and bit 28 in word 9 of the DUCT is set to zero to indicate the base-authority state.

## Called-Space Identification

Bytes 144-147 of the linkage-stack state entry formed by the stacking PROGRAM CALL instruction are assigned as the called-space identification (CSI). If the PROGRAM CALL operation was space switching, bytes 0 and 1 of the CSI contain the new primary ASN, and bytes 2 and 3 contain the rightmost two bytes of the ASTE sequence number in the new primary ASN-second-table entry. If the operation was the to-current-primary operation, the CSI is all zeros.

## Checksum

The CHECKSUM instruction computes a 32-bit checksum for a specified operand in storage. The program can easily use the 32-bit checksum to compute a 16-bit checksum if that is desired.

## Compare and Move Extended

The COMPARE LOGICAL LONG EXTENDED and MOVE LONG EXTENDED instructions are new versions of the COMPARE LOGICAL LONG and MOVE LONG instructions. The new versions increase the size of the operand-length specifications from 24 bits to 32 bits, and they periodically complete to allow software polling in a multiprocessing system.

## Concurrent Sense

When permitted by the program, the channel subsystem may retrieve sense data from the device when a unit-check condition is reported and provide the sense data to the program at the time of the interruption due to the unit-check condition. This avoids the need to obtain the sense information by means of a separate I/O operation. In particular, concurrent sense allows a control unit to be released more quickly from a contingent allegiance.

## Extended TOD Clock

The facility extends the TOD clock from 64 to 104 bits and provides the TOD programmable register, the privileged SET CLOCK PROGRAMMABLE FIELD instruction, and the STORE CLOCK EXTENDED instruction. Bits 16-31 of the 32-bit TOD programmable register are the TOD pro-



LzLabs GmbH. Ex. 1008-948

grammable field, which can be set by SET CLOCK PROGRAMMABLE FIELD. STORE CLOCK EXTENDED stores in a 16-byte storage operand; it stores zeros in bit positions 0-7 of the operand, bits 0-103 of the TOD clock in bit positions 8-111 of the operand, and the TOD programmable field in bit positions 112-127 of the operand. When the TOD clock is further extended in the future to have an additional leftmost byte (for when there is a carry from the current bit position 0 in the year 2042, if the standard epoch is used), STORE CLOCK EXTENDED will store that additional byte in bit positions 0-7 of its operand.

## Extended Translation 1

The facility provides the following instructions:

- CONVERT UNICODE TO UTF-8
- CONVERT UTF-8 TO UNICODE
- TRANSLATE EXTENDED

The conversion instructions convert between two-byte Unicode characters and one-to-four-byte UTF-8 characters. TRANSLATE EXTENDED can be used in place of a TRANSLATE AND TEST instruction that locates an escape character, followed by a TRANSLATE instruction that translates the bytes preceding the escape character.

## Extended Translation 2

The facility provides the following instructions:

- COMPARE LOGICAL LONG UNICODE
- MOVE LONG UNICODE
- PACK ASCII
- PACK UNICODE
- TEST DECIMAL
- TRANSLATE ONE TO ONE
- TRANSLATE ONE TO TWO
- TRANSLATE TWO TO ONE
- TRANSLATE TWO TO TWO
- UNPACK ASCII
- UNPACK UNICODE

The instructions perform operations on double-byte, ASCII, and decimal data. The double-byte data may be Unicode data — data that uses the binary codes of the Unicode Worldwide Character Standard and enables the use of characters of most of the world's written languages.

## Immediate and Relative Instruction

The facility provides the following instructions:

- ADD HALFWORD IMMEDIATE
- BRANCH RELATIVE AND SAVE
- BRANCH RELATIVE ON CONDITION
- BRANCH RELATIVE ON COUNT
- BRANCH RELATIVE ON INDEX HIGH
- BRANCH RELATIVE ON INDEX LOW OR EQUAL
- COMPARE HALFWORD IMMEDIATE
- LOAD HALFWORD IMMEDIATE
- MULTIPLY SINGLE (MS, MSR)
- MULTIPLY HALFWORD IMMEDIATE
- TEST UNDER MASK HIGH
- TEST UNDER MASK LOW

The instructions have new instruction formats named RI and RSI, except that MULTIPLY SINGLE has formats RRE and RX. The instructions with “IMMEDIATE” in their names use a 16-bit signed binary integer in an $I_2$ field. The TEST UNDER MASK HIGH/LOW instructions use a 16-bit mask in an $I_2$ field. MULTIPLY SINGLE and MULTIPLY HALFWORD IMMEDIATE return only the rightmost 32 bits of the product. The branch instructions have an $I_2$ field whose contents are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address. The branch instructions allow branching to a location at an offset of up to plus 64K - 2 bytes or minus 64K bytes relative to the location of the branch instruction.

## Move-Page Facility 2

The MOVE PAGE instruction moves a page of data from main storage to main storage or expanded storage or from expanded storage to main storage. An invalid page is indicated by a page-translation exception if the condition-code-option bit in general register 0 is zero, or it is indicated by a setting of the condition code if the condition-code-option bit is one. General register 0 also contains (1) a destination-reference-intention bit that causes a page-translation-exception condition instead of movement to expanded storage, and (2) an access key that can be specified to apply to either the source operand or the destination operand. The definition of MOVE PAGE of move-page facility 2 is in


LzLabs GmbH. Ex. 1008-949

Chapter 10, “Control Instructions.” MOVE PAGE of move-page facility 1 was introduced in ESA/370 and is on some ESA/390 models, and its definition is in Chapter 7, “General Instructions.”

## PER 2

Bit 8 in control register 9 is assigned as the branch-address control, and bit 10 is assigned as the storage-alteration-space control. The branch-address control specifies, when one, that a successful-branching event is to occur only if the branch-target location is within the storage area designated by means of control registers 10 and 11 (the designated storage area). The storage-alteration-space control specifies, when one, that a storage-alteration event is to occur only for a reference to the designated storage area within designated address spaces. Bit 24 in the segment-table designation is assigned as the storage-alteration-event bit. When this bit is one, storage-alteration events are to occur in the address space specified by the segment-table designation. Monitoring for general-register-alteration events is omitted.

Bit 4 in real locations 150-151 is assigned to indicate a store-using-real-address event. A store-using-real-address event is indicated when bits 2 and 4 in locations 150-151 are both ones. Bits 9-13 of locations 150-151 are assigned as the addressing-and-translation-mode identification (ATMID). The ATMID indicates the values of PSW bits 32, 5, 16, and 17 at the beginning of execution of any instruction that causes a PER event and changes any of PSW bits 5, 16, and 17. Bits 14 and 15 of locations 150-151 are assigned as the PER STD identification, which identifies the segment-table designation (STD) that was used to translate a reference that causes a storage-alteration event. The PER access identification at real location 161 is predictable even if the instruction that caused the storage-alteration event turned DAT off.

## Perform Locked Operation

The PERFORM LOCKED OPERATION instruction uses a program-lock-token (PLT) logical address as a designator of a lock internal to the configuration. A PLT is a value produced by a model-dependent transformation of the PLT logical address. Programs being executed by different CPUs can be assured of specifying the same lock only by specifying PLT logical addresses that are the same and that can be transformed to the same real address by the different CPUs. After obtaining the lock selected by the PLT, the instruction performs any of six operations specified by a function code: compare and load, compare and swap, double compare and swap, compare and swap and store, compare and swap and double store, and compare and swap and triple store. The function code further specifies word or doubleword operands. All operations on multiple storage operands by a PERFORM LOCKED OPERATION instruction appear to occur entirely either before or after all operations on the same operands by another PERFORM LOCKED OPERATION instruction executed by another CPU, provided that both of the instructions use the same lock.

## Program Call Fast

The PROGRAM CALL FAST instruction has the same op code as PROGRAM CALL. When the program-call-fast control, bit 28 of control register 0, is one and bits 12-23 (the linkage-index part) of the second-operand address (the PC number) have the value 31 (01F hex), either the PROGRAM CALL FAST definition or the PROGRAM CALL definition applies. When either of those two conditions is not met, the PROGRAM CALL definition applies. PROGRAM CALL FAST uses bits 25-31 of the second-operand address as an entry index to select an entry in the PCF entry table whose real origin is at real locations 196-199. PROGRAM CALL FAST forms a linkage-stack entry the same as stacking PROGRAM CALL, except that the called-space identification for PROGRAM CALL FAST is always all zeros. PROGRAM CALL FAST with space switching obtains the new primary segment-table designation directly from the PCF-entry-table entry.

## Resume Program

The second-operand address of the RESUME PROGRAM instruction is formed by means of the $B_2$ and $D_2$ field of the instruction and designates a save area, which is intended to contain the access-register and general-register contents and certain PSW fields of an interrupted program. The PSW fields are the address space control, condi-



LzLabs GmbH. Ex. 1008-950

tion code, program mask, addressing mode, and instruction address. A problem-state interruption-handling program can return to the interrupted program by first restoring the contents of all registers except for one access-and-general register pair and then issuing a RESUME PROGRAM instruction whose $B_2$ field designates that remaining pair. RESUME PROGRAM restores the contents of the $B_2$ access register and general register and the PSW fields from the save area. A parameter list that immediately follows the RESUME PROGRAM instruction contains the offsets in the save area of the fields to be restored.

## Set Address Space Control Fast

The SET ADDRESS SPACE CONTROL FAST (SACF) instruction performs the functions of the SET ADDRESS SPACE CONTROL (SAC) instruction, except that SACF does not perform serialization or checkpoint synchronization or cause prefetched instructions to be discarded.

## Square Root

The SQUARE ROOT instruction (SQDR and SQER) extracts the square root of a floating-point operand in either the long (SQDR) or the short (SQER) format. Program-interruption code 001D hex is assigned to the square-root exception, which is recognized if the input operand is less than zero.

## Storage-Protection Override

Bit 7 of control register 0 is assigned as the storage-protection-override control. When bit 7 is one, key-controlled protection is ignored for references by the CPU to storage locations having an associated storage-key value of 9.

## Store System Information

The facility consists of the privileged STORE SYSTEM INFORMATION instruction, which can obtain information about any of three levels of configuration at or below the level that is executing the program: level 1, the basic machine; level 2, a logical partition; and level 3, a virtual machine. A function code in general register 0 specifies whether the current-level number is to be provided by the instruction or whether information about a specified level is to be provided. In the latter case, values, called selectors, in general registers 0 and 1 specify the information to be provided.

## String Instruction

The MOVE STRING instruction moves a string of bytes from a source location to a destination location until an ending character (one byte) specified in a general register has been moved. The COMPARE LOGICAL STRING instruction compares two byte strings until an ending character specified in a general register is found in either string or an inequality is found. The SEARCH STRING instruction searches a byte string of a specified length until a character specified in a general register is found. MOVE STRING and COMPARE LOGICAL STRING are particularly useful in a C-programming-language program in which strings are normally delimited by an all-zeros byte.

## Subspace Group

A subspace group is a group of address spaces consisting of a base space and subspaces. A dispatchable unit can use the BRANCH IN SUBSPACE GROUP (BSG) instruction to transfer control within a subspace group that is associated with the dispatchable unit. The following fields are assigned:

- Bit 22 of the segment-table designation (STD), the subspace-group-control bit (G), indicates, when one, that the address space specified by the STD is a base space or a subspace.
- Bit 31 of the ASN-second-table entry (ASTE), the base-space bit (B), indicates, when one, that the address space specified by the ASTE is a base space.
- Bits 1-25 of word 0 of the dispatchable-unit control table (DUCT), the base-ASTE origin (BASTEO), designate the ASTE for the base space of a subspace group associated with the dispatchable unit that is represented by the DUCT.
- Bit 0 of word 1 of the DUCT, the subspace-active bit (SA), indicates, when zero, that the dispatchable unit either last used BSG to transfer control to its base space or has not


LzLabs GmbH. Ex. 1008-951

used BSG at all, or, when one, that the dispatchable unit last used BSG to transfer control to a subspace.

- Bits 1-25 of word 1 of the DUCT, the subspace-ASTE origin (SSASTEO), designate the ASTE for the subspace that was last transferred to by means of BSG.
- Bits 0-31 of word 3 of the DUCT, the subspace-ASTE sequence number (SSASTESN), are a copy of the ASTE seqence number that was in the ASTE for the subspace that was last transferred to by means of BSG.

BSG uses an access-list-entry token (ALET) as a specification of its destination address space. The subspaces of a dispatchable unit's subspace group are designated by entries on the dispatchable-unit access list. ALET 0 designates the base space, and ALET 1 designates the last entered subspace.

The following instructions are modified to perform operations called subspace-replacement operations: LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, and SET SECONDARY ASN. When one of the named instructions establishes a new primary or secondary STD that designates the base space of the current dispatchable unit when the dispatchable unit is subspace active, the instruction replaces bits 1-23 and 25-31 of the STD with the corresponding bits of the STD in the subspace ASTE.

A branch trace entry is made for BSG if branch tracing is on and ASN tracing is off, or a BSG trace entry is made if ASN tracing is on.

## Suppression on Protection

During a program interruption due to a protection exception, bit 29 of real locations 144-147 is set to zero or one. If it is set to one, the instruction execution during which the exception was recognized was suppressed, and the protected location is identified in other bit positions of locations 144-147 and in real location 160. If bit 29 is zero, the instruction execution may have been terminated, and the contents of those other bit positions and of location 160 are unpredictable. Bit 29 is set to one if the protection exception is due to access-list-controlled protection or page protection. Bit 29 may be set to one if the protection exception is due to key-controlled protection or low-address protection. If the virtual-address enchancement of suppression on protection is installed, bit 29 is set to one when DAT was on only if the address stored is one that was to be translated by DAT.

## TOD-Clock-Control Override

The facility consists of the TOD-clock-control-override control, bit 10 of control register 14. When this bit is one, the TOD clock can be set by the SET CLOCK instruction regardless of the settings of the manual TOD-clock controls in the configuration.

## Trap

The TRAP (TRAP2 and TRAP4) instruction can overlay instructions in a program to give control to a trap program that can simulate the overlaid instructions and perform fix-up operations on data being processed. TRAP2 is for overlaying a two-byte instruction, and TRAP4 is for overlaying a four-byte instruction or the first four bytes of a six byte instruction. TRAP uses a trap-control-block address and TRAP-enabled bit in bytes 44-47 of the dispatchable-unit control table. The trap control block designates a trap save area and a trap program. The trap control block and trap save area are in the home address space. The trap program is in the primary address space.

## z/Architecture

The following additional general instructions are available in the ESA/390 architectural mode when the z/Architecture architectural mode is installed:

- ADD LOGICAL WITH CARRY (ALC, ALCR)
- BRANCH RELATIVE AND SAVE LONG
- BRANCH RELATIVE ON CONDITION LONG
- DIVIDE LOGICAL (DL, DLR)
- EXTRACT PSW
- LOAD ADDRESS RELATIVE LONG
- LOAD REVERSED (LRV, LRVH, LRVR)
- MULTIPLY LOGICAL (ML, MLR)
- ROTATE LEFT SINGLE LOGICAL
- SET ADDRESSING MODE (SAM24, SAM31)
- STORE REVERSED (STRV, STRVH)
- SUBTRACT LOGICAL WITH BORROW (SLB, SLBR)



LzLabs GmbH. Ex. 1008-952

- TEST ADDRESSING MODE

The privileged STORE FACILITY LIST instruction is also available.

The operations of the instructions and a new SIGNAL PROCESSOR order for switching between architectural modes are introduced on page 1-5 in Chapter 1, “Introduction.”

## Comparison of Facilities

Figure D-1 shows the facilities offered in ESA/370 and how each facility is provided in ESA/390.

| ESA/370 Facility | Availa-bility in ESA/390 |
|---|---|
| Basic ESA/370 facilities | B |
| Compare until substring equal | B |
| Expanded storage | ES |
| Move inverse | B |
| Move page | B[1] |
| Private space | B |
| Vector | V |

**Explanation:**

[1] Either the move-page facility 1 or the move-page facility 2 is basic in ESA/390 mode.
B Basic in ESA/390 mode.
ES Provided in both ESA/370 and ESA/390 as the expanded-storage facility.
V Provided in both ESA/370 and ESA/390 as the vector facility.

*Figure D-1. Availability of ESA/370 Facilities in ESA/390*


LzLabs GmbH. Ex. 1008-953

LzLabs GmbH. Ex. 1008-954

# Appendix E. Comparison between 370-XA and ESA/370


New Facilities in ESA/370 . . . . . E-1
Access Registers . . . . . E-1
Compare until Substring Equal . . . . . E-1
Home Address Space . . . . . E-1
Linkage Stack . . . . . E-2
Load and Store Using Real Address . . . E-2
Move Page Facility 1 . . . . . E-2
Move with Source or Destination Key . . . E-2
Private Space . . . . . E-2
Comparison of Facilities . . . . . E-2
Summary of Changes . . . . . E-2
New Instructions Provided . . . . . E-2
Comparison of PSW Formats . . . . . E-3
New Control-Register Assignments . . . . E-3
New Assigned Storage Locations . . . . . E-3
New Exceptions . . . . . E-4
Change to Secondary-Space Mode . . . . E-4
Changes to ASN-Second-Table Entry and ASN Translation . . . . . E-4
Changes to Entry-Table Entry and PC-Number Translation . . . . . E-5
Changes to PROGRAM CALL . . . . . E-5
Changes to SET ADDRESS SPACE CONTROL . . . . . E-5
Effects in New Translation Modes . . . . . E-5
Effects on Interlocks for Virtual-Storage References . . . . . E-5
Effect on INSERT ADDRESS SPACE CONTROL . . . . . E-6
Effect on LOAD REAL ADDRESS . . . . . E-6
Effect on TEST PENDING INTERRUPTION . . . . . E-6
Effect on TEST PROTECTION . . . . . E-6


This appendix provides (1) a list of the facilities that are new in ESA/370 and not provided in 370-XA, (2) a description of the handling in ESA/370 of the facilities available in 370-XA, (3) a list of changes between 370-XA and ESA/370, and (4) a list of how 370-XA facilities are affected by the new translation modes in ESA/370.

## New Facilities in ESA/370

The following facilities are new in ESA/370 and are not provided in 370-XA. Access registers, home address space, linkage stack, load and store using real address, and move with source or destination key are provided by all ESA/370 models. Compare until substring equal, move page, and private space are provided by some ESA/370 models.

## Access Registers

Sixteen access registers and a translation mode named the access-register mode allow designation of storage operands in up to sixteen different address spaces by means of the B fields of instructions and the R fields of certain instructions. The dispatchable-unit and primary-space access lists contain the addressing capabilities that are usable by means of the access registers. The use of an access-list entry is controlled by the extended authorization index in control register 8. Instructions are provided for examining and changing the contents of the access registers and for purging the access-register-translation-lookaside buffer.

## Compare until Substring Equal

An instruction is provided for comparing two byte strings until equal substrings of a specified length are found or the end of the longer operand is reached.

## Home Address Space

A translation mode named the home-space mode allows the control program to quickly gain control in and access the home address space, which is where the control program keeps the principal control blocks for a dispatchable unit. The space-switch event can indicate a transfer of control to or from the home address space.

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-955

## Linkage Stack

A bit in the entry-table entry controls whether PROGRAM CALL performs the 370-XA, or basic, operation or the stacking operation. The stacking operation allows increased status changing, and it saves status in a linkage-stack state entry, from which status is restored by the PROGRAM RETURN instruction. The linkage stack can also be used in a branch-type linkage. Instructions are provided for examining and changing the contents of the last state entry and for testing the contents of an access register by means of a specified extended authorization index.

## Load and Store Using Real Address

Instructions are provided for loading and storing from a general register through the use of a real address. The storing operation can be indicated by a store-using-real-address PER event.

## Move Page Facility 1

The MOVE PAGE instruction moves a page of data from main storage to main storage or expanded storage or from expanded storage to main storage. An invalid page is indicated by a setting of the condition code.

## Move with Source or Destination Key

Instructions are provided for moving data with a specified access key that applies to the references to either the source or the destination storage area; the PSW key applies to the references to the other storage area.

## Private Space

A bit in the segment-table designation can be set to one to prevent the use of translation-lookaside-buffer entries for common segments and to prevent the application of low-address protection and fetch-protection override to the specified address space.

# Comparison of Facilities

Figure E-1 shows the facilities offered in 370-XA and how each facility is provided in ESA/370.

| 370-XA Facility | Availability in ESA/370 |
|---|---|
| Basic 370-XA facilities | B[1] |
| Expanded storage | ES |
| Move inverse | MI |
| Vector | V |

**Explanation:**

[1] Compatibility for privileged programs is not provided when the address-space-function control, bit 15 of control register 0, is one.
B Basic in ESA/370 mode.
ES Provided in both 370-XA and ESA/370 as the expanded-storage facility.
MI Provided in both 370-XA and ESA/370 as the move-inverse facility.
V Provided in both 370-XA and ESA/370 as the vector facility.

*Figure E-1. Availability of 370-XA Facilities in ESA/370*

# Summary of Changes

This section summarizes the changes between 370-XA and ESA/370. Most of these changes are simply additions in ESA/370 beyond 370-XA or apply only when the ESA/370 address-space-function (ASF) control, bit 15 of control register 0, is one. Some of the changes apply regardless of the value of the ASF control.

## New Instructions Provided

Figure E-2 on page E-3 shows those instructions which are basic in ESA/370 but not provided in 370-XA. All 370-XA instructions are provided in ESA/370.



LzLabs GmbH. Ex. 1008-956

| Instruction Name | Mne-monic | Op Code | Availa-bility |
|---|---|---|---|
| BRANCH AND STACK | BAKR | B240 | B[1] |
| COMPARE UNTIL SUBSTRING EQUAL | CUSE | B257 | CU |
| COPY ACCESS | CPYA | B24D | B |
| EXTRACT ACCESS | EAR | B24F | B |
| EXTRACT STACKED REGISTERS | EREG | B249 | B[1] |
| EXTRACT STACKED STATE | ESTA | B24A | B[1] |
| LOAD ACCESS MULTIPLE | LAM | 9A | B |
| LOAD ADDRESS EXTENDED | LAE | 51 | B |
| LOAD USING REAL ADDRESS | LURA | B24B | B |
| MODIFY STACKED STATE | MSTA | B247 | B[1] |
| MOVE PAGE | MVPG | B254 | M1 |
| MOVE WITH DESTINATION KEY | MVCDK | E50F | B |
| MOVE WITH SOURCE KEY | MVCSK | E50E | B |
| PROGRAM RETURN | PR | 0101 | B[1] |
| PURGE ALB | PALB | B248 | B |
| SET ACCESS | SAR | B24E | B |
| STORE ACCESS MULTIPLE | STAM | 9B | B |
| STORE USING REAL ADDRESS | STURA | B246 | B |
| TEST ACCESS | TAR | B24C | B[1] |

**Explanation:**

[1] Instruction can be executed successfully only when the address-space-function control, bit 15 of control register 0, is one.
B Instruction is basic.
CU Compare-until-substring-equal facility.
M1 Move-page facility 1.

*Figure E-2. New Instructions Provided*

## Comparison of PSW Formats

In 370-XA, PSW bit 16 is the address-space control, and a one in bit position 17 of the PSW is invalid. In ESA/370, PSW bits 16 and 17 are the address-space control.

## New Control-Register Assignments

Figure E-3 shows those assignments of control-register bits and fields that are new in ESA/370 compared to 370-XA.

| Ctrl Reg | Bits | Name of Bit or Field |
|---|---|---|
| 0 | 15 | Address-space-function control |
| 1 | 0 | Primary space-switch-event control[1] |
| 1 | 23 | Primary private-space control |
| 2 | 1-25 | Dispatchable-unit-control-table origin |
| 5 | 1-25 | Primary-ASN-second-table-entry origin[2] |
| 7 | 23 | Secondary private-space control |
| 8 | 0-15 | Extended authorization index |
| 9 | 4 | Store-using-real-address-event mask |
| 13 | 0 | Home space-switch-event control |
| 13 | 1-19 | Home segment-table origin |
| 13 | 23 | Home private-space control |
| 13 | 25-31 | Home segment-table length |
| 15 | 1-28 | Linkage-stack-entry address |

**Explanation:**

[1] Only the name of this bit is new. The bit has the same position and function as the space-switch-event control of 370-XA.
[2] This assignment applies only if bit 15 of control register 0 is one. If bit 15 is zero, control register 5 contains the linkage-table designation as in 370-XA.

*Figure E-3. New Control-Register Assignments*

In 370-XA, and in ESA/370 when the address-space-function (ASF) control, bit 15 of control register 0, is zero, control register 5 contains the linkage-table designation. In ESA/370 when the ASF control is one, control register 5 contains the primary ASN-second-table-entry origin, and the linkage-table designation is in the primary ASN-second-table entry.

## New Assigned Storage Locations

Figure E-4 on page E-4 shows those storage locations that are assigned in ESA/370 and not assigned in 370-XA.


LzLabs GmbH. Ex. 1008-957

| Name of Field | Assigned Storage Location and Length* |
|---|---|
| Exception access identification | R 160 1 |
| PER access identification | R 161 1 |
| Machine-check access-register save area | R 288 64 |
| Store-status access-register save area | A 288 64 |

**Explanation:**

* The first number is the address, the second the length.
A Absolute location.
R Real location.

*Figure E-4. New Assigned Storage Locations*

Bit 33 of the machine-check-interruption code, the access-register-validity bit, is assigned in ESA/370 and not assigned in 370-XA.

In both 370-XA and ESA/370, the translation-exception identification is stored at real locations 144-147 during a program interruption due to a segment-translation or page-translation exception. In 370-XA, bits 20-31 of this translation-exception identification are unpredictable. In ESA/370, bits 20-29 are unpredictable, and bits 30-31 are set to identify the type of virtual address that caused the exception.

## New Exceptions

Figure E-5 shows those new exceptions that may be recognized in ESA/370 and are not recognized in 370-XA.

| Exception Name | Interruption Code (hex) |
|---|---|
| ALET specification[1] | 0028 |
| ALEN translation[1] | 0029 |
| ALE sequence[1] | 002A |
| ASTE validity[1] | 002B |
| ASTE sequence[1] | 002C |
| Extended authority[1] | 002D |
| Stack full[2] | 0030 |
| Stack empty[2] | 0031 |
| Stack specification[2] | 0032 |
| Stack type[2] | 0033 |
| Stack operation[2] | 0034 |

**Explanation:**

[1] May be recognized during access-register translation.
[2] May be recognized during linkage-stack operations.

*Figure E-5. New Exceptions*

## Change to Secondary-Space Mode

In 370-XA in the secondary-space mode, it is unpredictable whether instructions are fetched from the primary address space or the secondary address space. In ESA/370 in the secondary-space mode, instructions are fetched from the primary address space.

## Changes to ASN-Second-Table Entry and ASN Translation

In 370-XA, and in ESA/370 when the address-space-function (ASF) control, bit 15 of control register 0 is zero, the ASN-second-table entry has a length of 16 bytes and is aligned on a 16-byte boundary. In ESA/370 when the ASF control is one, the ASN-second-table entry has a length of 64 bytes and is aligned on a 64-byte boundary. ASN translation is affected by this change.



LzLabs GmbH. Ex. 1008-958

## Changes to Entry-Table Entry and PC-Number Translation

In 370-XA, and in ESA/370 when the address-space-function (ASF) control, bit 15 of control register 0 is zero, the entry-table entry has a length of 16 bytes. In ESA/370 when the ASF control is one, the entry-table entry has a length of 32 bytes. PC-number translation is affected by this change and also by the change to the location of the linkage-table designation described in "New Control Register Assignments" in this appendix.

## Changes to PROGRAM CALL

In 370-XA, and in ESA/370 when the address-space-function (ASF) control, bit 15 of control register 0 is zero, a space-switching PROGRAM CALL obtains the address of the ASN-second-table entry for the new primary address space by means of ASN translation. In ESA/370 when the ASF control is one, PROGRAM CALL obtains the address of the ASN-second-table entry either by means of ASN translation or directly from the entry-table entry, and which of these occurs is unpredictable.

In ESA/370 when the ASF control is zero or when the ASF control is one and the PC-type bit, bit 128 of the 32-byte entry-table entry, is zero, PROGRAM CALL performs the 370-XA operation, called the basic operation. In ESA/370 when both the ASF control and the PC-type bit are ones, PROGRAM CALL performs a different operation, called the stacking operation.

## Changes to SET ADDRESS SPACE CONTROL

In 370-XA, for SET ADDRESS SPACE CONTROL, bit 22 of the second-operand address must be zero; otherwise, a specification exception is recognized. In ESA/370, bit 22 may be one in order to specify the setting of either the access-register mode or the home-space mode, depending on bit 23.

## Effects in New Translation Modes

ESA/370 has two new translation modes named the access-register mode and the home-space mode. These modes result when DAT is on and PSW bits 16 and 17 are 01 or 11 binary, respectively. This section summarizes the effects of the new translation modes on operations that would otherwise be the same as in 370-XA. For LOAD REAL ADDRESS, the effect applies whether DAT is on or off.

## Effects on Interlocks for Virtual-Storage References

In 370-XA and ESA/370, in the real mode, primary-space mode, or secondary-space mode, when a store is made to a location from which a succeeding instruction is fetched and the same effective address is used for both the store and the fetch, the results of the store appear to be completed before the fetch. Thus, it is possible for an instruction to modify the next succeeding instruction in storage. In ESA/370, in the access-register mode or home-space mode, an instruction that is a store-type operand of a preceding instruction may appear to be fetched before the store occurs. Thus, it is not assured that an instruction can modify the succeeding instruction.

In 370-XA and ESA/370, for those instructions which alter the contents of storage and have more than one operand, the instruction definition normally describes the results that are obtained when the operands overlap in storage. In 370-XA, and in ESA/370 in other than the access-register mode, operand overlap is recognized if the effective addresses of the two operands are the same. In ESA/370, in the access-register mode, recognition of operand overlap additionally requires that the effective space designations of the two operands be the same. The effective space designation for an operand is the contents of the access register used to access the operand, except that, if access register 0 is used, the contents are treated as being all zeros.


LzLabs GmbH. Ex. 1008-959

## Effect on INSERT ADDRESS SPACE CONTROL

In 370-XA, INSERT ADDRESS SPACE CONTROL sets bit 22 of general register $R_1$ to zero, and it sets the condition code to 0 or 1. In ESA/370, because of the new translation modes, INSERT ADDRESS SPACE CONTROL may set bit 22 to one, and it may set the condition code to 2 or 3.

## Effect on LOAD REAL ADDRESS

In 370-XA, when LOAD REAL ADDRESS sets any of condition codes 1-3, indicating an exception situation, it places an address related to the situation in general register $R_1$, and it sets bit 0 of the register to zero. Condition code 3 indicates that the segment-table or page-table length is exceeded. In ESA/370, when PSW bits 16 and 17 are 01 binary, condition code 3 may alternatively indicate an exception situation encountered during access-register translation, in which case the interruption code assigned to the exception is placed in general register $R_1$, and bit 0 of the register is set to one.

## Effect on TEST PENDING INTERRUPTION

In 370-XA and ESA/370, a zero second-operand address of TEST PENDING INTERRUPTION specifies a store at real locations 184-191. In this case, in ESA/370 in the access-register mode, it is unpredictable whether access-register translation occurs for the access register designated by the $B_2$ field. If access-register translation occurs and the access register is in error, an exception is recognized. If the translation occurs and there is no exception, the resulting segment-table designation is not used; that is, the store still occurs at real locations 184-191.

## Effect on TEST PROTECTION

In 370-XA, TEST PROTECTION sets condition code 3 if it encounters an exception situation during dynamic address translation. In ESA/370 in the access-register mode, TEST PROTECTION may alternatively set condition code 3 because of an exception situation encountered during access-register translation.



LzLabs GmbH. Ex. 1008-960

# Appendix F. Comparison between System/370 and 370-XA


New Facilities in 370-XA . . . . . F-1
Bimodal Addressing . . . . . F-1
31-Bit Logical Addressing . . . . . F-1
31-Bit Real and Absolute Addressing . . . F-1
Page Protection . . . . . F-2
Tracing . . . . . F-2
Incorrect-Length-Indication Suppression . F-2
Status Verification . . . . . F-2
Comparison of Facilities . . . . . F-2
Summary of Changes . . . . . F-3
Changes in Instructions Provided . . . . . F-3
Input/Output Comparison . . . . . F-5
Comparison of PSW Formats . . . . . F-5
Changes in Control-Register Assignments . F-6
Changes in Assigned Storage Locations . F-6
Changes to SIGNAL PROCESSOR . . . . F-6
Machine-Check Changes . . . . . F-7
Changes to Addressing Wraparound . . . F-7
Changes to LOAD REAL ADDRESS . . . F-7
Changes to 31-Bit Real Operand Addresses . . . . . F-8


This appendix provides (1) a list of the facilities that are new in 370-XA and not provided in System/370, (2) a description of the handling in 370-XA of the facilities available in System/370, and (3) a list of changes between System/370 and 370-XA.

## New Facilities in 370-XA

The following facilities are new in 370-XA and are not provided in System/370.

## Bimodal Addressing

Two modes of operation are provided: a 24-bit addressing mode, for the execution of old programs, and a 31-bit addressing mode. The mode is controlled by bit 32 in the PSW, and unprivileged instructions are provided that examine and set the mode. These instructions conveniently permit combining old programs, which must operate in the 24-bit addressing mode, and new programs which can take advantage of the 31-bit addressing mode.

## 31-Bit Logical Addressing

The 31-bit logical addressing includes the ability to perform either 24-bit or 31-bit address arithmetic for operand address generation and includes extensions to the following addresses, which are always 31 bits, regardless of the addressing mode:

- Instruction address in PSW bits 33-63
- PER starting address in control register 10
- PER ending address in control register 11
- Translation-exception identification stored at real locations 144-147
- PER address stored at real locations 152-155
- Monitor code stored at real locations 156-159
- Entry instruction address in the entry-table entry

## 31-Bit Real and Absolute Addressing

The following fields provide the leftmost part of 31-bit addresses, or the entire address, as appropriate, regardless of the setting of the addressing mode. Except where indicated, the addresses are real.

- Prefix register (absolute)
- Primary segment-table origin* in control register 1
- Linkage-table origin in control register 5
- Secondary segment-table origin* in control register 7
- ASN-first-table origin in control register 14
- Page-table origin in the segment-table entry
- Page-frame real address in the page-table entry
- ASN-second-table origin in the AFT entry
- Segment-table origin*, linkage-table origin, and authority-table origin in the AST entry
- Entry-table origin in the linkage-table entry
- Address in format-1 CCWs (absolute)

*Unpredictable whether address is real or absolute

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-961

## Page Protection

A page-protection bit is provided in the page-table entry. Page protection can be used in a manner similar to the System/370 segment protection, which is not included in 370-XA.

## Tracing

Included are a trace-table origin, branch trace control, ASN trace control, and explicit trace-control bits in control register 12. Also included are the instruction TRACE and a new program-interruption condition called trace-table exception. When branch tracing is on, a trace entry is made for the successful execution of the following instructions:

- BRANCH AND LINK (BALR) when the $R_2$ field is nonzero
- BRANCH AND SAVE (BASR) when the $R_2$ field is nonzero
- BRANCH AND SAVE AND SET MODE (BASSM) when the $R_2$ field is nonzero

When ASN tracing is on, an entry is made in the trace table for each execution of the following instructions:

- PROGRAM CALL
- PROGRAM TRANSFER
- SET SECONDARY ASN

When explicit tracing is on, execution of TRACE causes a trace entry to be made.

## Incorrect-Length-Indication Suppression

The incorrect-length-indication-suppression facility allows the indication of incorrect length to be suppressed when using format-1 CCWs in the same manner as when using format-0 CCWs or System/370 CCWs. Bit 24 of word 1 of the ORB provides the capability of indicating or suppressing recognition of incorrect length for an immediate operation.

## Status Verification

The status-verification facility provides an indication (bit 26 of the subchannel logout in the extended-status word) when the channel subsystem detects device status with a combination of bits that was inappropriate at the time status was presented.

---

## Comparison of Facilities

Figure F-1 on page F-3 shows the facilities offered in System/370 and whether or not each facility is provided in 370-XA.



LzLabs GmbH. Ex. 1008-962

| System/370 Facility | Availability in 370-XA |
|---|---|
| Commercial instruction set | P[1] |
| Block-multiplexer channels | F |
| Branch and save | B |
| Byte-multiplexer channels | F |
| Channel indirect data addressing | B |
| Channel-set switching | F |
| Clear I/O | F |
| Command retry | B |
| Conditional swapping | B |
| CPU timer and clock comparator | B |
| Direct control | - |
| Dual address space | P[2] |
| Expanded storage | ES |
| Extended | P[3] |
| Extended-precision floating point | B |
| Extended real addressing | R[4] |
| External signals | - |
| Fast release | F |
| Floating point | B |
| Halt device | F |
| I/O extended logout | - |
| Limited channel logout | F |
| Move inverse | MI |
| Multiprocessing | B[5] |
| PSW-key handling | B |
| Recovery extensions | - |
| Segment protection | R[6] |
| Selector channels | F |
| Service signal | B |
| Start-I/O-fast queuing | F |
| Storage-key-instruction extensions | B |
| Storage-key 4K-byte block | P[7] |
| Suspend and resume | F |
| Test block | B |
| Translation | P[8] |
| Vector | V |
| 31-bit IDAWs | B |

*Figure F-1 (Part 1 of 2). Availability of System/370 Facilities in 370-XA*

**Explanation:**

- Not provided in 370-XA.
1 The following items, which are part of the basic computing function in System/370, are not provided in 370-XA: BC mode, interval timer, and 2K-byte protection blocks. Also see the following instructions lists for those instructions basic in System/370 which are not provided in 370-XA.
2 All of the dual-address-space facility is provided except for DAS tracing.
3 See the following instruction list for those instructions that are part of the System/370 extended facility and that are provided in 370-XA.
4 Replaced with 31-bit real addressing.
5 With the exception of the inclusion of more than one CPU, all the functions associated with the System/370 multiprocessing facility are basic.
6 Replaced by page protection.
7 Only single-key 4K-byte protection blocks are provided, but the storage-key-exception control is not.
8 The 370-XA translation provides only the 4K-byte page size and only the 1M-byte segment size. See also the following instruction lists.
B Basic in 370-XA.
ES Provided in both System/370 and 370-XA as the expanded-storage facility.
F Not provided, but a comparable function is provided by the channel subsystem.
MI Provided in both System/370 and 370-XA as the move-inverse facility.
P Partially available in 370-XA.
R Replaced with a comparable facility.
V Provided in both System/370 and 370-XA as the vector facility.

*Figure F-1 (Part 2 of 2). Availability of System/370 Facilities in 370-XA*

# Summary of Changes

## Changes in Instructions Provided

The following figures show those instructions which are optional or not provided in either System/370 or 370-XA. Those instructions which are basic in both System/370 and 370-XA are not shown.



LzLabs GmbH. Ex. 1008-963

| Instruction Name* | Mne-monic | Op Code | System/370 | 370-XA |
|---|---|---|---|---|
| BRANCH AND SAVE | BASR | 0D | BS | B |
| BRANCH AND SAVE | BAS | 4D | BS | B |
| BRANCH AND SAVE AND SET MODE | BASSM | 0C | - | B |
| BRANCH AND SET MODE | BSM | 0B | - | B |
| COMPARE AND FORM CODEWORD | CFC | B21A | - | B |
| COMPARE AND SWAP | CS | BA | SW | B |
| COMPARE DOUBLE AND SWAP | CDS | BB | SW | B |
| DIVIDE (extended) | DXR | B22D | - | B |
| INSERT PROGRAM MASK | IPM | B222 | - | B |
| MOVE INVERSE | MVCIN | E8 | MI | MI |
| UPDATE TREE | UPT | 0102 | - | B |

**Explanation:**

- `-` Instruction is not provided.
- `*` Those instructions which are part of the floating-point and extended-precision floating-point facilities in System/370 are basic in 370-XA and are not shown. Similarly, those unprivileged instructions which are part of the vector facility are not shown.
- `B` Instruction is basic.
- `BS` Branch-and-save facility.
- `MI` Move-inverse facility.
- `SW` Conditional-swapping facility.

*Figure F-2. Unprivileged Instructions Provided*

| Instruction Name* | Mne-monic | Op Code | System/370 | 370-XA |
|---|---|---|---|---|
| CONNECT CHANNEL SET | CONCS | B200 | CS | - |
| DISCONNECT CHANNEL SET | DISCS | B201 | CS | - |
| EXTRACT PRIMARY ASN | EPAR | B226 | DU | B |
| EXTRACT SECONDARY ASN | ESAR | B227 | DU | B |
| INSERT ADDRESS SPACE CONTROL | IAC | B224 | DU | B |
| INSERT PSW KEY | IPK | B20B | PK | B |
| INSERT STORAGE KEY | ISK | 09 | B | - |
| INSERT STORAGE KEY EXTENDED | ISKE | B229 | EK | B |
| INSERT VIRTUAL STORAGE KEY | IVSK | B223 | DU | B |
| INVALIDATE PAGE TABLE ENTRY | IPTE | B221 | EF | B |
| LOAD ADDRESS SPACE PARAMETERS | LASP | E500 | DU | B |
| LOAD REAL ADDRESS | LRA | B1 | TR | B |
| MOVE TO PRIMARY | MVCP | DA | DU | B |
| MOVE TO SECONDARY | MVCS | DB | DU | B |
| MOVE WITH KEY | MVCK | D9 | DU | B |
| PAGE IN | PGIN | B22E | ES | ES |
| PAGE OUT | PGOUT | B22F | ES | ES |
| PROGRAM CALL | PC | B218 | DU | B |
| PROGRAM TRANSFER | PT | B228 | DU | B |
| PURGE TLB | PTLB | B20D | TR | B |
| READ DIRECT | RDD | 85 | DC | - |
| RESET REFERENCE BIT | RRB | B213 | TR | - |
| RESET REFERENCE BIT EXTENDED | RRBE | B22A | EK | B |
| SET ADDRESS SPACE CONTROL | SAC | B219 | DU | B |
| SET CLOCK COMPARATOR | SCKC | B206 | CK | B |
| SET CPU TIMER | SPT | B208 | CK | B |
| SET PREFIX | SPX | B210 | MP | B |
| SET PSW KEY FROM ADDRESS | SPKA | B20A | PK | B |
| SET SECONDARY ASN | SSAR | B225 | DU | B |
| SET STORAGE KEY | SSK | 08 | B | - |
| SET STORAGE KEY EXTENDED | SSKE | B22B | EK | B |
| SIGNAL PROCESSOR | SIGP | AE | MP | B |
| STORE CLOCK COMPARATOR | STCKC | B207 | CK | B |
| STORE CPU ADDRESS | STAP | B212 | MP | B |
| STORE CPU TIMER | STPT | B209 | CK | B |
| STORE PREFIX | STPX | B211 | MP | B |
| STORE THEN AND SYSTEM MASK | STNSM | AC | TR | B |
| STORE THEN OR SYSTEM MASK | STOSM | AD | TR | B |
| TEST BLOCK | TB | B22C | TB | B |
| TEST PROTECTION | TPROT | E501 | EF | B |
| TRACE | TRACE | 99 | - | B |
| WRITE DIRECT | WRD | 84 | DC | - |

**Explanation:**

- `-` Instruction is not provided.
- `*` Those privileged instructions which are part of the vector facility are not shown.
- `B` Instruction is basic.
- `CK` CPU-timer and clock-comparator facility.
- `CS` Channel-set-switching facility.
- `DC` Direct-control facility.
- `DU` Dual-address-space facility.
- `EF` Extended facility.
- `EK` Storage-key-instruction-extension facility.
- `ES` Expanded-storage facility.
- `MP` Multiprocessing facility.
- `PK` PSW-key-handling facility.
- `TB` Test-block facility.
- `TR` Translation facility.

*Figure F-3. Control Instructions Provided*



LzLabs GmbH. Ex. 1008-964

| Instruction Name | Mne-monic | Op Code | System/370 | 370-XA |
|---|---|---|---|---|
| CLEAR CHANNEL | CLRCH | 9F01 | RE | - |
| CLEAR I/O | CLRIO | 9D01 | B | - |
| HALT DEVICE | HDV | 9E01 | HD | - |
| HALT I/O | HIO | 9E00 | B | - |
| RESUME I/O | RIO | 9C02 | SR | - |
| START I/O | SIO | 9C00 | B | - |
| START I/O FAST RELEASE | SIOF | 9C01 | FR | - |
| STORE CHANNEL ID | STIDC | B203 | B | - |
| TEST CHANNEL | TCH | 9F00 | B | - |
| TEST I/O | TIO | 9D00 | B | - |
| CLEAR SUBCHANNEL | CSCH | B230 | - | B |
| HALT SUBCHANNEL | HSCH | B231 | - | B |
| MODIFY SUBCHANNEL | MSCH | B232 | - | B |
| RESET CHANNEL PATH | RCHP | B23B | - | B |
| RESUME SUBCHANNEL | RSCH | B238 | - | B |
| SET ADDRESS LIMIT | SAL | B237 | - | B |
| SET CHANNEL MONITOR | SCHM | B23C | - | B |
| START SUBCHANNEL | SSCH | B233 | - | B |
| STORE CHANNEL PATH STATUS | STCPS | B23A | - | B |
| STORE CHANNEL REPORT WORD | STCRW | B239 | - | B |
| STORE SUBCHANNEL | STSCH | B234 | - | B |
| TEST PENDING INTERRUPTION | TPI | B236 | - | B |
| TEST SUBCHANNEL | TSCH | B235 | - | B |

**Explanation:**

- Instruction is not provided.
B Instruction is basic.
FR Performs the SIOF function only when the fast-release facility is installed in the channel.
HD Performs the HDV function only when the halt-device facility is installed in the channel.
RE Performs the CLRCH function only when the recovery-extension facility is installed in the channel.
SR Suspend-and-resume facility.

*Figure F-4. I/O Instructions Provided*

## Input/Output Comparison

The channel subsystem has a different logical structure from that of the I/O facilities provided in System/370, with the result that I/O instructions, channels, channel sets, and I/O addressing are replaced in 370-XA by a new set of I/O instructions, by logical device addressing, and by device-accessing mechanisms.

Compatibility with System/370 has been maintained in the CCWs (format 0), 31-bit IDAWs, and channel programs.

In System/370, subchannels are not shared among channels, and each subchannel is associated with only one channel path. In 370-XA, each subchannel is uniquely associated with one I/O device, and that I/O device is uniquely associated with that one subchannel within the channel subsystem, regardless of the number of channel paths by which the I/O device is accessible to the channel subsystem.

Functions are provided in the channel subsystem in 370-XA to detect malfunctions and recover from them if possible. Malfunctions are reported to the program by means of a channel report.

In System/370, I/O interruptions are accepted only by the CPU to which the channel set is currently connected. The I/O interruption causes the I/O address identifying the channel and device causing the interruption to be stored at locations 186-187, and the measurement byte to be stored at real location 185. In 370-XA, I/O interruptions can be accepted by any CPU in the configuration. The subsystem ID and I/O-interruption parameter are stored in the doubleword at real location 184.

Associated with the new I/O instructions is a new program-interruption condition called operand exception.

## Comparison of PSW Formats

Figure F-5 shows those bits and fields in the PSW which are different between System/370 and 370-XA.

| Name of Bit or Field | PSW Bit | System/370 | 370-XA |
|---|---|---|---|
| PER Mask | 1 | TR | B |
| DAT Mode | 5 | TR | B |
| EC Mode | 12 | | |
| Bit 12 = 0 (BC Mode) | | B | - |
| Bit 12 = 1 (EC Mode) | | TR | B[1] |
| Address-space control | 16 | DU | B |
| Addressing mode | 32 | - | B |
| Instruction address | * | B | B |

**Explanation:**

- Mode is not provided.
* The instruction address is in PSW bits 40-63 in System/370 and bits 33-63 in 370-XA.
[1] In 370-XA, PSW bit 12 must be one, and the term "EC mode" is not used.
B Basic.
DU Provided as part of the dual-address-space facility.
TR Provided as part of the translation facility.

*Figure F-5. Comparison of PSW Formats*


LzLabs GmbH. Ex. 1008-965

## Changes in Control-Register Assignments

Figure F-6 on page F-6 shows those bits and fields in the control registers which are different between System/370 and 370-XA.

| Name of Bit or Field | Control-Register Position for | |
|---|---|---|
| | System/370 | 370-XA |
| Block-multiplexing control | 0.0 | - |
| Fetch-protection override | - | 0.6 |
| Storage-key-exception control | 0.7 | - |
| Page-fault-assist control | 0.13 | - |
| Interval-timer subclass mask | 0.24 | - |
| External-signal subclass mask | 0.26 | - |
| Space-switch-event control | 1.31 | 1.0 |
| Primary segment-table origin | 1.8-1.25 | 1.1-1.19 |
| Primary segment-table length | 1.0-1.7 | 1.25-1.31 |
| Channel masks | 2.0-2.31 | - |
| Linkage-table origin | 5.8-5.24 | 5.1-5.24 |
| I/O-interruption subclass mask | - | 6.0-6.7 |
| Secondary segment-table length | 7.0-7.7 | 7.25-7.31 |
| Secondary segment-table origin | 7.8-7.25 | 7.1-7.19 |
| PER starting address | 10.8-10.31 | 10.1-10.31 |
| PER ending address | 11.8-11.31 | 11.1-11.31 |
| Branch-trace control | - | 12.0 |
| Trace-entry address | - | 12.1-12.29 |
| ASN-trace control | - | 12.30 |
| Explicit-trace control | - | 12.31 |
| Check-stop control | 14.0 | - |
| Synchronous-MCEL control | 14.1 | - |
| I/O-extended-logout control | 14.2 | - |
| Channel-report-pending subclass mask | - | 14.3 |
| Asynchronous-MCEL control | 14.8 | - |
| Asynchronous-fixed-log control | 14.9 | - |
| ASN-first-table origin | 14.20-14.31 | 14.13-14.31 |
| MCEL address | 15.8-15.28 | - |

**Explanation:**

\- Bit or field is not provided.

*Figure F-6. Differences in Control-Register Assignments*

## Changes in Assigned Storage Locations

Figure F-7 shows those assigned storage locations where changes have been made between System/370 and 370-XA.

| Name of Field | Assigned Storage Location and Length* for | |
|---|---|---|
| | System/370 | 370-XA |
| Channel-status word | 64 8 | - |
| Channel-address word | 72 4 | - |
| Interval timer | 80 4 | - |
| Trace-table designation | 84 4 | - |
| Channel ID | 168 4 | - |
| IOEL address | 172 4 | - |
| Limited channel logout | 176 4 | - |
| Subsystem ID | - | 184 4 |
| Measurement byte | 185 1 | - |
| I/O address | 186 2 | - |
| I/O-interruption parameter | - | 188 4 |
| Region code | 252 4 | - |
| Fixed-logout area | 256 96 | 256 16 |
| Store-status model-dependent save area | 268 4 | - |
| CPU identity | 795 1 | - |

**Explanation:**

\- Field is not provided.
* The first number is the address, the second the length.

*Figure F-7. Differences in Assigned Storage Locations*

## Changes to SIGNAL PROCESSOR

Figure F-8 on page F-7 and Figure F-9 on page F-7 show those SIGNAL PROCESSOR orders and status codes where changes have been made between System/370 and 370-XA. In addition to these changes, a parameter is provided as part of the SIGNAL PROCESSOR instruction in 370-XA. The parameter is used by the store-status-at-address and set-prefix orders.



LzLabs GmbH. Ex. 1008-966

| Name of Order | Order Code | |
|---|---|---|
| | System/ 370 | 370-XA |
| Initial program reset | 07 | - |
| Program reset | 08 | - |
| Initial microprogram load | 0A | - |
| Set prefix | - | 0D |
| Store status at address | - | 0E |
| **Explanation:**<br>- Order is not provided. | | |

*Figure F-8. Signal-Processor Orders*

| Name of Status Bit | Bit Position | |
|---|---|---|
| | System/370 | 370-XA |
| Incorrect state | - | 22 |
| Invalid parameter | - | 23 |
| Not ready | 28 | - |
| **Explanation:**<br>- Status bit is not provided. | | |

*Figure F-9. Signal-Processor Status Bits*

## Machine-Check Changes

Figure F-10 summarizes those bits and fields in the machine-check-interruption code (MCIC) where changes have been made between System/370 and 370-XA. In addition to these changes, the region code, the machine-check-extended logout, and asynchronous fixed logouts have been eliminated in 370-XA.

| Machine-Check-Interruption Condition or Field | MCIC Bits | |
|---|---|---|
| | System/ 370 | 370-XA |
| Interval-timer damage | 3 | - |
| Channel report pending | - | 9 |
| Channel-subsystem damage | - | 11 |
| Delayed | 15 | - |
| Region-code validity | 25 | - |
| Logout validity | 30 | - |
| MCEL length | 48-63 | - |
| **Explanation:**<br>- Condition or field is not provided. | | |

*Figure F-10. Machine-Check-Interruption-Code Bits*

## Changes to Addressing Wraparound

In System/370, addresses wrap from $2^{24}$ - 1 to zero (or vice versa). In 370-XA, for the 24-bit addressing mode, effective addresses wrap from $2^{24}$ - 1 to zero (or vice versa). For the 31-bit addressing mode, effective addresses wrap from $2^{31}$ - 1 to zero (or vice versa). Except as noted below, real and absolute addresses wrap from $2^{31}$ - 1 to zero.

In 370-XA, the following items cause an I/O program check instead of wraparound:

- Successive CCWs of a CCW list
- Successive IDAWs of an IDAW list
- Successive bytes of I/O data

For DAT-table entries, it is model-dependent whether addresses wrap or cause an addressing exception.

## Changes to LOAD REAL ADDRESS

For LOAD REAL ADDRESS, the addressing of DAT tables is changed to be unpredictable with respect to whether prefixing is applied and to be unpredictable with respect to whether an addressing exception is recognized or wraparound occurs when the calculated address of a page-table or segment-table entry exceeds $2^{31}$ - 1.


LzLabs GmbH. Ex. 1008-967

## Changes to 31-Bit Real Operand Addresses

The following instructions operate by using 31-bit real addresses in System/370. In 370-XA, these instructions operate under control of the addressing mode, bit 32 of the PSW. As a result, in the 24-bit addressing mode, these instructions operate by using 24-bit addresses.

- INSERT STORAGE KEY EXTENDED
- RESET REFERENCE BIT EXTENDED
- SET STORAGE KEY EXTENDED
- TEST BLOCK



LzLabs GmbH. Ex. 1008-968

# Appendix G. Table of Powers of 2

| PLUS | | MINUS |
|---|---|---|
| 1 | 0 | 1. |
| 2 | 1 | 0.5 |
| 4 | 2 | 0.25 |
| 8 | 3 | 0.125 |
| 16 | 4 | 0.0625 |
| 32 | 5 | 0.03125 |
| 64 | 6 | 0.01562 5 |
| 128 | 7 | 0.00781 25 |
| 256 | 8 | 0.00390 625 |
| 512 | 9 | 0.00195 3125 |
| 1,024 | 10 | 0.00097 65625 |
| 2,048 | 11 | 0.00048 82812 5 |
| 4,096 | 12 | 0.00024 41406 25 |
| 8,192 | 13 | 0.00012 20703 125 |
| 16,384 | 14 | 0.00006 10351 5625 |
| 32,768 | 15 | 0.00003 05175 78125 |
| 65,536 | 16 | 0.00001 52587 89062 5 |
| 131,072 | 17 | 0.00000 76293 94531 25 |
| 262,144 | 18 | 0.00000 38146 97265 625 |
| 524,288 | 19 | 0.00000 19073 48632 8125 |
| 1,048,576 | 20 | 0.00000 09536 74316 40625 |
| 2,097,152 | 21 | 0.00000 04768 37158 20312 5 |
| 4,194,304 | 22 | 0.00000 02384 18579 10156 25 |
| 8,388,608 | 23 | 0.00000 01192 09289 55078 125 |
| 16,777,216 | 24 | 0.00000 00596 04644 77539 0625 |
| 33,554,432 | 25 | 0.00000 00298 02322 38769 53125 |
| 67,108,864 | 26 | 0.00000 00149 01161 19384 76562 5 |
| 134,217,728 | 27 | 0.00000 00074 50580 59692 38281 25 |
| 268,435,456 | 28 | 0.00000 00037 25290 29846 19140 625 |
| 536,870,912 | 29 | 0.00000 00018 62645 14923 09570 3125 |
| 1,073,741,824 | 30 | 0.00000 00009 31322 57461 54785 15625 |
| 2,147,483,648 | 31 | 0.00000 00004 65661 28730 77392 57812 5 |
| 4,294,967,296 | 32 | 0.00000 00002 32830 64365 38696 28906 25 |
| 8,589,934,592 | 33 | 0.00000 00001 16415 32182 69348 14453 125 |
| 17,179,869,184 | 34 | 0.00000 00000 58207 66091 34674 07226 5625 |
| 34,359,738,368 | 35 | 0.00000 00000 29103 83045 67337 03613 28125 |
| 68,719,476,736 | 36 | 0.00000 00000 14551 91522 83668 51806 64062 5 |
| 137,438,953,472 | 37 | 0.00000 00000 07275 95761 41834 25903 32031 25 |
| 274,877,906,944 | 38 | 0.00000 00000 03637 97880 70917 12951 66015 625 |
| 549,755,813,888 | 39 | 0.00000 00000 01818 98940 35458 56475 83007 8125 |
| 1,099,511,627,776 | 40 | 0.00000 00000 00909 49470 17729 28237 91503 90625 |
| 2,199,023,255,552 | 41 | 0.00000 00000 00454 74735 08864 64118 95751 95312 5 |
| 4,398,046,511,104 | 42 | 0.00000 00000 00227 37367 54432 32059 47875 97656 25 |
| 8,796,093,022,208 | 43 | 0.00000 00000 00113 68683 77216 16029 73937 98828 125 |
| 17,592,186,044,416 | 44 | 0.00000 00000 00056 84341 88608 08014 86968 99414 0625 |
| 35,184,372,088,832 | 45 | 0.00000 00000 00028 42170 94304 04007 43484 49707 03125 |
| 70,368,744,177,664 | 46 | 0.00000 00000 00014 21085 47152 02003 71742 24853 51562 5 |
| 140,737,488,355,328 | 47 | 0.00000 00000 00007 10542 73576 01001 85871 12426 75781 25 |
| 281,474,976,710,656 | 48 | 0.00000 00000 00003 55271 36788 00500 92935 56213 37890 625 |
| 562,949,953,421,312 | 49 | 0.00000 00000 00001 77635 68394 00250 46467 78106 68945 3125 |
| 1,125,899,906,842,624 | 50 | 0.00000 00000 00000 88817 84197 00125 23233 89053 34472 65625 |
| 2,251,799,813,685,248 | 51 | 0.00000 00000 00000 44408 92098 50062 61616 94526 67236 32812 5 |
| 4,503,599,627,370,496 | 52 | 0.00000 00000 00000 22204 46049 25031 30808 47263 33618 16406 25 |
| 9,007,199,254,740,992 | 53 | 0.00000 00000 00000 11102 23024 62515 65404 23631 66809 08203 125 |
| 18,014,398,509,481,984 | 54 | 0.00000 00000 00000 05551 11512 31257 82702 11815 83404 54101 5625 |
| 36,028,797,018,963,968 | 55 | 0.00000 00000 00000 02775 55756 15628 91351 05907 91702 27050 78125 |
| 72,057,594,037,927,936 | 56 | 0.00000 00000 00000 01387 77878 07814 45675 52953 95851 13525 39062 5 |
| 144,115,188,075,855,872 | 57 | 0.00000 00000 00000 00693 88939 03907 22837 76476 97925 56762 69531 25 |
| 288,230,376,151,711,744 | 58 | 0.00000 00000 00000 00346 94469 51953 61418 88238 48962 78381 34765 625 |
| 576,460,752,303,423,488 | 59 | 0.00000 00000 00000 00173 47234 75976 80709 44119 24481 39190 67382 8125 |
| 1,152,921,504,606,846,976 | 60 | 0.00000 00000 00000 00086 73617 37988 40354 72059 62240 69595 33691 40625 |
| 2,305,843,009,213,693,952 | 61 | 0.00000 00000 00000 00043 36808 68994 20177 36029 81120 34797 66845 70312 5 |
| 4,611,686,018,427,387,904 | 62 | 0.00000 00000 00000 00021 68404 34497 10088 68014 90560 17398 83422 85156 25 |
| 9,223,372,036,854,775,808 | 63 | 0.00000 00000 00000 00010 84202 17248 55044 34007 45280 08699 41711 42578 125 |
| 18,446,744,073,709,551,616 | 64 | 0.00000 00000 00000 00005 42101 08624 27522 17003 72640 04349 70855 71289 0625 |

*Figure G-1 (Part 1 of 2). Powers of 2*

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-969

```
                    18,446,744,073,709,551,616   64
                    36,893,488,147,419,103,232   65
                    73,786,976,294,838,206,464   66
                   147,573,952,589,676,412,928   67

                   295,147,905,179,352,825,856   68
                   590,295,810,358,705,651,712   69
                 1,180,591,620,717,411,303,424   70
                 2,361,183,241,434,822,606,848   71

                 4,722,366,482,869,645,213,696   72
                 9,444,732,965,739,290,427,392   73
                18,889,465,931,478,580,854,784   74
                37,778,931,862,957,161,709,568   75

                75,557,863,725,914,323,419,136   76
               151,115,727,451,828,646,838,272   77
               302,231,454,903,657,293,676,544   78
               604,462,909,807,314,587,353,088   79

             1,208,925,819,614,629,174,706,176   80
             2,417,851,639,229,258,349,412,352   81
             4,835,703,278,458,516,698,824,704   82
             9,671,406,556,917,033,397,649,408   83

            19,342,813,113,834,066,795,298,816   84
            38,685,626,227,668,133,590,597,632   85
            77,371,252,455,336,267,181,195,264   86
           154,742,504,910,672,534,362,390,528   87

           309,485,009,821,345,068,724,781,056   88
           618,970,019,642,690,137,449,562,112   89
         1,237,940,039,285,380,274,899,124,224   90
         2,475,880,078,570,760,549,798,248,448   91

         4,951,760,157,141,521,099,596,496,896   92
         9,903,520,314,283,042,199,192,993,792   93
        19,807,040,628,566,084,398,385,987,584   94
        39,614,081,257,132,168,796,771,975,168   95

        79,228,162,514,264,337,593,543,950,336   96
       158,456,325,028,528,675,187,087,900,672   97
       316,912,650,057,057,350,374,175,801,344   98
       633,825,300,114,114,700,748,351,602,688   99

     1,267,650,600,228,229,401,496,703,205,376  100
     2,535,301,200,456,458,802,993,406,410,752  101
     5,070,602,400,912,917,605,986,812,821,504  102
    10,141,204,801,825,835,211,973,625,643,008  103

    20,282,409,603,651,670,423,947,251,286,016  104
    40,564,819,207,303,340,847,894,502,572,032  105
    81,129,638,414,606,681,695,789,005,144,064  106
   162,259,276,829,213,363,391,578,010,288,128  107

   324,518,553,658,426,726,783,156,020,576,256  108
   649,037,107,316,853,453,566,312,041,152,512  109
 1,298,074,214,633,706,907,132,624,082,305,024  110
 2,596,148,429,267,413,814,265,248,164,610,048  111

 5,192,296,858,534,827,628,530,496,329,220,096  112
10,384,593,717,069,655,257,060,992,658,440,192  113
20,769,187,434,139,310,514,121,985,316,880,384  114
41,538,374,868,278,621,028,243,970,633,760,768  115

 83,076,749,736,557,242,056,487,941,267,521,536  116
166,153,499,473,114,484,112,975,882,535,043,072  117
332,306,998,946,228,968,225,951,765,070,086,144  118
664,613,997,892,457,936,451,903,530,140,172,288  119

1,329,227,995,784,915,872,903,807,060,280,344,576  120
2,658,455,991,569,831,745,807,614,120,560,689,152  121
5,316,911,983,139,663,491,615,228,241,121,378,304  122
10,633,823,966,279,326,983,230,456,482,242,756,608  123

21,267,647,932,558,653,966,460,912,964,485,513,216  124
42,535,295,865,117,307,932,921,825,928,971,026,432  125
85,070,591,730,234,615,865,843,651,857,942,052,864  126
170,141,183,460,469,231,731,687,303,715,884,105,728  127

340,282,366,920,938,463,463,374,607,431,768,211,456  128
```

*Figure G-1 (Part 2 of 2). Powers of 2*



LzLabs GmbH. Ex. 1008-970

# Appendix H. Hexadecimal Tables

The following tables aid in converting hexadecimal values to decimal values, or the reverse.

***Direct Conversion Table***

This table provides direct conversion of decimal and hexadecimal numbers in these ranges:

```
Hexadecimal      Decimal
000 to FFF     0000 to 4095
```

To convert numbers outside these ranges, and to convert fractions, use the hexadecimal and decimal conversion tables that follow the direct conversion table in this appendix.

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-971

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00_ | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0008 | 0009 | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 |
| 01_ | 0016 | 0017 | 0018 | 0019 | 0020 | 0021 | 0022 | 0023 | 0024 | 0025 | 0026 | 0027 | 0028 | 0029 | 0030 | 0031 |
| 02_ | 0032 | 0033 | 0034 | 0035 | 0036 | 0037 | 0038 | 0039 | 0040 | 0041 | 0042 | 0043 | 0044 | 0045 | 0046 | 0047 |
| 03_ | 0048 | 0049 | 0050 | 0051 | 0052 | 0053 | 0054 | 0055 | 0056 | 0057 | 0058 | 0059 | 0060 | 0061 | 0062 | 0063 |
| 04_ | 0064 | 0065 | 0066 | 0067 | 0068 | 0069 | 0070 | 0071 | 0072 | 0073 | 0074 | 0075 | 0076 | 0077 | 0078 | 0079 |
| 05_ | 0080 | 0081 | 0082 | 0083 | 0084 | 0085 | 0086 | 0087 | 0088 | 0089 | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 |
| 06_ | 0096 | 0097 | 0098 | 0099 | 0100 | 0101 | 0102 | 0103 | 0104 | 0105 | 0106 | 0107 | 0108 | 0109 | 0110 | 0111 |
| 07_ | 0112 | 0113 | 0114 | 0115 | 0116 | 0117 | 0118 | 0119 | 0120 | 0121 | 0122 | 0123 | 0124 | 0125 | 0126 | 0127 |
| 08_ | 0128 | 0129 | 0130 | 0131 | 0132 | 0133 | 0134 | 0135 | 0136 | 0137 | 0138 | 0139 | 0140 | 0141 | 0142 | 0143 |
| 09_ | 0144 | 0145 | 0146 | 0147 | 0148 | 0149 | 0150 | 0151 | 0152 | 0153 | 0154 | 0155 | 0156 | 0157 | 0158 | 0159 |
| 0A_ | 0160 | 0161 | 0162 | 0163 | 0164 | 0165 | 0166 | 0167 | 0168 | 0169 | 0170 | 0171 | 0172 | 0173 | 0174 | 0175 |
| 0B_ | 0176 | 0177 | 0178 | 0179 | 0180 | 0181 | 0182 | 0183 | 0184 | 0185 | 0186 | 0187 | 0188 | 0189 | 0190 | 0191 |
| 0C_ | 0192 | 0193 | 0194 | 0195 | 0196 | 0197 | 0198 | 0199 | 0200 | 0201 | 0202 | 0203 | 0204 | 0205 | 0206 | 0207 |
| 0D_ | 0208 | 0209 | 0210 | 0211 | 0212 | 0213 | 0214 | 0215 | 0216 | 0217 | 0218 | 0219 | 0220 | 0221 | 0222 | 0223 |
| 0E_ | 0224 | 0225 | 0226 | 0227 | 0228 | 0229 | 0230 | 0231 | 0232 | 0233 | 0234 | 0235 | 0236 | 0237 | 0238 | 0239 |
| 0F_ | 0240 | 0241 | 0242 | 0243 | 0244 | 0245 | 0246 | 0247 | 0248 | 0249 | 0250 | 0251 | 0252 | 0253 | 0254 | 0255 |
| 10_ | 0256 | 0257 | 0258 | 0259 | 0260 | 0261 | 0262 | 0263 | 0264 | 0265 | 0266 | 0267 | 0268 | 0269 | 0270 | 0271 |
| 11_ | 0272 | 0273 | 0274 | 0275 | 0276 | 0277 | 0278 | 0279 | 0280 | 0281 | 0282 | 0283 | 0284 | 0285 | 0286 | 0287 |
| 12_ | 0288 | 0289 | 0290 | 0291 | 0292 | 0293 | 0294 | 0295 | 0296 | 0297 | 0298 | 0299 | 0300 | 0301 | 0302 | 0303 |
| 13_ | 0304 | 0305 | 0306 | 0307 | 0308 | 0309 | 0310 | 0311 | 0312 | 0313 | 0314 | 0315 | 0316 | 0317 | 0318 | 0319 |
| 14_ | 0320 | 0321 | 0322 | 0323 | 0324 | 0325 | 0326 | 0327 | 0328 | 0329 | 0330 | 0331 | 0332 | 0333 | 0334 | 0335 |
| 15_ | 0336 | 0337 | 0338 | 0339 | 0340 | 0341 | 0342 | 0343 | 0344 | 0345 | 0346 | 0347 | 0348 | 0349 | 0350 | 0351 |
| 16_ | 0352 | 0353 | 0354 | 0355 | 0356 | 0357 | 0358 | 0359 | 0360 | 0361 | 0362 | 0363 | 0364 | 0365 | 0366 | 0367 |
| 17_ | 0368 | 0369 | 0370 | 0371 | 0372 | 0373 | 0374 | 0375 | 0376 | 0377 | 0378 | 0379 | 0380 | 0381 | 0382 | 0383 |
| 18_ | 0384 | 0385 | 0386 | 0387 | 0388 | 0389 | 0390 | 0391 | 0392 | 0393 | 0394 | 0395 | 0396 | 0397 | 0398 | 0399 |
| 19_ | 0400 | 0401 | 0402 | 0403 | 0404 | 0405 | 0406 | 0407 | 0408 | 0409 | 0410 | 0411 | 0412 | 0413 | 0414 | 0415 |
| 1A_ | 0416 | 0417 | 0418 | 0419 | 0420 | 0421 | 0422 | 0423 | 0424 | 0425 | 0426 | 0427 | 0428 | 0429 | 0430 | 0431 |
| 1B_ | 0432 | 0433 | 0434 | 0435 | 0436 | 0437 | 0438 | 0439 | 0440 | 0441 | 0442 | 0443 | 0444 | 0445 | 0446 | 0447 |
| 1C_ | 0448 | 0449 | 0450 | 0451 | 0452 | 0453 | 0454 | 0455 | 0456 | 0457 | 0458 | 0459 | 0460 | 0461 | 0462 | 0463 |
| 1D_ | 0464 | 0465 | 0466 | 0467 | 0468 | 0469 | 0470 | 0471 | 0472 | 0473 | 0474 | 0475 | 0476 | 0477 | 0478 | 0479 |
| 1E_ | 0480 | 0481 | 0482 | 0483 | 0484 | 0485 | 0486 | 0487 | 0488 | 0489 | 0490 | 0491 | 0492 | 0493 | 0494 | 0495 |
| 1F_ | 0496 | 0497 | 0498 | 0499 | 0500 | 0501 | 0502 | 0503 | 0504 | 0505 | 0506 | 0507 | 0508 | 0509 | 0510 | 0511 |
| 20_ | 0512 | 0513 | 0514 | 0515 | 0516 | 0517 | 0518 | 0519 | 0520 | 0521 | 0522 | 0523 | 0524 | 0525 | 0526 | 0527 |
| 21_ | 0528 | 0529 | 0530 | 0531 | 0532 | 0533 | 0534 | 0535 | 0536 | 0537 | 0538 | 0539 | 0540 | 0541 | 0542 | 0543 |
| 22_ | 0544 | 0545 | 0546 | 0547 | 0548 | 0549 | 0550 | 0551 | 0552 | 0553 | 0554 | 0555 | 0556 | 0557 | 0558 | 0559 |
| 23_ | 0560 | 0561 | 0562 | 0563 | 0564 | 0565 | 0566 | 0567 | 0568 | 0569 | 0570 | 0571 | 0572 | 0573 | 0574 | 0575 |
| 24_ | 0576 | 0577 | 0578 | 0579 | 0580 | 0581 | 0582 | 0583 | 0584 | 0585 | 0586 | 0587 | 0588 | 0589 | 0590 | 0591 |
| 25_ | 0592 | 0593 | 0594 | 0595 | 0596 | 0597 | 0598 | 0599 | 0600 | 0601 | 0602 | 0603 | 0604 | 0605 | 0606 | 0607 |
| 26_ | 0608 | 0609 | 0610 | 0611 | 0612 | 0613 | 0614 | 0615 | 0616 | 0617 | 0618 | 0619 | 0620 | 0621 | 0622 | 0623 |
| 27_ | 0624 | 0625 | 0626 | 0627 | 0628 | 0629 | 0630 | 0631 | 0632 | 0633 | 0634 | 0635 | 0636 | 0637 | 0638 | 0639 |
| 28_ | 0640 | 0641 | 0642 | 0643 | 0644 | 0645 | 0646 | 0647 | 0648 | 0649 | 0650 | 0651 | 0652 | 0653 | 0654 | 0655 |
| 29_ | 0656 | 0657 | 0658 | 0659 | 0660 | 0661 | 0662 | 0663 | 0664 | 0665 | 0666 | 0667 | 0668 | 0669 | 0670 | 0671 |
| 2A_ | 0672 | 0673 | 0674 | 0675 | 0676 | 0677 | 0678 | 0679 | 0680 | 0681 | 0682 | 0683 | 0684 | 0685 | 0686 | 0687 |
| 2B_ | 0688 | 0689 | 0690 | 0691 | 0692 | 0693 | 0694 | 0695 | 0696 | 0697 | 0698 | 0699 | 0700 | 0701 | 0702 | 0703 |
| 2C_ | 0704 | 0705 | 0706 | 0707 | 0708 | 0709 | 0710 | 0711 | 0712 | 0713 | 0714 | 0715 | 0716 | 0717 | 0718 | 0719 |
| 2D_ | 0720 | 0721 | 0722 | 0723 | 0724 | 0725 | 0726 | 0727 | 0728 | 0729 | 0730 | 0731 | 0732 | 0733 | 0734 | 0735 |
| 2E_ | 0736 | 0737 | 0738 | 0739 | 0740 | 0741 | 0742 | 0743 | 0744 | 0745 | 0746 | 0747 | 0748 | 0749 | 0750 | 0751 |
| 2F_ | 0752 | 0753 | 0754 | 0755 | 0756 | 0757 | 0758 | 0759 | 0760 | 0761 | 0762 | 0763 | 0764 | 0765 | 0766 | 0767 |
| 30_ | 0768 | 0769 | 0770 | 0771 | 0772 | 0773 | 0774 | 0775 | 0776 | 0777 | 0778 | 0779 | 0780 | 0781 | 0782 | 0783 |
| 31_ | 0784 | 0785 | 0786 | 0787 | 0788 | 0789 | 0790 | 0791 | 0792 | 0793 | 0794 | 0795 | 0796 | 0797 | 0798 | 0799 |
| 32_ | 0800 | 0801 | 0802 | 0803 | 0804 | 0805 | 0806 | 0807 | 0808 | 0809 | 0810 | 0811 | 0812 | 0813 | 0814 | 0815 |
| 33_ | 0816 | 0817 | 0818 | 0819 | 0820 | 0821 | 0822 | 0823 | 0824 | 0825 | 0826 | 0827 | 0828 | 0829 | 0830 | 0831 |
| 34_ | 0832 | 0833 | 0834 | 0835 | 0836 | 0837 | 0838 | 0839 | 0840 | 0841 | 0842 | 0843 | 0844 | 0845 | 0846 | 0847 |
| 35_ | 0848 | 0849 | 0850 | 0851 | 0852 | 0853 | 0854 | 0855 | 0856 | 0857 | 0858 | 0859 | 0860 | 0861 | 0862 | 0863 |
| 36_ | 0864 | 0865 | 0866 | 0867 | 0868 | 0869 | 0870 | 0871 | 0872 | 0873 | 0874 | 0875 | 0876 | 0877 | 0878 | 0879 |
| 37_ | 0880 | 0881 | 0882 | 0883 | 0884 | 0885 | 0886 | 0887 | 0888 | 0889 | 0890 | 0891 | 0892 | 0893 | 0894 | 0895 |
| 38_ | 0896 | 0897 | 0898 | 0899 | 0900 | 0901 | 0902 | 0903 | 0904 | 0905 | 0906 | 0907 | 0908 | 0909 | 0910 | 0911 |
| 39_ | 0912 | 0913 | 0914 | 0915 | 0916 | 0917 | 0918 | 0919 | 0920 | 0921 | 0922 | 0923 | 0924 | 0925 | 0926 | 0927 |
| 3A_ | 0928 | 0929 | 0930 | 0931 | 0932 | 0933 | 0934 | 0935 | 0936 | 0937 | 0938 | 0939 | 0940 | 0941 | 0942 | 0943 |
| 3B_ | 0944 | 0945 | 0946 | 0947 | 0948 | 0949 | 0950 | 0951 | 0952 | 0953 | 0954 | 0955 | 0956 | 0957 | 0958 | 0959 |
| 3C_ | 0960 | 0961 | 0962 | 0963 | 0964 | 0965 | 0966 | 0967 | 0968 | 0969 | 0970 | 0971 | 0972 | 0973 | 0974 | 0975 |
| 3D_ | 0976 | 0977 | 0978 | 0979 | 0980 | 0981 | 0982 | 0983 | 0984 | 0985 | 0986 | 0987 | 0988 | 0989 | 0990 | 0991 |
| 3E_ | 0992 | 0993 | 0994 | 0995 | 0996 | 0997 | 0998 | 0999 | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 |
| 3F_ | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 |



LzLabs GmbH. Ex. 1008-972

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40_ | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 | 1031 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 |
| 41_ | 1040 | 1041 | 1042 | 1043 | 1044 | 1045 | 1046 | 1047 | 1048 | 1049 | 1050 | 1051 | 1052 | 1053 | 1054 | 1055 |
| 42_ | 1056 | 1057 | 1058 | 1059 | 1060 | 1061 | 1062 | 1063 | 1064 | 1065 | 1066 | 1067 | 1068 | 1069 | 1070 | 1071 |
| 43_ | 1072 | 1073 | 1074 | 1075 | 1076 | 1077 | 1078 | 1079 | 1080 | 1081 | 1082 | 1083 | 1084 | 1085 | 1086 | 1087 |
| 44_ | 1088 | 1089 | 1090 | 1091 | 1092 | 1093 | 1094 | 1095 | 1096 | 1097 | 1098 | 1099 | 1100 | 1101 | 1102 | 1103 |
| 45_ | 1104 | 1105 | 1106 | 1107 | 1108 | 1109 | 1110 | 1111 | 1112 | 1113 | 1114 | 1115 | 1116 | 1117 | 1118 | 1119 |
| 46_ | 1120 | 1121 | 1122 | 1123 | 1124 | 1125 | 1126 | 1127 | 1128 | 1129 | 1130 | 1131 | 1132 | 1133 | 1134 | 1135 |
| 47_ | 1136 | 1137 | 1138 | 1139 | 1140 | 1141 | 1142 | 1143 | 1144 | 1145 | 1146 | 1147 | 1148 | 1149 | 1150 | 1151 |
| 48_ | 1152 | 1153 | 1154 | 1155 | 1156 | 1157 | 1158 | 1159 | 1160 | 1161 | 1162 | 1163 | 1164 | 1165 | 1166 | 1167 |
| 49_ | 1168 | 1169 | 1170 | 1171 | 1172 | 1173 | 1174 | 1175 | 1176 | 1177 | 1178 | 1179 | 1180 | 1181 | 1182 | 1183 |
| 4A_ | 1184 | 1185 | 1186 | 1187 | 1188 | 1189 | 1190 | 1191 | 1192 | 1193 | 1194 | 1195 | 1196 | 1197 | 1198 | 1199 |
| 4B_ | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | 1206 | 1207 | 1208 | 1209 | 1210 | 1211 | 1212 | 1213 | 1214 | 1215 |
| 4C_ | 1216 | 1217 | 1218 | 1219 | 1220 | 1221 | 1222 | 1223 | 1224 | 1225 | 1226 | 1227 | 1228 | 1229 | 1230 | 1231 |
| 4D_ | 1232 | 1233 | 1234 | 1235 | 1236 | 1237 | 1238 | 1239 | 1240 | 1241 | 1242 | 1243 | 1244 | 1245 | 1246 | 1247 |
| 4E_ | 1248 | 1249 | 1250 | 1251 | 1252 | 1253 | 1254 | 1255 | 1256 | 1257 | 1258 | 1259 | 1260 | 1261 | 1262 | 1263 |
| 4F_ | 1264 | 1265 | 1266 | 1267 | 1268 | 1269 | 1270 | 1271 | 1272 | 1273 | 1274 | 1275 | 1276 | 1277 | 1278 | 1279 |
| 50_ | 1280 | 1281 | 1282 | 1283 | 1284 | 1285 | 1286 | 1287 | 1288 | 1289 | 1290 | 1291 | 1292 | 1293 | 1294 | 1295 |
| 51_ | 1296 | 1297 | 1298 | 1299 | 1300 | 1301 | 1302 | 1303 | 1304 | 1305 | 1306 | 1307 | 1308 | 1309 | 1310 | 1311 |
| 52_ | 1312 | 1313 | 1314 | 1315 | 1316 | 1317 | 1318 | 1319 | 1320 | 1321 | 1322 | 1323 | 1324 | 1325 | 1326 | 1327 |
| 53_ | 1328 | 1329 | 1330 | 1331 | 1332 | 1333 | 1334 | 1335 | 1336 | 1337 | 1338 | 1339 | 1340 | 1341 | 1342 | 1343 |
| 54_ | 1344 | 1345 | 1346 | 1347 | 1348 | 1349 | 1350 | 1351 | 1352 | 1353 | 1354 | 1355 | 1356 | 1357 | 1358 | 1359 |
| 55_ | 1360 | 1361 | 1362 | 1363 | 1364 | 1365 | 1366 | 1367 | 1368 | 1369 | 1370 | 1371 | 1372 | 1373 | 1374 | 1375 |
| 56_ | 1376 | 1377 | 1378 | 1379 | 1380 | 1381 | 1382 | 1383 | 1384 | 1385 | 1386 | 1387 | 1388 | 1389 | 1390 | 1391 |
| 57_ | 1392 | 1393 | 1394 | 1395 | 1396 | 1397 | 1398 | 1399 | 1400 | 1401 | 1402 | 1403 | 1404 | 1405 | 1406 | 1407 |
| 58_ | 1408 | 1409 | 1410 | 1411 | 1412 | 1413 | 1414 | 1415 | 1416 | 1417 | 1418 | 1419 | 1420 | 1421 | 1422 | 1423 |
| 59_ | 1424 | 1425 | 1426 | 1427 | 1428 | 1429 | 1430 | 1431 | 1432 | 1433 | 1434 | 1435 | 1436 | 1437 | 1438 | 1439 |
| 5A_ | 1440 | 1441 | 1442 | 1443 | 1444 | 1445 | 1446 | 1447 | 1448 | 1449 | 1450 | 1451 | 1452 | 1453 | 1454 | 1455 |
| 5B_ | 1456 | 1457 | 1458 | 1459 | 1460 | 1461 | 1462 | 1463 | 1464 | 1465 | 1466 | 1467 | 1468 | 1469 | 1470 | 1471 |
| 5C_ | 1472 | 1473 | 1474 | 1475 | 1476 | 1477 | 1478 | 1479 | 1480 | 1481 | 1482 | 1483 | 1484 | 1485 | 1486 | 1487 |
| 5D_ | 1488 | 1489 | 1490 | 1491 | 1492 | 1493 | 1494 | 1495 | 1496 | 1497 | 1498 | 1499 | 1500 | 1501 | 1502 | 1503 |
| 5E_ | 1504 | 1505 | 1506 | 1507 | 1508 | 1509 | 1510 | 1511 | 1512 | 1513 | 1514 | 1515 | 1516 | 1517 | 1518 | 1519 |
| 5F_ | 1520 | 1521 | 1522 | 1523 | 1524 | 1525 | 1526 | 1527 | 1528 | 1529 | 1530 | 1531 | 1532 | 1533 | 1534 | 1535 |
| 60_ | 1536 | 1537 | 1538 | 1539 | 1540 | 1541 | 1542 | 1543 | 1544 | 1545 | 1546 | 1547 | 1548 | 1549 | 1550 | 1551 |
| 61_ | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 | 1559 | 1560 | 1561 | 1562 | 1563 | 1564 | 1565 | 1566 | 1567 |
| 62_ | 1568 | 1569 | 1570 | 1571 | 1572 | 1573 | 1574 | 1575 | 1576 | 1577 | 1578 | 1579 | 1580 | 1581 | 1582 | 1583 |
| 63_ | 1584 | 1585 | 1586 | 1587 | 1588 | 1589 | 1590 | 1591 | 1592 | 1593 | 1594 | 1595 | 1596 | 1597 | 1598 | 1599 |
| 64_ | 1600 | 1601 | 1602 | 1603 | 1604 | 1605 | 1606 | 1607 | 1608 | 1609 | 1610 | 1611 | 1612 | 1613 | 1614 | 1615 |
| 65_ | 1616 | 1617 | 1618 | 1619 | 1620 | 1621 | 1622 | 1623 | 1624 | 1625 | 1626 | 1627 | 1628 | 1629 | 1630 | 1631 |
| 66_ | 1632 | 1633 | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 | 1640 | 1641 | 1642 | 1643 | 1644 | 1645 | 1646 | 1647 |
| 67_ | 1648 | 1649 | 1650 | 1651 | 1652 | 1653 | 1654 | 1655 | 1656 | 1657 | 1658 | 1659 | 1660 | 1661 | 1662 | 1663 |
| 68_ | 1664 | 1665 | 1666 | 1667 | 1668 | 1669 | 1670 | 1671 | 1672 | 1673 | 1674 | 1675 | 1676 | 1677 | 1678 | 1679 |
| 69_ | 1680 | 1681 | 1682 | 1683 | 1684 | 1685 | 1686 | 1687 | 1688 | 1689 | 1690 | 1691 | 1692 | 1693 | 1694 | 1695 |
| 6A_ | 1696 | 1697 | 1698 | 1699 | 1700 | 1701 | 1702 | 1703 | 1704 | 1705 | 1706 | 1707 | 1708 | 1709 | 1710 | 1711 |
| 6B_ | 1712 | 1713 | 1714 | 1715 | 1716 | 1717 | 1718 | 1719 | 1720 | 1721 | 1722 | 1723 | 1724 | 1725 | 1726 | 1727 |
| 6C_ | 1728 | 1729 | 1730 | 1731 | 1732 | 1733 | 1734 | 1735 | 1736 | 1737 | 1738 | 1739 | 1740 | 1741 | 1742 | 1743 |
| 6D_ | 1744 | 1745 | 1746 | 1747 | 1748 | 1749 | 1750 | 1751 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 | 1758 | 1759 |
| 6E_ | 1760 | 1761 | 1762 | 1763 | 1764 | 1765 | 1766 | 1767 | 1768 | 1769 | 1770 | 1771 | 1772 | 1773 | 1774 | 1775 |
| 6F_ | 1776 | 1777 | 1778 | 1779 | 1780 | 1781 | 1782 | 1783 | 1784 | 1785 | 1786 | 1787 | 1788 | 1789 | 1790 | 1791 |
| 70_ | 1792 | 1793 | 1794 | 1795 | 1796 | 1797 | 1798 | 1799 | 1800 | 1801 | 1802 | 1803 | 1804 | 1805 | 1806 | 1807 |
| 71_ | 1808 | 1809 | 1810 | 1811 | 1812 | 1813 | 1814 | 1815 | 1816 | 1817 | 1818 | 1819 | 1820 | 1821 | 1822 | 1823 |
| 72_ | 1824 | 1825 | 1826 | 1827 | 1828 | 1829 | 1830 | 1831 | 1832 | 1833 | 1834 | 1835 | 1836 | 1837 | 1838 | 1839 |
| 73_ | 1840 | 1841 | 1842 | 1843 | 1844 | 1845 | 1846 | 1847 | 1848 | 1849 | 1850 | 1851 | 1852 | 1853 | 1854 | 1855 |
| 74_ | 1856 | 1857 | 1858 | 1859 | 1860 | 1861 | 1862 | 1863 | 1864 | 1865 | 1866 | 1867 | 1868 | 1869 | 1870 | 1871 |
| 75_ | 1872 | 1873 | 1874 | 1875 | 1876 | 1877 | 1878 | 1879 | 1880 | 1881 | 1882 | 1883 | 1884 | 1885 | 1886 | 1887 |
| 76_ | 1888 | 1889 | 1890 | 1891 | 1892 | 1893 | 1894 | 1895 | 1896 | 1897 | 1898 | 1899 | 1900 | 1901 | 1902 | 1903 |
| 77_ | 1904 | 1905 | 1906 | 1907 | 1908 | 1909 | 1910 | 1911 | 1912 | 1913 | 1914 | 1915 | 1916 | 1917 | 1918 | 1919 |
| 78_ | 1920 | 1921 | 1922 | 1923 | 1924 | 1925 | 1926 | 1927 | 1928 | 1929 | 1930 | 1931 | 1932 | 1933 | 1934 | 1935 |
| 79_ | 1936 | 1937 | 1938 | 1939 | 1940 | 1941 | 1942 | 1943 | 1944 | 1945 | 1946 | 1947 | 1948 | 1949 | 1950 | 1951 |
| 7A_ | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 | 1960 | 1961 | 1962 | 1963 | 1964 | 1965 | 1966 | 1967 |
| 7B_ | 1968 | 1969 | 1970 | 1971 | 1972 | 1973 | 1974 | 1975 | 1976 | 1977 | 1978 | 1979 | 1980 | 1981 | 1982 | 1983 |
| 7C_ | 1984 | 1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 |
| 7D_ | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 |
| 7E_ | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 | 2025 | 2026 | 2027 | 2028 | 2029 | 2030 | 2031 |
| 7F_ | 2032 | 2033 | 2034 | 2035 | 2036 | 2037 | 2038 | 2039 | 2040 | 2041 | 2042 | 2043 | 2044 | 2045 | 2046 | 2047 |


LzLabs GmbH. Ex. 1008-973

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 80_ | 2048 | 2049 | 2050 | 2051 | 2052 | 2053 | 2054 | 2055 | 2056 | 2057 | 2058 | 2059 | 2060 | 2061 | 2062 | 2063 |
| 81_ | 2064 | 2065 | 2066 | 2067 | 2068 | 2069 | 2070 | 2071 | 2072 | 2073 | 2074 | 2075 | 2076 | 2077 | 2078 | 2079 |
| 82_ | 2080 | 2081 | 2082 | 2083 | 2084 | 2085 | 2086 | 2087 | 2088 | 2089 | 2090 | 2091 | 2092 | 2093 | 2094 | 2095 |
| 83_ | 2096 | 2097 | 2098 | 2099 | 2100 | 2101 | 2102 | 2103 | 2104 | 2105 | 2106 | 2107 | 2108 | 2109 | 2110 | 2111 |
| 84_ | 2112 | 2113 | 2114 | 2115 | 2116 | 2117 | 2118 | 2119 | 2120 | 2121 | 2122 | 2123 | 2124 | 2125 | 2126 | 2127 |
| 85_ | 2128 | 2129 | 2130 | 2131 | 2132 | 2133 | 2134 | 2135 | 2136 | 2137 | 2138 | 2139 | 2140 | 2141 | 2142 | 2143 |
| 86_ | 2144 | 2145 | 2146 | 2147 | 2148 | 2149 | 2150 | 2151 | 2152 | 2153 | 2154 | 2155 | 2156 | 2157 | 2158 | 2159 |
| 87_ | 2160 | 2161 | 2162 | 2163 | 2164 | 2165 | 2166 | 2167 | 2168 | 2169 | 2170 | 2171 | 2172 | 2173 | 2174 | 2175 |
| 88_ | 2176 | 2177 | 2178 | 2179 | 2180 | 2181 | 2182 | 2183 | 2184 | 2185 | 2186 | 2187 | 2188 | 2189 | 2190 | 2191 |
| 89_ | 2192 | 2193 | 2194 | 2195 | 2196 | 2197 | 2198 | 2199 | 2200 | 2201 | 2202 | 2203 | 2204 | 2205 | 2206 | 2207 |
| 8A_ | 2208 | 2209 | 2210 | 2211 | 2212 | 2213 | 2214 | 2215 | 2216 | 2217 | 2218 | 2219 | 2220 | 2221 | 2222 | 2223 |
| 8B_ | 2224 | 2225 | 2226 | 2227 | 2228 | 2229 | 2230 | 2231 | 2232 | 2233 | 2234 | 2235 | 2236 | 2237 | 2238 | 2239 |
| 8C_ | 2240 | 2241 | 2242 | 2243 | 2244 | 2245 | 2246 | 2247 | 2248 | 2249 | 2250 | 2251 | 2252 | 2253 | 2254 | 2255 |
| 8D_ | 2256 | 2257 | 2258 | 2259 | 2260 | 2261 | 2262 | 2263 | 2264 | 2265 | 2266 | 2267 | 2268 | 2269 | 2270 | 2271 |
| 8E_ | 2272 | 2273 | 2274 | 2275 | 2276 | 2277 | 2278 | 2279 | 2280 | 2281 | 2282 | 2283 | 2284 | 2285 | 2286 | 2287 |
| 8F_ | 2288 | 2289 | 2290 | 2291 | 2292 | 2293 | 2294 | 2295 | 2296 | 2297 | 2298 | 2299 | 2300 | 2301 | 2302 | 2303 |
| 90_ | 2304 | 2305 | 2306 | 2307 | 2308 | 2309 | 2310 | 2311 | 2312 | 2313 | 2314 | 2315 | 2316 | 2317 | 2318 | 2319 |
| 91_ | 2320 | 2321 | 2322 | 2323 | 2324 | 2325 | 2326 | 2327 | 2328 | 2329 | 2330 | 2331 | 2332 | 2333 | 2334 | 2335 |
| 92_ | 2336 | 2337 | 2338 | 2339 | 2340 | 2341 | 2342 | 2343 | 2344 | 2345 | 2346 | 2347 | 2348 | 2349 | 2350 | 2351 |
| 93_ | 2352 | 2353 | 2354 | 2355 | 2356 | 2357 | 2358 | 2359 | 2360 | 2361 | 2362 | 2363 | 2364 | 2365 | 2366 | 2367 |
| 94_ | 2368 | 2369 | 2370 | 2371 | 2372 | 2373 | 2374 | 2375 | 2376 | 2377 | 2378 | 2379 | 2380 | 2381 | 2382 | 2383 |
| 95_ | 2384 | 2385 | 2386 | 2387 | 2388 | 2389 | 2390 | 2391 | 2392 | 2393 | 2394 | 2395 | 2396 | 2397 | 2398 | 2399 |
| 96_ | 2400 | 2401 | 2402 | 2403 | 2404 | 2405 | 2406 | 2407 | 2408 | 2409 | 2410 | 2411 | 2412 | 2413 | 2414 | 2415 |
| 97_ | 2416 | 2417 | 2418 | 2419 | 2420 | 2421 | 2422 | 2423 | 2424 | 2425 | 2426 | 2427 | 2428 | 2429 | 2430 | 2431 |
| 98_ | 2432 | 2433 | 2434 | 2435 | 2436 | 2437 | 2438 | 2439 | 2440 | 2441 | 2442 | 2443 | 2444 | 2445 | 2446 | 2447 |
| 99_ | 2448 | 2449 | 2450 | 2451 | 2452 | 2453 | 2454 | 2455 | 2456 | 2457 | 2458 | 2459 | 2460 | 2461 | 2462 | 2463 |
| 9A_ | 2464 | 2465 | 2466 | 2467 | 2468 | 2469 | 2470 | 2471 | 2472 | 2473 | 2474 | 2475 | 2476 | 2477 | 2478 | 2479 |
| 9B_ | 2480 | 2481 | 2482 | 2483 | 2484 | 2485 | 2486 | 2487 | 2488 | 2489 | 2490 | 2491 | 2492 | 2493 | 2494 | 2495 |
| 9C_ | 2496 | 2497 | 2498 | 2499 | 2500 | 2501 | 2502 | 2503 | 2504 | 2505 | 2506 | 2507 | 2508 | 2509 | 2510 | 2511 |
| 9D_ | 2512 | 2513 | 2514 | 2515 | 2516 | 2517 | 2518 | 2519 | 2520 | 2521 | 2522 | 2523 | 2524 | 2525 | 2526 | 2527 |
| 9E_ | 2528 | 2529 | 2530 | 2531 | 2532 | 2533 | 2534 | 2535 | 2536 | 2537 | 2538 | 2539 | 2540 | 2541 | 2542 | 2543 |
| 9F_ | 2544 | 2545 | 2546 | 2547 | 2548 | 2549 | 2550 | 2551 | 2552 | 2553 | 2554 | 2555 | 2556 | 2557 | 2558 | 2559 |
| A0_ | 2560 | 2561 | 2562 | 2563 | 2564 | 2565 | 2566 | 2567 | 2568 | 2569 | 2570 | 2571 | 2572 | 2573 | 2574 | 2575 |
| A1_ | 2576 | 2577 | 2578 | 2579 | 2580 | 2581 | 2582 | 2583 | 2584 | 2585 | 2586 | 2587 | 2588 | 2589 | 2590 | 2591 |
| A2_ | 2592 | 2593 | 2594 | 2595 | 2596 | 2597 | 2598 | 2599 | 2600 | 2601 | 2602 | 2603 | 2604 | 2605 | 2606 | 2607 |
| A3_ | 2608 | 2609 | 2610 | 2611 | 2612 | 2613 | 2614 | 2615 | 2616 | 2617 | 2618 | 2619 | 2620 | 2621 | 2622 | 2623 |
| A4_ | 2624 | 2625 | 2626 | 2627 | 2628 | 2629 | 2630 | 2631 | 2632 | 2633 | 2634 | 2635 | 2636 | 2637 | 2638 | 2639 |
| A5_ | 2640 | 2641 | 2642 | 2643 | 2644 | 2645 | 2646 | 2647 | 2648 | 2649 | 2650 | 2651 | 2652 | 2653 | 2654 | 2655 |
| A6_ | 2656 | 2657 | 2658 | 2659 | 2660 | 2661 | 2662 | 2663 | 2664 | 2665 | 2666 | 2667 | 2668 | 2669 | 2670 | 2671 |
| A7_ | 2672 | 2673 | 2674 | 2675 | 2676 | 2677 | 2678 | 2679 | 2680 | 2681 | 2682 | 2683 | 2684 | 2685 | 2686 | 2687 |
| A8_ | 2688 | 2689 | 2690 | 2691 | 2692 | 2693 | 2694 | 2695 | 2696 | 2697 | 2698 | 2699 | 2700 | 2701 | 2702 | 2703 |
| A9_ | 2704 | 2705 | 2706 | 2707 | 2708 | 2709 | 2710 | 2711 | 2712 | 2713 | 2714 | 2715 | 2716 | 2717 | 2718 | 2719 |
| AA_ | 2720 | 2721 | 2722 | 2723 | 2724 | 2725 | 2726 | 2727 | 2728 | 2729 | 2730 | 2731 | 2732 | 2733 | 2734 | 2735 |
| AB_ | 2736 | 2737 | 2738 | 2739 | 2740 | 2741 | 2742 | 2743 | 2744 | 2745 | 2746 | 2747 | 2748 | 2749 | 2750 | 2751 |
| AC_ | 2752 | 2753 | 2754 | 2755 | 2756 | 2757 | 2758 | 2759 | 2760 | 2761 | 2762 | 2763 | 2764 | 2765 | 2766 | 2767 |
| AD_ | 2768 | 2769 | 2770 | 2771 | 2772 | 2773 | 2774 | 2775 | 2776 | 2777 | 2778 | 2779 | 2780 | 2781 | 2782 | 2783 |
| AE_ | 2784 | 2785 | 2786 | 2787 | 2788 | 2789 | 2790 | 2791 | 2792 | 2793 | 2794 | 2795 | 2796 | 2797 | 2798 | 2799 |
| AF_ | 2800 | 2801 | 2802 | 2803 | 2804 | 2805 | 2806 | 2807 | 2808 | 2809 | 2810 | 2811 | 2812 | 2813 | 2814 | 2815 |
| B0_ | 2816 | 2817 | 2818 | 2819 | 2820 | 2821 | 2822 | 2823 | 2824 | 2825 | 2826 | 2827 | 2828 | 2829 | 2830 | 2831 |
| B1_ | 2832 | 2833 | 2834 | 2835 | 2836 | 2837 | 2838 | 2839 | 2840 | 2841 | 2842 | 2843 | 2844 | 2845 | 2846 | 2847 |
| B2_ | 2848 | 2849 | 2850 | 2851 | 2852 | 2853 | 2854 | 2855 | 2856 | 2857 | 2858 | 2859 | 2860 | 2861 | 2862 | 2863 |
| B3_ | 2864 | 2865 | 2866 | 2867 | 2868 | 2869 | 2870 | 2871 | 2872 | 2873 | 2874 | 2875 | 2876 | 2877 | 2878 | 2879 |
| B4_ | 2880 | 2881 | 2882 | 2883 | 2884 | 2885 | 2886 | 2887 | 2888 | 2889 | 2890 | 2891 | 2892 | 2893 | 2894 | 2895 |
| B5_ | 2896 | 2897 | 2898 | 2899 | 2900 | 2901 | 2902 | 2903 | 2904 | 2905 | 2906 | 2907 | 2908 | 2909 | 2910 | 2911 |
| B6_ | 2912 | 2913 | 2914 | 2915 | 2916 | 2917 | 2918 | 2919 | 2920 | 2921 | 2922 | 2923 | 2924 | 2925 | 2926 | 2927 |
| B7_ | 2928 | 2929 | 2930 | 2931 | 2932 | 2933 | 2934 | 2935 | 2936 | 2937 | 2938 | 2939 | 2940 | 2941 | 2942 | 2943 |
| B8_ | 2944 | 2945 | 2946 | 2947 | 2948 | 2949 | 2950 | 2951 | 2952 | 2953 | 2954 | 2955 | 2956 | 2957 | 2958 | 2959 |
| B9_ | 2960 | 2961 | 2962 | 2963 | 2964 | 2965 | 2966 | 2967 | 2968 | 2969 | 2970 | 2971 | 2972 | 2973 | 2974 | 2975 |
| BA_ | 2976 | 2977 | 2978 | 2979 | 2980 | 2981 | 2982 | 2983 | 2984 | 2985 | 2986 | 2987 | 2988 | 2989 | 2990 | 2991 |
| BB_ | 2992 | 2993 | 2994 | 2995 | 2996 | 2997 | 2998 | 2999 | 3000 | 3001 | 3002 | 3003 | 3004 | 3005 | 3006 | 3007 |
| BC_ | 3008 | 3009 | 3010 | 3011 | 3012 | 3013 | 3014 | 3015 | 3016 | 3017 | 3018 | 3019 | 3020 | 3021 | 3022 | 3023 |
| BD_ | 3024 | 3025 | 3026 | 3027 | 3028 | 3029 | 3030 | 3031 | 3032 | 3033 | 3034 | 3035 | 3036 | 3037 | 3038 | 3039 |
| BE_ | 3040 | 3041 | 3042 | 3043 | 3044 | 3045 | 3046 | 3047 | 3048 | 3049 | 3050 | 3051 | 3052 | 3053 | 3054 | 3055 |
| BF_ | 3056 | 3057 | 3058 | 3059 | 3060 | 3061 | 3062 | 3063 | 3064 | 3065 | 3066 | 3067 | 3068 | 3069 | 3070 | 3071 |



LzLabs GmbH. Ex. 1008-974

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C0_ | 3072 | 3073 | 3074 | 3075 | 3076 | 3077 | 3078 | 3079 | 3080 | 3081 | 3082 | 3083 | 3084 | 3085 | 3086 | 3087 |
| C1_ | 3088 | 3089 | 3090 | 3091 | 3092 | 3093 | 3094 | 3095 | 3096 | 3097 | 3098 | 3099 | 3100 | 3101 | 3102 | 3103 |
| C2_ | 3104 | 3105 | 3106 | 3107 | 3108 | 3109 | 3110 | 3111 | 3112 | 3113 | 3114 | 3115 | 3116 | 3117 | 3118 | 3119 |
| C3_ | 3120 | 3121 | 3122 | 3123 | 3124 | 3125 | 3126 | 3127 | 3128 | 3129 | 3130 | 3131 | 3132 | 3133 | 3134 | 3135 |
| C4_ | 3136 | 3137 | 3138 | 3139 | 3140 | 3141 | 3142 | 3143 | 3144 | 3145 | 3146 | 3147 | 3148 | 3149 | 3150 | 3151 |
| C5_ | 3152 | 3153 | 3154 | 3155 | 3156 | 3157 | 3158 | 3159 | 3160 | 3161 | 3162 | 3163 | 3164 | 3165 | 3166 | 3167 |
| C6_ | 3168 | 3169 | 3170 | 3171 | 3172 | 3173 | 3174 | 3175 | 3176 | 3177 | 3178 | 3179 | 3180 | 3181 | 3182 | 3183 |
| C7_ | 3184 | 3185 | 3186 | 3187 | 3188 | 3189 | 3190 | 3191 | 3192 | 3193 | 3194 | 3195 | 3196 | 3197 | 3198 | 3199 |
| C8_ | 3200 | 3201 | 3202 | 3203 | 3204 | 3205 | 3206 | 3207 | 3208 | 3209 | 3210 | 3211 | 3212 | 3213 | 3214 | 3215 |
| C9_ | 3216 | 3217 | 3218 | 3219 | 3220 | 3221 | 3222 | 3223 | 3224 | 3225 | 3226 | 3227 | 3228 | 3229 | 3230 | 3231 |
| CA_ | 3232 | 3233 | 3234 | 3235 | 3236 | 3237 | 3238 | 3239 | 3240 | 3241 | 3242 | 3243 | 3244 | 3245 | 3246 | 3247 |
| CB_ | 3248 | 3249 | 3250 | 3251 | 3252 | 3253 | 3254 | 3255 | 3256 | 3257 | 3258 | 3259 | 3260 | 3261 | 3262 | 3263 |
| CC_ | 3264 | 3265 | 3266 | 3267 | 3268 | 3269 | 3270 | 3271 | 3272 | 3273 | 3274 | 3275 | 3276 | 3277 | 3278 | 3279 |
| CD_ | 3280 | 3281 | 3282 | 3283 | 3284 | 3285 | 3286 | 3287 | 3288 | 3289 | 3290 | 3291 | 3292 | 3293 | 3294 | 3295 |
| CE_ | 3296 | 3297 | 3298 | 3299 | 3300 | 3301 | 3302 | 3303 | 3304 | 3305 | 3306 | 3307 | 3308 | 3309 | 3310 | 3311 |
| CF_ | 3312 | 3313 | 3314 | 3315 | 3316 | 3317 | 3318 | 3319 | 3320 | 3321 | 3322 | 3323 | 3324 | 3325 | 3326 | 3327 |
| D0_ | 3328 | 3329 | 3330 | 3331 | 3332 | 3333 | 3334 | 3335 | 3336 | 3337 | 3338 | 3339 | 3340 | 3341 | 3342 | 3343 |
| D1_ | 3344 | 3345 | 3346 | 3347 | 3348 | 3349 | 3350 | 3351 | 3352 | 3353 | 3354 | 3355 | 3356 | 3357 | 3358 | 3359 |
| D2_ | 3360 | 3361 | 3362 | 3363 | 3364 | 3365 | 3366 | 3367 | 3368 | 3369 | 3370 | 3371 | 3372 | 3373 | 3374 | 3375 |
| D3_ | 3376 | 3377 | 3378 | 3379 | 3380 | 3381 | 3382 | 3383 | 3384 | 3385 | 3386 | 3387 | 3388 | 3389 | 3390 | 3391 |
| D4_ | 3392 | 3393 | 3394 | 3395 | 3396 | 3397 | 3398 | 3399 | 3400 | 3401 | 3402 | 3403 | 3404 | 3405 | 3406 | 3407 |
| D5_ | 3408 | 3409 | 3410 | 3411 | 3412 | 3413 | 3414 | 3415 | 3416 | 3417 | 3418 | 3419 | 3420 | 3421 | 3422 | 3423 |
| D6_ | 3424 | 3425 | 3426 | 3427 | 3428 | 3429 | 3430 | 3431 | 3432 | 3433 | 3434 | 3435 | 3436 | 3437 | 3438 | 3439 |
| D7_ | 3440 | 3441 | 3442 | 3443 | 3444 | 3445 | 3446 | 3447 | 3448 | 3449 | 3450 | 3451 | 3452 | 3453 | 3454 | 3455 |
| D8_ | 3456 | 3457 | 3458 | 3459 | 3460 | 3461 | 3462 | 3463 | 3464 | 3465 | 3466 | 3467 | 3468 | 3469 | 3470 | 3471 |
| D9_ | 3472 | 3473 | 3474 | 3475 | 3476 | 3477 | 3478 | 3479 | 3480 | 3481 | 3482 | 3483 | 3484 | 3485 | 3486 | 3487 |
| DA_ | 3488 | 3489 | 3490 | 3491 | 3492 | 3493 | 3494 | 3495 | 3496 | 3497 | 3498 | 3499 | 3500 | 3501 | 3502 | 3503 |
| DB_ | 3504 | 3505 | 3506 | 3507 | 3508 | 3509 | 3510 | 3511 | 3512 | 3513 | 3514 | 3515 | 3516 | 3517 | 3518 | 3519 |
| DC_ | 3520 | 3521 | 3522 | 3523 | 3524 | 3525 | 3526 | 3527 | 3528 | 3529 | 3530 | 3531 | 3532 | 3533 | 3534 | 3535 |
| DD_ | 3536 | 3537 | 3538 | 3539 | 3540 | 3541 | 3542 | 3543 | 3544 | 3545 | 3546 | 3547 | 3548 | 3549 | 3550 | 3551 |
| DE_ | 3552 | 3553 | 3554 | 3555 | 3556 | 3557 | 3558 | 3559 | 3560 | 3561 | 3562 | 3563 | 3564 | 3565 | 3566 | 3567 |
| DF_ | 3568 | 3569 | 3570 | 3571 | 3572 | 3573 | 3574 | 3575 | 3576 | 3577 | 3578 | 3579 | 3580 | 3581 | 3582 | 3583 |
| E0_ | 3584 | 3585 | 3586 | 3587 | 3588 | 3589 | 3590 | 3591 | 3592 | 3593 | 3594 | 3595 | 3596 | 3597 | 3598 | 3599 |
| E1_ | 3600 | 3601 | 3602 | 3603 | 3604 | 3605 | 3606 | 3607 | 3608 | 3609 | 3610 | 3611 | 3612 | 3613 | 3614 | 3615 |
| E2_ | 3616 | 3617 | 3618 | 3619 | 3620 | 3621 | 3622 | 3623 | 3624 | 3625 | 3626 | 3627 | 3628 | 3629 | 3630 | 3631 |
| E3_ | 3632 | 3633 | 3634 | 3635 | 3636 | 3637 | 3638 | 3639 | 3640 | 3641 | 3642 | 3643 | 3644 | 3645 | 3646 | 3647 |
| E4_ | 3648 | 3649 | 3650 | 3651 | 3652 | 3653 | 3654 | 3655 | 3656 | 3657 | 3658 | 3659 | 3660 | 3661 | 3662 | 3663 |
| E5_ | 3664 | 3665 | 3666 | 3667 | 3668 | 3669 | 3670 | 3671 | 3672 | 3673 | 3674 | 3675 | 3676 | 3677 | 3678 | 3679 |
| E6_ | 3680 | 3681 | 3682 | 3683 | 3684 | 3685 | 3686 | 3687 | 3688 | 3689 | 3690 | 3691 | 3692 | 3693 | 3694 | 3695 |
| E7_ | 3696 | 3697 | 3698 | 3699 | 3700 | 3701 | 3702 | 3703 | 3704 | 3705 | 3706 | 3707 | 3708 | 3709 | 3710 | 3711 |
| E8_ | 3712 | 3713 | 3714 | 3715 | 3716 | 3717 | 3718 | 3719 | 3720 | 3721 | 3722 | 3723 | 3724 | 3725 | 3726 | 3727 |
| E9_ | 3728 | 3729 | 3730 | 3731 | 3732 | 3733 | 3734 | 3735 | 3736 | 3737 | 3738 | 3739 | 3740 | 3741 | 3742 | 3743 |
| EA_ | 3744 | 3745 | 3746 | 3747 | 3748 | 3749 | 3750 | 3751 | 3752 | 3753 | 3754 | 3755 | 3756 | 3757 | 3758 | 3759 |
| EB_ | 3760 | 3761 | 3762 | 3763 | 3764 | 3765 | 3766 | 3767 | 3768 | 3769 | 3770 | 3771 | 3772 | 3773 | 3774 | 3775 |
| EC_ | 3776 | 3777 | 3778 | 3779 | 3780 | 3781 | 3782 | 3783 | 3784 | 3785 | 3786 | 3787 | 3788 | 3789 | 3790 | 3791 |
| ED_ | 3792 | 3793 | 3794 | 3795 | 3796 | 3797 | 3798 | 3799 | 3800 | 3801 | 3802 | 3803 | 3804 | 3805 | 3806 | 3807 |
| EE_ | 3808 | 3809 | 3810 | 3811 | 3812 | 3813 | 3814 | 3815 | 3816 | 3817 | 3818 | 3819 | 3820 | 3821 | 3822 | 3823 |
| EF_ | 3824 | 3825 | 3826 | 3827 | 3828 | 3829 | 3830 | 3831 | 3832 | 3833 | 3834 | 3835 | 3836 | 3837 | 3838 | 3839 |
| F0_ | 3840 | 3841 | 3842 | 3843 | 3844 | 3845 | 3846 | 3847 | 3848 | 3849 | 3850 | 3851 | 3852 | 3853 | 3854 | 3855 |
| F1_ | 3856 | 3857 | 3858 | 3859 | 3860 | 3861 | 3862 | 3863 | 3864 | 3865 | 3866 | 3867 | 3868 | 3869 | 3870 | 3871 |
| F2_ | 3872 | 3873 | 3874 | 3875 | 3876 | 3877 | 3878 | 3879 | 3880 | 3881 | 3882 | 3883 | 3884 | 3885 | 3886 | 3887 |
| F3_ | 3888 | 3889 | 3890 | 3891 | 3892 | 3893 | 3894 | 3895 | 3896 | 3897 | 3898 | 3899 | 3900 | 3901 | 3902 | 3903 |
| F4_ | 3904 | 3905 | 3906 | 3907 | 3908 | 3909 | 3910 | 3911 | 3912 | 3913 | 3914 | 3915 | 3916 | 3917 | 3918 | 3919 |
| F5_ | 3920 | 3921 | 3922 | 3923 | 3924 | 3925 | 3926 | 3927 | 3928 | 3929 | 3930 | 3931 | 3932 | 3933 | 3934 | 3935 |
| F6_ | 3936 | 3937 | 3938 | 3939 | 3940 | 3941 | 3942 | 3943 | 3944 | 3945 | 3946 | 3947 | 3948 | 3949 | 3950 | 3951 |
| F7_ | 3952 | 3953 | 3954 | 3955 | 3956 | 3957 | 3958 | 3959 | 3960 | 3961 | 3962 | 3963 | 3964 | 3965 | 3966 | 3967 |
| F8_ | 3968 | 3969 | 3970 | 3971 | 3972 | 3973 | 3974 | 3975 | 3976 | 3977 | 3978 | 3979 | 3980 | 3981 | 3982 | 3983 |
| F9_ | 3984 | 3985 | 3986 | 3987 | 3988 | 3989 | 3990 | 3991 | 3992 | 3993 | 3994 | 3995 | 3996 | 3997 | 3998 | 3999 |
| FA_ | 4000 | 4001 | 4002 | 4003 | 4004 | 4005 | 4006 | 4007 | 4008 | 4009 | 4010 | 4011 | 4012 | 4013 | 4014 | 4015 |
| FB_ | 4016 | 4017 | 4018 | 4019 | 4020 | 4021 | 4022 | 4023 | 4024 | 4025 | 4026 | 4027 | 4028 | 4029 | 4030 | 4031 |
| FC_ | 4032 | 4033 | 4034 | 4035 | 4036 | 4037 | 4038 | 4039 | 4040 | 4041 | 4042 | 4043 | 4044 | 4045 | 4046 | 4047 |
| FD_ | 4048 | 4049 | 4050 | 4051 | 4052 | 4053 | 4054 | 4055 | 4056 | 4057 | 4058 | 4059 | 4060 | 4061 | 4062 | 4063 |
| FE_ | 4064 | 4065 | 4066 | 4067 | 4068 | 4069 | 4070 | 4071 | 4072 | 4073 | 4074 | 4075 | 4076 | 4077 | 4078 | 4079 |
| FF_ | 4080 | 4081 | 4082 | 4083 | 4084 | 4085 | 4086 | 4087 | 4088 | 4089 | 4090 | 4091 | 4092 | 4093 | 4094 | 4095 |


LzLabs GmbH. Ex. 1008-975

## *Conversion Table: Hexadecimal and Decimal Integers*

| Hex | Decimal | Hex | Decimal | Hex | Decimal | Hex | Decimal | Hex | Decimal | Hex | Decimal | Hex | Decimal | Hex | Decimal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 268,435,456 | 1 | 16,777,216 | 1 | 1,048,576 | 1 | 65,536 | 1 | 4,096 | 1 | 256 | 1 | 16 | 1 | 1 |
| 2 | 536,870,912 | 2 | 33,554,432 | 2 | 2,097,152 | 2 | 131,072 | 2 | 8,192 | 2 | 512 | 2 | 32 | 2 | 2 |
| 3 | 805,306,368 | 3 | 50,331,648 | 3 | 3,145,728 | 3 | 196,608 | 3 | 12,288 | 3 | 768 | 3 | 48 | 3 | 3 |
| 4 | 1,073,741,824 | 4 | 67,108,864 | 4 | 4,194,304 | 4 | 262,144 | 4 | 16,384 | 4 | 1,024 | 4 | 64 | 4 | 4 |
| 5 | 1,342,177,280 | 5 | 83,886,080 | 5 | 5,242,880 | 5 | 327,680 | 5 | 20,480 | 5 | 1,280 | 5 | 80 | 5 | 5 |
| 6 | 1,610,612,736 | 6 | 100,663,296 | 6 | 6,291,456 | 6 | 393,216 | 6 | 24,576 | 6 | 1,536 | 6 | 96 | 6 | 6 |
| 7 | 1,879,048,192 | 7 | 117,440,512 | 7 | 7,340,032 | 7 | 458,752 | 7 | 28,672 | 7 | 1,792 | 7 | 112 | 7 | 7 |
| 8 | 2,147,483,648 | 8 | 134,217,728 | 8 | 8,388,608 | 8 | 524,288 | 8 | 32,768 | 8 | 2,048 | 8 | 128 | 8 | 8 |
| 9 | 2,415,919,104 | 9 | 150,994,944 | 9 | 9,437,184 | 9 | 589,824 | 9 | 36,864 | 9 | 2,304 | 9 | 144 | 9 | 9 |
| A | 2,684,354,560 | A | 167,772,160 | A | 10,485,760 | A | 655,360 | A | 40,960 | A | 2,560 | A | 160 | A | 10 |
| B | 2,952,790,016 | B | 184,549,376 | B | 11,534,336 | B | 720,896 | B | 45,056 | B | 2,816 | B | 176 | B | 11 |
| C | 3,221,225,472 | C | 201,326,592 | C | 12,582,912 | C | 786,432 | C | 49,152 | C | 3,072 | C | 192 | C | 12 |
| D | 3,489,660,928 | D | 218,103,808 | D | 13,631,488 | D | 851,968 | D | 53,248 | D | 3,328 | D | 208 | D | 13 |
| E | 3,758,096,384 | E | 234,881,024 | E | 14,680,064 | E | 917,504 | E | 57,344 | E | 3,584 | E | 224 | E | 14 |
| F | 4,026,531,840 | F | 251,658,240 | F | 15,728,640 | F | 983,040 | F | 61,440 | F | 3,840 | F | 240 | F | 15 |
| 8 | | 7 | | 6 | | 5 | | 4 | | 3 | | 2 | | 1 | |

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of the decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position.
4. Add the numbers selected from the table to form the decimal number.

EXAMPLE

| Conversion of Hexadecimal Value | D34 |
|---|---|
| 1. D | 3328 |
| 2. 3 | 48 |
| 3. 4 | + 4 |
| 4. Decimal | 3380 |

To convert integer numbers greater than the capacity of the table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D34_{16} = 3380_{10}$

D = 13
x16
208
3 = +3
211
x16
3376
4 = +4
3380

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.
   (b) Record the hexadecimal of the column containing the selected number.
   (c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1(c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder)
3. Using the remainder from step 2, repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine the terms to form the hexadecimal number.

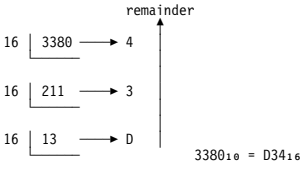
EXAMPLE

| Conversion of Decimal Value | 3380 |
|---|---|
| 1. D | -3328 |
| | 52 |
| 2. 3 | -48 |
| | 4 |
| 3. 4 | -4 |
| 4. Hexadecimal | D34 |

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order

Example: $3380_{10} = X_{16}$




$3380_{10} = D34_{16}$

POWERS OF 16 TABLE

Example: $268,435,456_{10} = (2.68435456 \times 10^8)_{10} = 1000\ 0000_{16} = (10^7)_{16}$

| $16^n$ | n |
|---|---|
| 1 | 0 |
| 16 | 1 |
| 256 | 2 |
| 4 096 | 3 |
| 65 536 | 4 |
| 1 048 576 | 5 |
| 16 777 216 | 6 |
| 268 435 456 | 7 |
| 4 294 967 296 | 8 |
| 68 719 476 736 | 9 |
| 1 099 511 627 776 | 10 = A |
| 17 592 186 044 416 | 11 = B |
| 281 474 976 710 656 | 12 = C |
| 4 503 599 627 370 496 | 13 = D |
| 72 057 594 037 927 936 | 14 = E |
| 1 152 921 504 606 846 976 | 15 = F |
| Decimal Values | |



LzLabs GmbH. Ex. 1008-976

### *Conversion Table: Hexadecimal and Decimal Fractions*

| HALFWORD | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BYTE | | | | | BYTE | | | | | | | | |
| Bits 0123 | | 4567 | | | 0123 | | | | 4567 | | | | |
| Hex | Decimal | Hex | Decimal | | Hex | Decimal | | | Hex | Decimal Equivalent | | | |
| .0 | .0000 | .00 | .0000 | 0000 | .000 | .0000 | 0000 | 0000 | .0000 | .0000 | 0000 | 0000 | 0000 |
| .1 | .0625 | .01 | .0039 | 0625 | .001 | .0002 | 4414 | 0625 | .0001 | .0000 | 1525 | 8789 | 0625 |
| .2 | .1250 | .02 | .0078 | 1250 | .002 | .0004 | 8828 | 1250 | .0002 | .0000 | 3051 | 7578 | 1250 |
| .3 | .1875 | .03 | .0117 | 1875 | .003 | .0007 | 3242 | 1875 | .0003 | .0000 | 4577 | 6367 | 1875 |
| .4 | .2500 | .04 | .0156 | 2500 | .004 | .0009 | 7656 | 2500 | .0004 | .0000 | 6103 | 5156 | 2500 |
| .5 | .3125 | .05 | .0195 | 3125 | .005 | .0012 | 2070 | 3125 | .0005 | .0000 | 7629 | 3945 | 3125 |
| .6 | .3750 | .06 | .0234 | 3750 | .006 | .0014 | 6484 | 3750 | .0006 | .0000 | 9155 | 2734 | 3750 |
| .7 | .4375 | .07 | .0273 | 4375 | .007 | .0017 | 0898 | 4375 | .0007 | .0001 | 0681 | 1523 | 4375 |
| .8 | .5000 | .08 | .0312 | 5000 | .008 | .0019 | 5312 | 5000 | .0008 | .0001 | 2207 | 0312 | 5000 |
| .9 | .5625 | .09 | .0351 | 5625 | .009 | .0021 | 9726 | 5625 | .0009 | .0001 | 3732 | 9101 | 5625 |
| .A | .6250 | .0A | .0390 | 6250 | .00A | .0024 | 4140 | 6250 | .000A | .0001 | 5258 | 7890 | 6250 |
| .B | .6875 | .0B | .0429 | 6875 | .00B | .0026 | 8554 | 6875 | .000B | .0001 | 6784 | 6679 | 6875 |
| .C | .7500 | .0C | .0468 | 7500 | .00C | .0029 | 2968 | 7500 | .000C | .0001 | 8310 | 5468 | 7500 |
| .D | .8125 | .0D | .0507 | 8125 | .00D | .0031 | 7382 | 8125 | .000D | .0001 | 9836 | 4257 | 8125 |
| .E | .8750 | .0E | .0546 | 8750 | .00E | .0034 | 1796 | 8750 | .000E | .0002 | 1362 | 3046 | 8750 |
| .F | .9375 | .0F | .0585 | 9375 | .00F | .0036 | 6210 | 9375 | .000F | .0002 | 2888 | 1835 | 9375 |
| 1 | | 2 | | | 3 | | | | 4 | | | | |

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

```
Find .A   in position 1    .6250

Find .0B  in position 2    .0429 6825

Find .00C in position 3   +.0029 2968 7500

       .ABC is equal to    .6708 9843 7500
```

TO CONVERT .13 DECIMAL TO HEXADECIMAL

```
1.  Find .1250 next lowest to        .1300
              subtract            - .1250              = .2 Hex

2.  Find .0039 0625 next lowest to   .0050 0000
                                  - .0039 0625         = .01

3.  Find .0009 7656 2500             .0010 9375 0000
                                  - .0009 7656 2500    = .004

4.  Find .0001 0681 1523 4375        .0001 1718 7500 0000
                                  - .0001 0681 1523 4375 = .0007

                                     .0000 1037 5976 5625 = .2147 Hex

5. .13 Decimal is approximately equal to ------------------------^
```

To convert fractions beyond the capacity of the table, use techniques below:

HEXADECIMAL TO FRACTION DECIMAL

Convert the hexadecimal fraction to its decimal equivalent using the same technique as for integer numbers. Divide the results by $16^n$ (n is the number of fraction positions).

Example: $.8A7 = .540771_{10}$

$8A7_{16} = 2215_{10}$ .540771

$16^3 = 4096$ 4096 | 2215.000000

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

.5408

```
8 <-- [ 8] x16
              .6528

A <-- [10] x16
              .4448

7 <-- [ 7] x16
              .1168
```



LzLabs GmbH. Ex. 1008-977

### *Hexadecimal Addition and Subtraction Table*

Example: 6 + 2 = 8, 8 - 2 = 6, and 8 - 6 = 2

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 |
| 2 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 |
| 3 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 |
| 4 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 |
| 5 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 |
| 6 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 8 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 9 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A |
| C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B |
| D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C |
| E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |

### *Hexadecimal Multiplication Table*

Example: 2 x 4 = 08, F x 2 = 1E

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| 2 | 02 | 04 | 06 | 08 | 0A | 0C | 0E | 10 | 12 | 14 | 16 | 18 | 1A | 1C | 1E |
| 3 | 03 | 06 | 09 | 0C | 0F | 12 | 15 | 18 | 1B | 1E | 21 | 24 | 27 | 2A | 2D |
| 4 | 04 | 08 | 0C | 10 | 14 | 18 | 1C | 20 | 24 | 28 | 2C | 30 | 34 | 38 | 3C |
| 5 | 05 | 0A | 0F | 14 | 19 | 1E | 23 | 28 | 2D | 32 | 37 | 3C | 41 | 46 | 4B |
| 6 | 06 | 0C | 12 | 18 | 1E | 24 | 2A | 30 | 36 | 3C | 42 | 48 | 4E | 54 | 5A |
| 7 | 07 | 0E | 15 | 1C | 23 | 2A | 31 | 38 | 3F | 46 | 4D | 54 | 5B | 62 | 69 |
| 8 | 08 | 10 | 18 | 20 | 28 | 30 | 38 | 40 | 48 | 50 | 58 | 60 | 68 | 70 | 78 |
| 9 | 09 | 12 | 1B | 24 | 2D | 36 | 3F | 48 | 51 | 5A | 63 | 6C | 75 | 7E | 87 |
| A | 0A | 14 | 1E | 28 | 32 | 3C | 46 | 50 | 5A | 64 | 6E | 78 | 82 | 8C | 96 |
| B | 0B | 16 | 21 | 2C | 37 | 42 | 4D | 58 | 63 | 6E | 79 | 84 | 8F | 9A | A5 |
| C | 0C | 18 | 24 | 30 | 3C | 48 | 54 | 60 | 6C | 78 | 84 | 90 | 9C | A8 | B4 |
| D | 0D | 1A | 27 | 34 | 41 | 4E | 5B | 68 | 75 | 82 | 8F | 9C | A9 | B6 | C3 |
| E | 0E | 1C | 2A | 38 | 46 | 54 | 62 | 70 | 7E | 8C | 9A | A8 | B6 | C4 | D2 |
| F | 0F | 1E | 2D | 3C | 4B | 5A | 69 | 78 | 87 | 96 | A5 | B4 | C3 | D2 | E1 |



LzLabs GmbH. Ex. 1008-978

# Appendix I. EBCDIC and Other Codes

The following table shows the Extended Binary-Coded-Decimal Interchange Code (EBCDIC) and other codes. Details are in the notes on page I-4.

| Dec | Hex | EBCDIC | AS- ISO (1) CII -8 | AS- ISO (1) IBM-PC | BookMaster Symbol Names(2) |
|---|---|---|---|---|---|
| 0 | 00 | NUL | NUL | NUL | |
| 1 | 01 | SOH | SOH | SOH ☺ | face |
| 2 | 02 | STX | STX | STX ☻ | FACE |
| 3 | 03 | ETX | ETX | ETX ♥ | HEART |
| 4 | 04 | SEL | EOT | EOT ♦ | DIAMOND |
| 5 | 05 | HT | ENQ | ENQ ♣ | CLUB |
| 6 | 06 | RNL | ACK | ACK ♠ | SPADE |
| 7 | 07 | DEL | BEL | BEL • | bullet |
| 8 | 08 | GE | BS | BS ◘ | revbul |
| 9 | 09 | SPS | HT | HT ○ | circle |
| 10 | 0A | RPT | LF | LF ◙ | revcir |
| 11 | 0B | VT | VT | VT ♂ | male |
| 12 | 0C | FF | FF | FF ♀ | female |
| 13 | 0D | CR | CR | CR ♪ | note18 |
| 14 | 0E | SO | SO | SO ♫ | note1616 |
| 15 | 0F | SI | SI | SI ☼ | sun |
| 16 | 10 | DLE | DLE | DLE ► | rahead |
| 17 | 11 | DC1 | DC1 | DC1 ◄ | lahead |
| 18 | 12 | DC2 | DC2 | DC2 ↕ | udarrow |
| 19 | 13 | DC3 | DC3 | DC3 ‼ | dblxclam |
| 20 | 14 | RES/ENP | DC4 | DC4 ¶ | par |
| 21 | 15 | NL | NAK | NAK § | section |
| 22 | 16 | BS | SYN | SYN | overline |
| 23 | 17 | POC | ETB | ETB ↨ | udarrowus |
| 24 | 18 | CAN | CAN | CAN ↑ | uarrow |
| 25 | 19 | EM | EM | EM ↓ | darrow |
| 26 | 1A | UBS | SUB | IFS → | rarrow |
| 27 | 1B | CU1 | ESC | ESC ← | larrow |
| 28 | 1C | IFS | FS | DEL ∟ | lnotusd |
| 29 | 1D | IGS | GS | GS ↔ | lrarrow |
| 30 | 1E | IRS | RS | RS ▲ | uahead |
| 31 | 1F | ITB/IUS | US | US ▼ | dahead |
| 32 | 20 | DS | SP | SP | |
| 33 | 21 | SOS | ! | ! | xclam |
| 34 | 22 | FS | " | " | sdq |
| 35 | 23 | WUS | # | # | numsign |
| 36 | 24 | BYP/INP | $ | $ | dollar |
| 37 | 25 | LF | % | % | percent |
| 38 | 26 | ETB | & | & | amp |
| 39 | 27 | ESC | ' | ' | ssq(3) |
| 40 | 28 | SA | ( | ( | lpar |
| 41 | 29 | SFE | ) | ) | rpar |
| 42 | 2A | SM/SW | * | * | asterisk |
| 43 | 2B | CSP | + | + | plus |
| 44 | 2C | MFA | , | , | comma |
| 45 | 2D | ENQ | – | – | hyphen or minus |
| 46 | 2E | ACK | . | . | period |
| 47 | 2F | BEL | / | / | divslash or slash |
| 48 | 30 | | 0 | 0 | |
| 49 | 31 | | 1 | 1 | |
| 50 | 32 | SYN | 2 | 2 | |
| 51 | 33 | IR | 3 | 3 | |
| 52 | 34 | PP | 4 | 4 | |
| 53 | 35 | TRN | 5 | 5 | |
| 54 | 36 | NBS | 6 | 6 | |
| 55 | 37 | EOT | 7 | 7 | |
| 56 | 38 | SBS | 8 | 8 | |
| 57 | 39 | IT | 9 | 9 | |
| 58 | 3A | RFF | : | : | colon |
| 59 | 3B | CU3 | ; | ; | semi |
| 60 | 3C | DC4 | < | < | lt |
| 61 | 3D | NAK | = | = | eq |
| 62 | 3E | | > | > | gt |
| 63 | 3F | SUB | ? | ? | quest |

**Control-Character Representations**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ACK | Acknowledge | ENP | Enable Presentation | ITB | Intermediate Transmission Block | SBS | Subscript |
| BEL | Bell | ENQ | Enquiry | | | SEL | Select |
| BS | Backspace | EO | Eight Ones | IUS | International Unit Separator | SFE | Start Field Extended |
| BYP | Bypass | EOT | End of Transmission | LF | Line Feed | SI | Shift In |
| CAN | Cancel | ESC | Escape | MFA | Modify Field Attribute | SM | Set Mode |
| CR | Carriage Return | ETB | End of Transmission Block | NAK | Negative Acknowledge | SO | Shift Out |
| CSP | Control Sequence Prefix | ETX | End of Text | NBS | Numeric Backspace | SOH | Start of Heading |
| CU1 | Customer Use 1 | FF | Form Feed | NL | New Line | SOS | Start of Significance |
| CU3 | Customer Use 3 | FS | Field Separator | NUL | Null | SPS | Superscript |
| DC1 | Device Control 1 | GE | Graphic Escape | POC | Program-Operator Communication | STX | Start of Text |
| DC2 | Device Control 2 | HT | Horizontal Tab | | | SUB | Substitute |
| DC3 | Device Control 3 | IFS | Interchange File Separator | PP | Presentation Position | SW | Switch |
| DC4 | Device Control 4 | IGS | Interchange Group Separator | RES | Restore | SYN | Synchronous Idle |
| DEL | Delete | INP | Inhibit Presentation | RFF | Required Form Feed | TRN | Transparent |
| DLE | Data Link Escape | IR | Index Return | RNL | Required New Line | UBS | Unit Backspace |
| DS | Digit Select | IRS | Interchange Record Separator | RPT | Repeat | VT | Vertical Tab |
| EM | End of Medium | IT | Indent Tab | SA | Set Attribute | WUS | Word Underscore |

**Formatting-Character Representations**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| NSP | Numeric Space | SP | Space | RSP | Required Space | SHY | Syllable Hyphen |

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-979

| Dec | Hex | EBCDIC(4) 81C | 94C | 037 | 500 | 1047 | AS-CII | ISO -8 | IBM -PC | BookMaster Symbol Names(2) |
|---|---|---|---|---|---|---|---|---|---|---|
| 64 | 40 | SP | SP | SP | SP | SP | @ | @ | @ | atsign |
| 65 | 41 | RSP | RSP | RSP | RSP | RSP | A | A | A | |
| 66 | 42 | | | â | â | â | B | B | B | ac |
| 67 | 43 | | | ä | ä | ä | C | C | C | ae |
| 68 | 44 | | | à | à | à | D | D | D | ag |
| 69 | 45 | | | á | á | á | E | E | E | aa |
| 70 | 46 | | | ã | ã | ã | F | F | F | at |
| 71 | 47 | | | å | å | å | G | G | G | ao |
| 72 | 48 | | | ç | ç | ç | H | H | H | cc |
| 73 | 49 | | | ñ | ñ | ñ | I | I | I | nt |
| 74 | 4A | | ¢ | ¢ | [ | ¢ | J | J | J | cent, lbrk |
| 75 | 4B | . | . | . | . | . | K | K | K | period |
| 76 | 4C | < | < | < | < | < | L | L | L | lt |
| 77 | 4D | ( | ( | ( | ( | ( | M | M | M | lpar |
| 78 | 4E | + | + | + | + | + | N | N | N | plus |
| 79 | 4F | | \| | \| | ! | \| | O | O | O | vbar, xclam |
| 80 | 50 | & | & | & | & | & | P | P | P | amp |
| 81 | 51 | | | é | é | é | Q | Q | Q | ea |
| 82 | 52 | | | ê | ê | ê | R | R | R | ec |
| 83 | 53 | | | ë | ë | ë | S | S | S | ee |
| 84 | 54 | | | è | è | è | T | T | T | eg |
| 85 | 55 | | | í | í | í | U | U | U | ia |
| 86 | 56 | | | î | î | î | V | V | V | ic |
| 87 | 57 | | | ï | ï | ï | W | W | W | ie |
| 88 | 58 | | | ì | ì | ì | X | X | X | ig |
| 89 | 59 | | | ß | ß | ß | Y | Y | Y | ss |
| 90 | 5A | | ! | ! | ] | ! | Z | Z | Z | xclam, rbrk |
| 91 | 5B | | $ | $ | $ | $ | [ | [ | [ | dollar, lbrk |
| 92 | 5C | * | * | * | * | * | \ | \ | \ | asterisk, bslash |
| 93 | 5D | ) | ) | ) | ) | ) | ] | ] | ] | rpar, rbrk |
| 94 | 5E | ; | ; | ; | ; | ; | ^ | ^ | ^ | semi, hat |
| 95 | 5F | | ¬ | ¬ | ^ | ^ | _ | _ | _ | lnot, hat, us |

| Dec | Hex | EBCDIC(4) 81C | 94C | 037 | 500 | 1047 | AS-CII | ISO -8 | IBM -PC | BookMaster Symbol Names(2) |
|---|---|---|---|---|---|---|---|---|---|---|
| 96 | 60 | - | - | - | - | - | ` | ` | ` | hyphen or minus, grave |
| 97 | 61 | / | / | / | / | / | a | a | a | divslash or slash |
| 98 | 62 | | | Â | Â | Â | b | b | b | Ac |
| 99 | 63 | | | Ä | Ä | Ä | c | c | c | Ae |
| 100 | 64 | | | À | À | À | d | d | d | Ag |
| 101 | 65 | | | Á | Á | Á | e | e | e | Aa |
| 102 | 66 | | | Ã | Ã | Ã | f | f | f | At |
| 103 | 67 | | | Å | Å | Å | g | g | g | Ao |
| 104 | 68 | | | Ç | Ç | Ç | h | h | h | Cc |
| 105 | 69 | | | Ñ | Ñ | Ñ | i | i | i | Nt |
| 106 | 6A | | ¦ | ¦ | ¦ | ¦ | j | j | j | splitvbar |
| 107 | 6B | , | , | , | , | , | k | k | k | comma |
| 108 | 6C | % | % | % | % | % | l | l | l | percent |
| 109 | 6D | _ | _ | _ | _ | _ | m | m | m | us |
| 110 | 6E | > | > | > | > | > | n | n | n | gt |
| 111 | 6F | ? | ? | ? | ? | ? | o | o | o | quest |
| 112 | 70 | | | ø | ø | ø | p | p | p | os |
| 113 | 71 | | | É | É | É | q | q | q | Ea |
| 114 | 72 | | | Ê | Ê | Ê | r | r | r | Ec |
| 115 | 73 | | | Ë | Ë | Ë | s | s | s | Ee |
| 116 | 74 | | | È | È | È | t | t | t | Eg |
| 117 | 75 | | | Í | Í | Í | u | u | u | Ia |
| 118 | 76 | | | Î | Î | Î | v | v | v | Ic |
| 119 | 77 | | | Ï | Ï | Ï | w | w | w | Ie |
| 120 | 78 | | | Ì | Ì | Ì | x | x | x | Ig |
| 121 | 79 | | | ` | ` | ` | y | y | y | grave |
| 122 | 7A | : | : | : | : | : | z | z | z | colon |
| 123 | 7B | | # | # | # | # | { | { | { | numsign, lbrc |
| 124 | 7C | | @ | @ | @ | @ | \| | \| | \| | atsign, vbar |
| 125 | 7D | ' | ' | ' | ' | ' | } | } | } | ssq(3), rbrc |
| 126 | 7E | = | = | = | = | = | ~ | ~ | ~ | eq, eqv |
| 127 | 7F | " | " | " | " | " | DEL | ⌂ | ⌂ | sdq, house |

**BookMaster Symbols for Character Set 0697 (See Note (4))**

| Symbol Name | Sym-bol | Description | Symbol Name | Sym-bol | Description | Symbol Name | Sym-bol | Description | Symbol Name | Sym-bol | Description |
|---|---|---|---|---|---|---|---|---|---|---|---|
| aa | á | a acute | Dstroke | Đ | D stroke | lpar | ( | left parenthesis | rpar | ) | right parenthesis |
| Aa | Á | A acute | ea | é | e acute | Lsterling | £ | pound sterling | sdq | " | straight double quote |
| ac | â | a circumflex | Ea | É | E acute | lt | < | less than | section | § | section |
| acute | ´ | accent acute | ec | ê | e circumflex | minus | – | minus operation | semi | ; | semicolon |
| Ac | Â | A circumflex | Ec | Ê | E circumflex | mu | µ | mu | slash | / | slash right |
| ae | ä | a umlaut | ee | ë | e umlaut | mult | × | multiply | smultdot | · | mult. dot small |
| aelig | æ | ae ligature | Ee | Ë | E umlaut | nt | ñ | n tilde | splitvbar | ¦ | split vertical bar |
| Ae | Ä | A umlaut | eg | è | e grave | Nt | Ñ | N tilde | ss | ß | German es-zet |
| AElig | Æ | AE ligature | Eg | È | E grave | numsign | # | number sign | ssq | ' | straight single quote |
| ag | à | a grave | eq | = | equals | oa | ó | o acute | sup1 | ¹ | superscript 1 |
| Ag | À | A grave | eth | ð | eth, Icelandic small | Oa | Ó | O acute | sup2 | ² | superscript 2 |
| amp | & | ampersand | Eth | Ð | Eth, Icelandic capital | oc | ô | o circumflex | sup3 | ³ | superscript 3 |
| ao | å | a overcircle | frac12 | ½ | one half | Oc | Ô | O circumflex | thorn | þ | thorn, Icelandic small |
| Ao | Å | A overcircle | frac14 | ¼ | one quarter | odqf | « | French open dbl. quote | Thorn | Þ | Thorn, Icelandic capital |
| asterisk | * | asterisk | frac34 | ¾ | three quarters | oe | ö | o umlaut | tilde | ˜ | tilde |
| at | ã | a tilde | grave | ` | accent grave | Oe | Ö | O umlaut | ua | ú | u acute |
| atsign | @ | at sign | gt | > | greater than | og | ò | o grave | Ua | Ú | U acute |
| At | Ã | A tilde | hat | ^ | hat | Og | Ò | O grave | uc | û | u circumflex |
| bslash | \ | back slash | hyphen | - | hyphen | os | ø | o slash | Uc | Û | U circumflex |
| cc | ç | c cedilla | ia | í | i acute | Os | Ø | O slash | ue | ü | u umlaut |
| Cc | Ç | C cedilla | Ia | Í | I acute | ot | õ | o tilde | Ue | Ü | U umlaut |
| cdqf | » | French close dbl. quote | ic | î | i circumflex | Ot | Õ | O tilde | ug | ù | u grave |
| cedilla | ¸ | cedilla | Ic | Î | I circumflex | overline | ¯ | overline | Ug | Ù | U grave |
| cent | ¢ | cent | ie | ï | i umlaut | par | ¶ | paragraph | umlaut | ¨ | umlaut |
| colon | : | colon | Ie | Ï | I umlaut | percent | % | percent | us | _ | underscore |
| comma | , | comma | ig | ì | i grave | period | . | period | vbar | \| | vertical bar |
| copyr | © | copyright | Ig | Ì | I grave | plus | + | plus | xclam | ! | exclamation point |
| currency | ¤ | currency international | inve | ¡ | inverted ! | pm | ± | plus-minus | ya | ý | y acute |
| degree | ° | degree | invq | ¿ | inverted ? | quest | ? | question mark | Ya | Ý | Y acute |
| div | ÷ | divide | lbrc | { | left brace | rbrc | } | right brace | ye | ÿ | y umlaut |
| divslash | / | division slash | lbrk | [ | left bracket | rbrk | ] | right bracket | yen | ¥ | yen |
| dollar | $ | dollar | lnot | ¬ | logical not | regtm | ® | registered trademark | | | |



LzLabs GmbH. Ex. 1008-980

| Dec | Hex | EBCDIC(4) 81C | 94C | 037 | 500 | 1047 | ISO -8 | IBM-PC 437 | 850 | BookMaster Symbol Names(2) |
|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 80 | | | Ø | Ø | Ø | | Ç | Ç | Os, Cc |
| 129 | 81 | a | a | a | a | a | | ü | ü | ue |
| 130 | 82 | b | b | b | b | b | BPH | é | é | ea |
| 131 | 83 | c | c | c | c | c | NBH | â | â | ac |
| 132 | 84 | d | d | d | d | d | IND | ä | ä | ae |
| 133 | 85 | e | e | e | e | e | NEL | à | à | ag |
| 134 | 86 | f | f | f | f | f | SSA | å | å | ao |
| 135 | 87 | g | g | g | g | g | ESA | ç | ç | cc |
| 136 | 88 | h | h | h | h | h | HTS | ê | ê | ec |
| 137 | 89 | i | i | i | i | i | HTJ | ë | ë | ee |
| 138 | 8A | | | « | « | « | VTS | è | è | odqf, eg |
| 139 | 8B | | | » | » | » | PLD | ï | ï | cdqf, ie |
| 140 | 8C | | | ð | ð | ð | PLU | î | î | eth, ic |
| 141 | 8D | | | ý | ý | ý | RI | ì | ì | ya, ig |
| 142 | 8E | | | þ | þ | þ | SS2 | Ä | Ä | thorn, Ae |
| 143 | 8F | | | ± | ± | ± | SS3 | Å | Å | pm, Ao |
| 144 | 90 | | | ° | ° | ° | DCS | É | É | degree, Ea |
| 145 | 91 | j | j | j | j | j | PU1 | æ | æ | aelig |
| 146 | 92 | k | k | k | k | k | PU2 | Æ | Æ | AElig |
| 147 | 93 | l | l | l | l | l | STS | ô | ô | oc |
| 148 | 94 | m | m | m | m | m | CCH | ö | ö | oe |
| 149 | 95 | n | n | n | n | n | MW | ò | ò | og |
| 150 | 96 | o | o | o | o | o | SPA | û | û | uc |
| 151 | 97 | p | p | p | p | p | EPA | ù | ù | ug |
| 152 | 98 | q | q | q | q | q | SOS | ÿ | ÿ | ye |
| 153 | 99 | r | r | r | r | r | | Ö | Ö | Oe |
| 154 | 9A | | | a | a | a | SCI | Ü | Ü | aus, Ue |
| 155 | 9B | | | º | º | º | CSI | ¢ | ø | ous, cent, os |
| 156 | 9C | | | æ | æ | æ | ST | £ | £ | aelig, Lsterling |
| 157 | 9D | | | | | | OSC | ¥ | Ø | cedilla, yen, Os |
| 158 | 9E | | | Æ | Æ | Æ | PM | ₧ | × | AElig, peseta, mult |
| 159 | 9F | | | ¤ | ¤ | ¤ | ACP | *f* | *f* | currency, fnof(5) |

| Dec | Hex | EBCDIC(4) 81C | 94C | 037 | 500 | 1047 | ISO -8 | IBM-PC 437 | 850 | BookMaster Symbol Names(2) |
|---|---|---|---|---|---|---|---|---|---|---|
| 160 | A0 | | | *μ* | *μ* | *μ* | RSP | á | á | mu(6), aa |
| 161 | A1 | | | ~ | ~ | ~ | ¡ | í | í | tilde, inve, ia |
| 162 | A2 | s | s | s | s | s | ¢ | ó | ó | cent, oa |
| 163 | A3 | t | t | t | t | t | £ | ú | ú | Lsterling, ua |
| 164 | A4 | u | u | u | u | u | ¤ | ñ | ñ | currency, nt |
| 165 | A5 | v | v | v | v | v | ¥ | Ñ | Ñ | yen, Nt |
| 166 | A6 | w | w | w | w | w | ¦ | ª | ª | splitvbar, aus |
| 167 | A7 | x | x | x | x | x | § | º | º | section, ous |
| 168 | A8 | y | y | y | y | y | ¨ | ¿ | ¿ | umlaut, invq |
| 169 | A9 | z | z | z | z | z | © | ⌐ | ® | copyr, lnotrev, regtm |
| 170 | AA | | | ¡ | ¡ | ¡ | ª | ¬ | ¬ | inve, aus, lnot |
| 171 | AB | | | ¿ | ¿ | ¿ | « | ½ | ½ | invq, odqf, frac12 |
| 172 | AC | | | Ð | Ð | Ð | ¬ | ¼ | ¼ | Dstroke or Eth, lnot, frac14 |
| 173 | AD | | | Ý | Ý | [ | SHY | ¡ | ¡ | Ya, lbrk, inve |
| 174 | AE | | | þ | þ | Þ | ® | « | « | Thorn, regtm, odqf |
| 175 | AF | | | ® | ® | ® | ¯ | » | » | regtm, overline, cdqf |
| 176 | B0 | | | ^ | ¢ | ¬ | ° | ⋮ | ⋮ | hat, cent, lnot, degree, box14 |
| 177 | B1 | | | £ | £ | £ | ± | ▒ | ▒ | Lsterling, pm, box12 |
| 178 | B2 | | | ¥ | ¥ | ¥ | ² | ▓ | ▓ | yen, sup2, box34 |
| 179 | B3 | | | · | · | · | ³ | │ | │ | smultdot, sup3, bxv |
| 180 | B4 | | | © | © | © | ´ | ┤ | ┤ | copyr, acute, bxrj |
| 181 | B5 | | | § | § | § | *μ* | ╡ | Á | section, mu(6), bx1012, Aa |
| 182 | B6 | | | ¶ | ¶ | ¶ | ¶ | ╢ | Â | par, bx2021, Ac |
| 183 | B7 | | | ¼ | ¼ | ¼ | · | ╖ | À | frac14, smultdot, bx0021, Ag |
| 184 | B8 | | | ½ | ½ | ½ | ¸ | ╕ | © | frac12, cedilla, bx0012, copyr |
| 185 | B9 | | | ¾ | ¾ | ¾ | ¹ | ╣ | ╣ | frac34, sup1, bx2022 |
| 186 | BA | | | [ | ¬ | Ý | º | ║ | ║ | lbrk, lnot, Ya, ous, bx2020 |
| 187 | BB | | | ] | \| | ¨ | » | ╗ | ╗ | rbrk, vbar, umlaut, cdqf, bx0022 |
| 188 | BC | | | | ¯ | ¯ | ¼ | ╝ | ╝ | overline, frac14, bx2002 |
| 189 | BD | | | ¨ | ¨ | ] | ½ | ╜ | ¢ | umlaut, rbrk, frac12, bx2001, cent |
| 190 | BE | | | ´ | ´ | ´ | ¾ | ╛ | ¥ | acute, frac34, bx1002, yen |
| 191 | BF | | | × | × | × | ¿ | ┐ | ┐ | mult, invq, bxur |

**Additional ISO-8 Control-Character Representations**

APC Application Program Command
BPH Break Permitted Here
CCH Cancel Character
CSI Control Sequence Introducer
DCS Device Control String
EPA End of Guarded Area
ESA End of Selected Area
HTJ Character Tabulation with Justification
HTS Character Tabulation Set
IFS Information Separator Four
IGS Information Separator Three
IND Index
IRS Information Separator Two
MW Message Waiting
NBH No Break Here
NEL Next Line
OSC Operating System Command
PLD Partial Line Down
PLU Partial Line Up
PM Privacy Message
PU1 Private Use One
PU2 Private Use Two
RI Reverse Line Feed (or Index)
SCI Single Character Introducer
SOS Start of String
SPA Start of Guarded Area
SSA Start of Selected Area
SS2 Single Shift Two
SS3 Single Shift Three
ST String Terminator
STS Set Transmit State
US Information Separator One
VTS Line Tabulation Set


LzLabs GmbH. Ex. 1008-981

| Dec | Hex | EBCDIC(4) 81C | 94C | 037 | 500 | 1047 | ISO -8 | IBM-PC 437 | 850 | BookMaster Symbol Names(2) |
|---|---|---|---|---|---|---|---|---|---|---|
| 192 | C0 | | | { | { | { | À | └ | └ | lbrc, Ag, bxll |
| 193 | C1 | A | A | A | A | A | Á | ┴ | ┴ | Aa, bxbj |
| 194 | C2 | B | B | B | B | B | Â | ┬ | ┬ | Ac, bxtj |
| 195 | C3 | C | C | C | C | C | Ã | ├ | ├ | At, bxlj |
| 196 | C4 | D | D | D | D | D | Ä | ─ | ─ | Ae, bxh |
| 197 | C5 | E | E | E | E | E | Å | ┼ | ┼ | Ao, bxcj |
| 198 | C6 | F | F | F | F | F | Æ | ╞ | ã | AElig, bx1210, at |
| 199 | C7 | G | G | G | G | G | Ç | ╟ | Ã | Cc, bx2120, At |
| 200 | C8 | H | H | H | H | H | È | ╚ | ╚ | Eg, bx2200 |
| 201 | C9 | I | I | I | I | I | É | ╔ | ╔ | Ea, bx0220 |
| 202 | CA | SHY | SHY | SHY | SHY | SHY | Ê | ╩ | ╩ | Ec, bx2202 |
| 203 | CB | | | ô | ô | ô | Ë | ╦ | ╦ | oc, Ee, bx0222 |
| 204 | CC | | | ö | ö | ö | Ì | ╠ | ╠ | oe, Ig, bx2220 |
| 205 | CD | | | ò | ò | ò | Í | ═ | ═ | og, Ia, bx0202 |
| 206 | CE | | | ó | ó | ó | Î | ╬ | ╬ | oa, Ic, bx2222 |
| 207 | CF | | | õ | õ | õ | Ï | ¤ | ¤ | ot, Ie, bx1202, currency |
| 208 | D0 | | | } | } | } | Ð | ╨ | ð | rbrc, Dstroke or Eth, bx2101, eth |
| 209 | D1 | J | J | J | J | J | Ñ | ╤ | Ð | Nt, bx0212, Dstroke or Eth |
| 210 | D2 | K | K | K | K | K | Ò | ╥ | Ê | Og, bx0121, Ec |
| 211 | D3 | L | L | L | L | L | Ó | ╙ | Ë | Oa, bx2100, Ee |
| 212 | D4 | M | M | M | M | M | Ô | ╘ | È | Oc, bx1200, Eg |
| 213 | D5 | N | N | N | N | N | Õ | ╒ | ı | Ot, bx0210, idotless |
| 214 | D6 | O | O | O | O | O | Ö | ╓ | Í | Oe, bx0120, Ia |
| 215 | D7 | P | P | P | P | P | × | ╫ | Î | mult, bx2121, Ic |
| 216 | D8 | Q | Q | Q | Q | Q | Ø | ╪ | Ï | Os, bx1212, Ie |
| 217 | D9 | R | R | R | R | R | Ù | ┘ | ┘ | Ug, bxlr |
| 218 | DA | | | ¹ | ¹ | ¹ | Ú | ┌ | ┌ | sup1, Ua, bxul |
| 219 | DB | | | û | û | û | Û | ▉ | ▉ | uc, Uc, BOX |
| 220 | DC | | | ü | ü | ü | Ü | ▄ | ▄ | ue, Ue, BOXBOT |
| 221 | DD | | | ù | ù | ù | Ý | ▌ | ¦ | ug, Ya, BOXLEFT, splitvbar |
| 222 | DE | | | ú | ú | ú | þ | ▐ | Ì | ua, thorn, BOXRIGHT, Ig |
| 223 | DF | | | ÿ | ÿ | ÿ | ß | ▀ | ▀ | ye, ss, BOXTOP |
| 224 | E0 | | | \ | \ | \ | à | α | Ó | bslash, ag, alpha, Oa |
| 225 | E1 | | NSP | ÷ | ÷ | ÷ | á | ß | ß | div, aa, ss |
| 226 | E2 | S | S | S | S | S | â | Γ | Ô | ac, Gamma, Oc |
| 227 | E3 | T | T | T | T | T | ã | π | Ò | at, pi, Og |
| 228 | E4 | U | U | U | U | U | ä | Σ | õ | ae, Sigma, ot |
| 229 | E5 | V | V | V | V | V | å | σ | Õ | ao, sigma, Ot |
| 230 | E6 | W | W | W | W | W | æ | μ | μ | aelig, mu(6) |
| 231 | E7 | X | X | X | X | X | ç | τ | þ | cc, tau, thorn |
| 232 | E8 | Y | Y | Y | Y | Y | è | Φ | Þ | eg, Phi, Thorn |
| 233 | E9 | Z | Z | Z | Z | Z | é | Θ | Ú | ea, Theta(5), Ua |
| 234 | EA | | | ² | ² | ² | ê | Ω | Û | sup2, ec, Omega, Uc |
| 235 | EB | | | Ô | Ô | Ô | ë | δ | Ù | Oc, ee, delta, Ug |
| 236 | EC | | | Ö | Ö | Ö | ì | ∞ | ý | Oe, ig, infinity, ya |
| 237 | ED | | | Ò | Ò | Ò | í | φ | Ý | Og, ia, phi, Ya |
| 238 | EE | | | Ó | Ó | Ó | î | ε | ¯ | Oa, ic, epsilon, overline |
| 239 | EF | | | Õ | Õ | Õ | ï | ∩ | ´ | Ot, ie, intersect, acute |
| 240 | F0 | 0 | 0 | 0 | 0 | 0 | ð | ≡ | SHY | eth, identical |
| 241 | F1 | 1 | 1 | 1 | 1 | 1 | ñ | ± | ± | nt, pm |
| 242 | F2 | 2 | 2 | 2 | 2 | 2 | ò | ≥ | = | og, ge, eq |
| 243 | F3 | 3 | 3 | 3 | 3 | 3 | ó | ≤ | ¾ | oa, le, frac34 |
| 244 | F4 | 4 | 4 | 4 | 4 | 4 | ô | ⌠ | ¶ | oc, inttop, par |
| 245 | F5 | 5 | 5 | 5 | 5 | 5 | õ | ⌡ | § | ot, intbot, section |
| 246 | F6 | 6 | 6 | 6 | 6 | 6 | ö | ÷ | ÷ | oe, div |
| 247 | F7 | 7 | 7 | 7 | 7 | 7 | ÷ | ≈ | ¸ | div, nearly(5), cedilla |
| 248 | F8 | 8 | 8 | 8 | 8 | 8 | ø | ° | ° | os, degree |
| 249 | F9 | 9 | 9 | 9 | 9 | 9 | ù | • | ¨ | ug, lmultdot, umlaut |
| 250 | FA | | | ³ | ³ | ³ | ú | · | · | sup3, ua, smultdot |
| 251 | FB | | | Û | Û | Û | û | √ | ¹ | Uc, uc, sqrt, sup1 |
| 252 | FC | | | Ü | Ü | Ü | ü | ⁿ | ³ | Ue, ue, supn, sup3 |
| 253 | FD | | | Ù | Ù | Ù | ý | ² | ² | Ug, ya, sup2 |
| 254 | FE | | | Ú | Ú | Ú | þ | ■ | ■ | Ua, thorn, sqbul |
| 255 | FF | EO | EO | EO | EO | EO | ÿ | RSP | RSP | ye |

**Notes**:

**(1)** The ASCII controls and graphics are from ANSI X3.4. The ISO-8 controls are from ISO 6429, and the graphics are from ISO 8859-1. The ISO-8 graphics are code page 00819, named ISO/ANSI Multilingual. IBM-PC controls and graphics are shown. The graphics are common to code page 00437, named Personal Computer, and code page 00850, named Personal Computer - Multilingual Page. Code pages 00437 and 00850 are shown separately beginning at X'80', after which they diverge in content.

**(2)** The symbol names shown are to be preceded by an ampersand (&) and followed by a period (.) to form a symbol. Source: *IBM BookMaster User's Guide Release 4.0*, SC34-5009.

**(3)** ASCII, ISO-8, and IBM-PC X'27' and EBCDIC X'7D' are an apostophe having the appearance of a straight single quote. The BookMaster “apos” produces a character having the appearance of an accent acute.

**(4)** Five columns of EBCDIC graphics are shown. The first is the 81-character character set 0640, called the syntactic character set, that is mapped the same on all EBCDIC code pages. The second is the standard IBM 94-character character set mapped on code page 00037. The third is code page 00037, named USA/Canada - CECP (Country Extended Code Page). The fourth is code page 00500, named International #5. The fifth is code page 01047, named Latin 1/Open Systems. Code pages 00037, 00500, 01047, and 00819 (ISO-8) all map the 189-character character set 0697. Source: *National Language Support Reference Manual Volume 2*, SE09-8002.

**(5)** $f$, ≈, and Θ are of nonstandard width.

**(6)** EBCDIC X'A0' and ISO-8 X'B5' are micro but resemble mu. The BookMaster “usec” produces a character of nonstandard width.



LzLabs GmbH. Ex. 1008-982

# Index

## Numerics

2K-IDAW control 15-25
370-XA architecture 1-10
  comparison of facilities with System/370 F-1
  comparison with ESA/370 E-1

## A

A (ADD) binary instruction 7-12
absolute address 3-4
absolute storage 3-4
access-control bits in storage key 3-8
access exceptions 6-35, 6-40
  priority of 6-40
  recognition of 6-35
access key 3-9
  for channel-program execution 3-9, 15-22
  for channel-subsystem monitoring 3-9
  for CPU 3-9
access list 5-46
  *See also* access-list entry
  accessing capability, revocation of 5-40
  allocation and invalidation of entries in 5-37
  authorizing the use of entries in 5-38
  concepts 5-35
  designation (ALD) 5-45
  length (ALL) 5-45
  origin (ALO) 5-45
access-list-controlled protection 3-11, D-1
  exception for 6-28
access-list entry (ALE) 5-46
  authorization index (ALEAX) 5-46
  number
    *See* ALEN
  sequence exception 6-19
    as an access exception 6-35
  sequence number (ALESN)
    in ALE 5-46
    in ALET 5-43
  token
    *See* ALET
access-register mode 3-28
access-register translation (ART) 5-42
  as part of LOAD REAL ADDRESS, TEST ACCESS, and TEST PROTECTION 5-48
  introduction to 5-35
  lookaside buffer
    *See* ALB
  sequence of table fetches 5-83
access-register-translation (ART) tables 5-44
access registers 2-4, D-1
  designation of 5-35
  functions of 5-34
  instructions for use of 5-41
  save areas for 3-48
  validity bit for 11-23
access to storage 5-78
  *See also* reference
  by use of MOVE PAGE 7-93
active
  device 16-15
  subchannel 16-15
active allegiance 15-12
active communication 15-12
activity-control field (SCSW) 16-13
  following TEST SUBCHANNEL 14-20
AD (ADD NORMALIZED) HFP instruction 18-8
  example A-39
ADB (ADD) BFP instruction 19-18
ADBR (ADD) BFP instruction 19-18
ADD BFP instructions 19-18
ADD binary instructions 7-12
ADD DECIMAL instruction 8-6
  example A-34
ADD HALFWORD IMMEDIATE instruction 7-12
ADD HALFWORD instruction 7-12
  example A-7
ADD LOGICAL instructions 7-13
ADD LOGICAL WITH CARRY instructions 7-13
ADD NORMALIZED HFP instructions 18-8
  example A-39
ADD UNNORMALIZED HFP instructions 18-10
  example A-39
additional floating-point (AFP) registers 9-2
address 3-2
  24-bit and 31-bit 3-5, F-1
    in branch-address generation 5-9
    in operand address generation 5-8
  31-bit real and absolute F-1
  absolute 3-4
  arithmetic 3-5, 5-8
    unsigned binary 7-4
  backward stack-entry 5-70
  base
    *See* base address
  branch
    *See* branch address
  channel-program
    *See* channel-program address
  comparison 12-1
    controls for 12-1
    effect on CPU state 4-2

© Copyright IBM Corp. 1990-2003



LzLabs GmbH. Ex. 1008-983

address *(continued)*
CPU
*See* CPU address
data (I/O)
*See* data address
effective
*See* effective address
extended save area 3-47
failing-storage
*See* failing-storage address
format 3-2
forward-section-header 5-70
generation 5-7
for storage addressing 3-5
I/O 13-5
instruction
*See* instruction address
invalid 6-16
logical
*See* logical address
numbering of for byte locations 3-2
PER
*See* PER address
prefixing
*See* prefix
primary virtual
*See* primary virtual address
real 3-4
secondary virtual
*See* secondary virtual address
size of 3-5
controlled by addressing mode 5-7
storage 3-2
summary information 3-40
translation
*See* dynamic address translation, prefix
types 3-3
virtual 3-4
wraparound
*See* wraparound
address-limit checking (I/O) 17-20
effect of I/O-system reset on 17-16
limit mode (bits in PMCW) 15-2
address-limit-checking control (I/O) 15-25, 16-11
used for IPL 17-18
address space 3-16
AR-specified 5-34
changing of 3-17
control bits
control bit 5-65
in PSW 4-6
use in address translation 3-28
created by DAT 3-27
number
*See* ASN
address-space-function (ASF) control bit 5-42
use in ASN translation 3-19
use in PC-number translation 5-27
address-and-translation-mode identification (ATMID) 4-17
addressing exception 6-16
as an access exception 6-35, 6-40
addressing mode 5-7
bit in entry-table entry 5-29
bit in linkage-stack state entry 5-72
bit in PSW 4-6
effect on address size 3-5
effect on operand-address generation 5-8
effect on sequential instruction-address generation 5-7
effect on wraparound 3-5
in branch-address generation 5-9
in examples A-7
in operand address generation 5-8
set by BRANCH AND SAVE AND SET MODE instruction 7-16
set by BRANCH AND SET MODE instruction 7-16
use of 5-14
ADR (ADD NORMALIZED) HFP instruction 18-8
AE (ADD NORMALIZED) HFP instruction 18-8
example A-39
AEB (ADD) BFP instruction 19-18
AEBR (ADD) BFP instruction 19-18
AER (ADD NORMALIZED) HFP instruction 18-8
AFP (additional floating-point) registers 9-2
AFP-register data exception 6-21
AFT (ASN first table) 3-19
AFTE (ASN-first-table entry) 3-19
AFTO (ASN-first-table origin) 3-19
AFX (ASN-first-table index) 3-18
invalid bit 3-19
translation exception 6-19
AH (ADD HALFWORD) instruction 7-12
example A-7
AHI (ADD HALFWORD IMMEDIATE) instruction 7-12
AKM (authorization key mask) 5-29
AL (ADD LOGICAL) instruction 7-13
ALB (ART-lookaside buffer) 5-53
entry
clearing of 5-55
effect of translation changes on 5-55
usable state 5-54
ALC (ADD LOGICAL WITH CARRY) instruction 7-13
ALCR (ADD LOGICAL WITH CARRY) instruction 7-13
ALD (access-list designation) 5-45
ALE
*See* access-list entry
ALEAX (access-list-entry authorization index) 5-46
ALEN (access-list-entry number) 5-44
invalid bit 5-46
translation exception 6-19
as an access exception 6-35



LzLabs GmbH. Ex. 1008-984

alert (class of machine-check condition) 11-12
alert interruption condition (I/O) 16-4
alert-status bit (I/O) 16-16
ALESN (access-list-entry sequence number)
in ALE 5-46
in ALET 5-43
ALET (access-list-entry token) 5-37, 5-43
specification exception 6-19
as an access exception 6-35
alignment 3-3
ALL (access-list length) 5-45
allegiance
active 15-12
channel-path 15-11
dedicated 15-12
effect on CLEAR SUBCHANNEL of 15-11
working 15-12
allowed interruptions 6-6
ALO (access-list origin) 5-45
ALR (ADD LOGICAL) instruction 7-13
alter-and-display controls 12-2
alteration
general-register (PER event) 4-23
storage (PER event) 4-22
ancillary-report bit
in channel-report word 17-24
in machine-check-interruption code 11-19
in subchannel logout 16-34
AND instructions 7-13
examples A-7
AP (ADD DECIMAL) instruction 8-6
example A-34
AR (ADD) binary instruction 7-12
AR-specified (access-register-specified) address space 3-16, 5-34
AR-specified (access-register-specified) virtual address 3-4
effective segment-table designation for 3-32
architectural mode
identification 3-47
indication of 12-2
selection of by IML controls 12-3
selection of by manual controls 12-2
selection of by signal-processor order 4-49
architecture
compatibility 1-12
arithmetic
address
*See* address arithmetic
binary 7-3
examples A-2
decimal 8-2
examples A-4, A-34
floating-point 9-1
examples A-5, A-39
logical (unsigned binary) 7-4
examples A-3
ART
*See* access-register translation
art-lookaside buffer
*See* ALB
ASCII character code
handled by architecture xxi
ASF-control bit
*See* address-space-function-control bit
ASN (address-space number) 3-17
authorization 3-23
first table (AFT) 3-19
first-table (AFT) origin (AFTO) 3-19
first-table index
*See* AFX
first table origin (AFTO) 3-19
in entry-table entry 5-29
second table (AST) 3-19
second-table (AST) origin (ASTO) 3-19
second-table address in ETE 5-29
second-table entry (ASTE)
address 5-66
basic (16-byte) 3-19
extended (64-byte) 5-47
for subspace groups 5-57
origin, in ALE 5-47
primary (PASTE) 5-28
pseudo 3-17
sequence exception 6-20
sequence exception as an access exception 6-35
sequence number (ASTESN), in ALE 5-47
sequence number (ASTESN), in ASTE 5-48
validity exception 6-20
validity exception as an access exception 6-35
second-table index
*See* ASX
trace-control bit 4-11
translation 3-18
exceptions 6-45
specification exception 6-19
specification exception as an access exception 6-35
translation-control bit 3-18, 5-22
assembler language A-7
instruction formats in
*See* instruction lists and page numbers in Appendix B
assigned storage locations 3-43
comparison of 370-XA with System/370 F-6
comparison of ESA/370 with 370-XA E-3
AST (ASN second table) 3-19
AST entry
*See* ASN-second-table entry
ASTE
*See* ASN-second-table entry



LzLabs GmbH. Ex. 1008-985

ASTESN (AST-entry sequence number)
in ALE 5-47
in ASTE 5-48
ASTO (ASN-second-table origin) 3-19
ASX (ASN-second-table index) 3-18
invalid bit 3-20
use in ART 5-47
translation exception 6-21
asynchronous-data-mover facility 1-10
asynchronous-pageout facility 1-10
AT
*See* authority table
ATL (authority-table length) 3-20
use in ART 5-48
ATMID (addressing-and-translation-mode identification) 4-17
ATO (authority-table origin) 3-20
use in ART 5-48
attached ART-table entry 5-54
attached segment-table or page-table entry 3-36
attachment of I/O devices 13-2
AU (ADD UNNORMALIZED) HFP instruction 18-10
example A-39
AUR (ADD UNNORMALIZED) HFP instruction 18-10
authority table (AT) 5-23
designation 3-20, 5-48
length 3-20, 5-48
origin 3-20, 5-48
authorization
ASN 3-23
index (AX) 3-24, 5-23
key mask (AKM) 5-29
mechanisms 5-21
summary of 5-25
testing of 5-63
authorization check 16-36
automatic reconfiguration 1-9
auxiliary storage 3-1, 3-26
availability (characteristic of a system) 1-14
AW (ADD UNNORMALIZED) HFP instruction 18-10
AWR (ADD UNNORMALIZED) HFP instruction 18-10
AX (authorization index) 3-24, 5-23
AXBR (ADD) BFP instruction 19-18
AXR (ADD NORMALIZED) HFP instruction 18-8

# B

B field of instruction 5-8
backed-up bit (machine-check condition) 11-19
backup
processing (synchronous machine-check condition) 11-20
backward stack-entry address 5-70
backward stack-entry validity bit 5-70
BAKR (BRANCH AND STACK) instruction 10-10
examples A-9
BAL (BRANCH AND LINK) instruction 7-14
examples A-8
BALR (BRANCH AND LINK) instruction 7-14
examples A-8
BAS (BRANCH AND SAVE) instruction 7-15
example A-8
base address 5-8
register for 2-3
base-AST-entry origin (BASTEO) 5-56
base space 5-11
base-space bit 5-58
base-authority state 10-7
basic AST entry 3-19
basic entry-table entry 5-28
basic I/O functions 15-1
basic operator facilities 12-1
basic PROGRAM CALL 5-61, 10-54
BASR (BRANCH AND SAVE) instruction 7-15
example A-8
BASSM (BRANCH AND SAVE AND SET MODE) instruction 7-16
example A-8
BASTEO (base-AST-entry origin) 5-56
BC (BRANCH ON CONDITION) instruction 7-17
example A-11
BCR (BRANCH ON CONDITION) instruction 7-17
BCT (BRANCH ON COUNT) instruction 7-18
example A-12
BCTR (BRANCH ON COUNT) instruction 7-18
example A-12
BFP
data class
testing of 19-46
BFP (binary floating point) 9-1
BFP data 19-4
conversion of 9-8
BFP facility 19-1
BFP-instruction data exception 6-21
bias for exponent 19-4
big endian 7-81
bimodal addressing 5-7, F-1
*See also* addressing mode
binary
*See also* fixed point
arithmetic 7-3
examples A-2
negative zero 7-3
number representation 7-2
examples A-2
overflow 7-3
example A-2
sign bit 7-3
binary floating point (BFP) 9-1
binary integer
conversion from floating point 18-12, 19-26
conversion to floating point 18-11, 19-26



LzLabs GmbH. Ex. 1008-986

binary-to-decimal conversion 7-67
example A-18
bit 3-2
numbering of within a group of bytes 3-2
block-concurrent storage references 5-88
block number
expanded storage 2-2
block of I/O data 15-22
block of storage 3-4
*See also* page
testing for usability of 10-107
borrow 7-128
boundary alignment 3-3
for instructions 5-3
branch address 5-9
control bit 4-15
in linkage-stack state entry 5-72
in trace entry 4-13
BRANCH AND LINK instructions 7-14
examples A-8
BRANCH AND SAVE AND SET MODE instruction 7-16
examples A-8
BRANCH AND SAVE instructions 7-15
examples A-8
BRANCH AND SET AUTHORITY instruction 10-7
BRANCH AND SET MODE instruction 7-16
examples A-8
BRANCH AND STACK instruction 10-10
examples A-9
BRANCH IN SUBSPACE GROUP instruction 10-13
BRANCH ON CONDITION instructions 7-17
example A-11
BRANCH ON COUNT instructions 7-18
example A-12
BRANCH ON INDEX HIGH instruction 7-18
examples A-12
BRANCH ON INDEX LOW OR EQUAL instruction 7-18
examples A-13
BRANCH RELATIVE AND SAVE instruction 7-19
BRANCH RELATIVE AND SAVE LONG instruction 7-19
BRANCH RELATIVE ON CONDITION instruction 7-20
BRANCH RELATIVE ON CONDITION LONG instruction 7-20
BRANCH RELATIVE ON COUNT instruction 7-21
BRANCH RELATIVE ON INDEX HIGH instruction 7-21
BRANCH RELATIVE ON INDEX LOW OR EQUAL instruction 7-21
branch state entry 5-71, 10-10
branch-trace-control bit 4-11
branching
branch-address generation 5-9
in a channel program 15-41
branching *(continued)*
relative 5-9
to perform decision making, loop control, and subroutine linkage 5-10
using the linkage stack 5-62
BRAS (BRANCH RELATIVE AND SAVE) instruction 7-19
BRASL (BRANCH RELATIVE AND SAVE LONG) instruction 7-19
BRC (BRANCH RELATIVE ON CONDITION) instruction 7-20
BRCL (BRANCH RELATIVE ON CONDITION LONG) instruction 7-20
BRCT (BRANCH RELATIVE ON COUNT) instruction 7-21
broadcasted-purging facility 1-1
BRXH (BRANCH RELATIVE ON INDEX HIGH) instruction 7-21
BRXLE (BRANCH RELATIVE ON INDEX LOW OR EQUAL) instruction 7-21
BSA (BRANCH AND SET AUTHORITY) instruction 10-7
BSG (BRANCH IN SUBSPACE GROUP) instruction 10-13
BSM (BRANCH AND SET MODE) instruction 7-16
example A-8
buffer storage (cache) 3-2
burst mode (channel-path operation) 13-3
busy
in I/O operations 13-7
in SIGNAL PROCESSOR 4-50
BXH (BRANCH ON INDEX HIGH) instruction 7-18
examples A-12
BXLE (BRANCH ON INDEX LOW OR EQUAL) instruction 7-18
examples A-13
bypassing POST and WAIT A-45
byte 3-2
numbering of in storage 3-2
byte index (BX) 3-27
byte-multiplex mode (channel-path operation) 13-3

# C

C (COMPARE) binary instruction 7-35
cache 3-2
called-space identification 5-72
cancel-I/O facility 1-9
CANCEL SUBCHANNEL instruction 14-4
capability list 5-39
carry 7-3
CBC (checking-block code) 11-2
invalid 11-2
in registers 11-10
in storage 11-6
in storage keys 11-7



LzLabs GmbH. Ex. 1008-987

CBC (checking-block code) *(continued)*
near-valid 11-2
valid 11-2
CCC (channel-control check) 16-27
CCW (channel-command word) 15-27
address of 16-18
byte count in 15-29
chaining 15-31
check (in subchannel logout) 16-33
command codes
*See* commands
contents of 15-28
current 15-27
designation of storage area in 15-28, 15-30
format-0 and format-1 15-28
format control 16-10
used for IPL 17-18
IDA flag in 15-29
in IPL
assigned storage locations for 3-43
indirect data addressing used in 13-7, 15-36
invalid format of 16-25
invalid specification of 16-24
PCI flag in 15-29
prefetch control in 16-10
used for IPL 17-18
prefetching 15-33
retry of
*See* command retry
role in I/O operations of 13-6
skip flag in 15-29
suspend flag in 15-29
CD (COMPARE) HFP instruction 18-10
CDB (COMPARE) BFP instruction 19-23
CDBR (COMPARE) BFP instruction 19-23
CDFBR (CONVERT FROM FIXED) BFP instruction 19-26
CDFR (CONVERT FROM FIXED) HFP instruction 18-11
CDR (COMPARE) HFP instruction 18-10
examples A-40
CDS (COMPARE DOUBLE AND SWAP) instruction 7-40
examples A-44
CE (COMPARE) HFP instruction 18-10
CEB (COMPARE) BFP instruction 19-23
CEBR (COMPARE) BFP instruction 19-23
CEFBR (CONVERT FROM FIXED) BFP instruction 19-26
CEFR (CONVERT FROM FIXED) HFP instruction 18-11
central processing unit
*See* CPU
CER (COMPARE) HFP instruction 18-10
CFC (COMPARE AND FORM CODEWORD) instruction 7-36
CFC (COMPARE AND FORM CODEWORD) instruction *(continued)*
example A-51
CFDBR (CONVERT TO FIXED) BFP instruction 19-26
CFDR (CONVERT TO FIXED) HFP instruction 18-12
CFEBR (CONVERT TO FIXED) BFP instruction 19-26
CFER (CONVERT TO FIXED) HFP instruction 18-12
CFXBR (CONVERT TO FIXED) BFP instruction 19-26
CFXR (CONVERT TO FIXED) HFP instruction 18-12
CH (COMPARE HALFWORD) instruction 7-42
example A-14
chaining check (subchannel status) 16-29
chaining of CCWs 15-31
command
*See* command chaining of CCWs
data
*See* data chaining of CCWs
chaining of CRWs 17-23, 17-24
change bit in storage key 3-8
change recording 3-14
channel-command word
*See* CCW
channel commands
*See* commands (I/O)
channel-control check (subchannel status) 16-27
channel-data check (subchannel status) 16-26
channel path 13-2
active allegiance for 15-12
available for selection 15-13
dedicated allegiance for 15-12
effect of I/O-system reset on 17-15
masks in SCHIB
*See* LPM, LPUM, PAM, PIM, PNOM, POM
multipath mode of 15-3, 15-21
not operational 16-12
parallel-I/O-interface type 13-3
serial-I/O-interface type 13-2
storing of status for 14-16
type of 13-2, 13-5
working allegiance for 15-12
channel-path identifier
*See* CHPID
channel-path reset 17-13
effect of I/O-system reset on 17-15
channel-path-reset function 15-45
completion of 15-45
initiation by RESET CHANNEL PATH 14-9
reset signal issued as part of 17-13
signaling for 15-45
channel-path-status word 14-16
channel-path timeout
indicator for (in ERW) 16-37
channel program 15-27
2K-IDAW control for 15-25
branching in
*See* TIC



LzLabs GmbH. Ex. 1008-988

channel program *(continued)*
execution of 13-6, 15-21
resumption of 14-11
suspension of 13-8, 15-38
format-2-IDAW control for 15-25
modification control for 15-23
serialization 5-92
streaming-mode control for 15-23
suspend control for 15-23
synchronization control for 15-24
channel-program address 16-18
field-validity flag for in IRB 16-34
used for IPL 17-18
channel report 17-22
generated as a result of RCHP 14-9
channel report pending 11-18, 17-22
effect of I/O-system reset on 17-16
subclass-mask bit for 11-26
channel-report word
*See* CRW
channel subsystem 2-6, 13-1
addressing used in 13-5
damage 11-19
effect of I/O-system reset on 17-14
effect of power-on reset on 4-43
isolated state of 16-36
channel-subsystem-call facility 1-9
channel-subsystem monitoring 17-1
effect of I/O-system reset on 17-16
channel-subsystem recovery 11-4, 17-21
channel-subsystem timer 17-2
effect of I/O-system reset on 17-16
channel-subsystem timing 17-2
channel-subsystem timing-facility bit (in PMCW) 15-4
characteristic (of HFP number) 18-1
*See also* exponent
characters
represented by eight-bit code xxi
check bits 3-2, 11-2
check stop 4-3, 11-11
as signal-processor status 4-53
during manual operation 12-1
effect on CPU timer 4-36
entering of 11-14
indicator 12-2
malfunction alert for 6-12
system 11-11
checking block 11-2
checking-block code
*See* CBC
checkpoint 11-2
checkpoint synchronization 11-3
action 11-4
operations 11-3
CHECKSUM instruction 7-22
CHI (COMPARE HALFWORD IMMEDIATE) instruction 7-42
CHPID (channel-path identifier) 13-5
in PMCW 15-7
used in RESET CHANNEL PATH 14-9
CIPHER MESSAGE instruction 7-26
CIPHER MESSAGE WITH CHAINING instruction 7-26
CKSM (CHECKSUM) instruction 7-22
CL (COMPARE LOGICAL) instruction 7-42
class
of BFP data 19-5
testing of 19-46
CLC (COMPARE LOGICAL) instruction 7-42
example A-14
CLCL (COMPARE LOGICAL LONG) instruction 7-43
example A-16
CLCLE (COMPARE LOGICAL LONG EXTENDED) instruction 7-45
CLCLU (COMPARE LOGICAL LONG UNICODE) instruction 7-47
clear function 15-14
bit in SCSW for 16-13
completion of 15-15
initiated by CLEAR SUBCHANNEL 14-5
path management for 15-14
pending 16-15
signaling for 15-15
subchannel modification by 15-14
clear reset 4-42
clear signal 17-12
issued as part of clear function 15-15
CLEAR SUBCHANNEL instruction 14-5
*See also* clear function
effect on device status of 15-15
function initiated by 15-14
use of after RESET CHANNEL PATH 14-9
clearing operation
by clear-reset function 4-42
by load-clear key 12-3
by system-reset-clear key 12-5
by TEST BLOCK instruction 10-107
CLI (COMPARE LOGICAL) instruction 7-42
example A-15
CLM (COMPARE LOGICAL CHARACTERS UNDER MASK) instruction 7-43
example A-15
clock
*See* TOD clock
clock comparator 4-35
external interruption 6-11
save areas for 3-48
validity bit for 11-23
clock unit 4-33, 7-125
CLR (COMPARE LOGICAL) instruction 7-42
example A-15



LzLabs GmbH. Ex. 1008-989

CLST (COMPARE LOGICAL STRING) instruction 7-50
examples A-17
code
ASCII
handled by architecture xxi
checking-block
*See* CBC
command (in CCW)
*See* command code in CCW
condition
*See* condition code
data-exception (DXC) 6-15
decimal digit and sign 8-2
deferred condition (I/O) 16-8
EBCDIC
handled by architecture xxi
table for I-1
eight-bit
handled by architecture xxi
error-recovery (I/O) 17-24
exception-extension 6-15
external-damage 11-24
validity bit for 11-22
I/O-interruption subclass 15-2
instruction-length
*See* ILC
interruption
*See* interruption code
linkage-stack-entry type 5-69
monitor
*See* monitor code
operation 5-2
PER
*See* PER code
reporting-source (I/O) 17-24
storage-access (in subchannel logout) 16-34
version 10-90
codeword (for sorting operations) 7-36
example A-52
command chaining of CCWs 15-34
effect of status modifier on 15-35
flag in CCW for 15-28
overview of 13-8
command code in CCW 15-29
*See also* commands
*See also* common I/O-device commands
applicable flags 15-40
invalid 16-24
command codes
*See* command code in CCW
command retry 15-42
effect on PCI of 15-36
commands (I/O) 15-29
*See also* common I/O-device commands
transfer in channel 15-41
common I/O-device commands
publication referenced xxi
common-segment bit 3-30
COMPARE AND FORM CODEWORD instruction 7-36
example A-51
COMPARE AND SIGNAL BFP instructions 19-24
COMPARE AND SWAP instruction 7-40
COMPARE AND SWAP AND PURGE instruction 10-17
COMPARE AND SWAP instruction
examples A-44
COMPARE BFP instructions 19-23
COMPARE binary instructions 7-35
COMPARE DECIMAL instruction 8-6
example A-34
COMPARE DOUBLE AND SWAP instruction 7-40
examples A-44
COMPARE HALFWORD IMMEDIATE instruction 7-42
COMPARE HALFWORD instruction 7-42
example A-14
COMPARE HFP instructions 18-10
examples A-40
COMPARE LOGICAL instructions 7-42
COMPARE LOGICAL CHARACTERS UNDER MASK instruction 7-43
example A-15
COMPARE LOGICAL instructions
examples A-14
COMPARE LOGICAL LONG EXTENDED instruction 7-45
COMPARE LOGICAL LONG instruction 7-43
example A-16
COMPARE LOGICAL LONG UNICODE instruction 7-47
COMPARE LOGICAL STRING instruction 7-50
examples A-17
COMPARE UNTIL SUBSTRING EQUAL instruction 7-51
comparison
address
*See* address comparison
between 370-XA and ESA/370 E-1
between ESA/370 and ESA/390 D-1
between System/370 and 370-XA F-1
decimal 8-6
example A-34
hexadecimal-floating-point
examples A-40
logical 7-4
examples A-14
of BFP data 19-8
signed-binary 7-4
TOD-clock 4-35
compatibility 1-12
among systems implementing different architectures 1-13



LzLabs GmbH. Ex. 1008-990

compatibility *(continued)*
among systems implementing same architecture 1-12
control-program 1-13
problem-state 1-13
completion of I/O functions
by channel-path-reset function 15-45
by clear function 15-15
by halt function 15-16
during data transfer 15-43
during initiation 15-42
for immediate commands 15-43
start and resume 15-42
completion of instruction execution 5-16
completion of unit of operation 5-18
compression facility
publication referenced xxii
COMPUTE INTERMEDIATE MESSAGE DIGEST instruction 7-55
COMPUTE LAST MESSAGE DIGEST instruction 7-55
COMPUTE MESSAGE AUTHENTICATION CODE instruction 7-61
conceptual sequence 5-78
as related to storage-operand accesses 5-90
conclusion of I/O operations 13-8, 16-1
during data transfer 15-43
during initiation 15-42
for immediate commands 15-43
prior to initiation 15-42
conclusion of instruction execution 5-16
concurrency of access for storage references 5-88
concurrent sense D-2
in ECW 16-40
indicator for (in ERW) 16-37
concurrent-sense count (in ERW) 16-37
concurrent-sense facility 17-21
condition code 4-6
deferred 16-8
for BFP instructions 19-9
in PSW 4-6
summary C-1
tested by BRANCH ON CONDITION instruction 7-17
used for decision making 5-10
validity bit for 11-22
conditional-swapping instructions
*See* COMPARE AND SWAP instruction, COMPARE DOUBLE AND SWAP instruction
conditions for interruption
*See* interruption conditions
configuration 2-1
of storage 3-4
configuration-alert facility (I/O) 17-21
connective
*See* logical connective
consistency (storage operand) 5-87
examples A-47, A-49
console device 12-1
console integration 1-9
control 4-1
instructions 10-1
manual
*See* manual operation
control-program compatibility 1-13
control register 2-4, 4-6
comparison, 370-XA with System/370 F-6
comparison, ESA/370 with 370-XA E-3
save areas 3-49
validity bit 11-23
control-register assignment 4-8
(CRx.y indicates control register x, bit position y)
CR0.1:
SSM-suppression-control bit 6-30, 10-87
CR0.2:
TOD-clock-sync-control bit 4-30, 4-34
CR0.3:
low-address-protection-control bit 3-12
CR0.4:
extraction-authority-control bit 5-22
CR0.5:
secondary-space-control bit 3-28, 5-22
CR0.6:
fetch-protection-override-control bit 3-11
CR0.7:
storage-protection-override-control bit 3-10
CR0.8-12:
translation format 3-28
CR0.13:
AFP-register-control bit 9-3
CR0.14:
vector-control bit 4-10
CR0.15:
address-space-function-control bit 5-42
CR0.16:
malfunction-alert subclass-mask bit 6-12
CR0.17:
emergency-signal subclass-mask bit 6-11
CR0.18:
external-call subclass-mask bit 6-12
CR0.19:
TOD-clock sync-check subclass-mask bit 6-13
CR0.20:
clock-comparator subclass-mask bit 6-11
CR0.21:
CPU-timer subclass-mask bit 6-11
CR0.22:
service-signal subclass-mask bit 6-13
CR0.25:
interrupt-key subclass-mask bit 6-12
CR0.27:
ETR subclass-mask bit 6-12



LzLabs GmbH. Ex. 1008-991

control-register assignment *(continued)*
CR0.28:
program-call-fast-control bit 5-23
CR0.29:
crypto-control bit 6-21
CR1:
primary segment-table designation (PSTD) 3-28
CR1.0:
primary space-switch-event-control bit 3-28, 6-29
CR1.1-19:
primary segment-table origin (PSTO) 3-28
CR1.22:
primary subspace-group-control bit 3-29
CR1.23:
primary private-space-control bit 3-29
CR1.24:
primary storage-alteration-event-control bit 3-29
CR1.25-31:
primary segment-table length (PSTL) 3-29
CR2.1-25:
dispatchable-unit-control-table origin (DUCTO) 5-43
CR3.0-15:
PSW-key mask (PKM) 5-22
CR3.16-31:
secondary ASN (SASN) 3-17
CR4.0-15:
authorization index (AX) 3-24, 5-23
CR4.16-31:
primary ASN (PASN) 3-17
CR5.0:
subsystem-linkage-control bit 5-22, 5-27
CR5.1-24:
linkage-table origin (LTO) 5-28
CR5.1-25:
primary-AST-entry origin (PASTEO) 5-28, 5-43
CR5.25-31:
linkage-table length (LTL) 5-28
CR6.0-7:
I/O-interruption subclass mask 6-14
CR7:
secondary segment-table designation (SSTD) 3-29
CR7.1-19:
secondary segment-table origin (SSTO) 3-29
CR7.22:
secondary subspace-group-control bit 3-29
CR7.23:
secondary private-space-control bit 3-29
CR7.24:
secondary storage-alteration-event-control bit 3-29
CR7.25-31:
secondary segment-table length (SSTL) 3-29
CR8.0-15:
extended authorization index (EAX) 5-43

control-register assignment *(continued)*
CR8.16-31:
monitor-mask bits 6-24
CR9.0:
PER successful-branching-event-mask bit 4-15
CR9.1:
PER instruction-fetching-event-mask bit 4-15
CR9.2:
PER storage-alteration-event-mask bit 4-15
CR9.3:
PER general-register-alteration-event-mask bit 4-15
CR9.4:
PER store-using-real-address-event-mask bit 4-15
CR9.8:
PER branch-address-control bit 4-15
CR9.10:
PER storage-alteration-space-control bit 4-15
CR9.16-31:
PER general-register-mask bits 4-16
CR10.1-31:
PER starting address 4-15
CR11.1-31:
PER ending address 4-15
CR12.0:
branch-trace-control bit 4-11
CR12.1-29:
trace-entry address 4-11
CR12.30:
ASN-trace-control bit 4-11
CR12.31:
explicit-trace-control bit 4-11
CR13:
home segment-table designation (HSTD) 3-29
CR13.0:
home space-switch-event-control bit 3-29, 6-29
CR13.1-19:
home segment-table origin (HSTO) 3-30
CR13.23:
home private-space-control bit 3-30
CR13.24:
home storage-alteration-event-control bit 3-30
CR13.25-31:
home segment-table length (HSTL) 3-30
CR14.2:
extended-save-area-control bit 4-44, 11-13
CR14.3:
channel-report-pending subclass-mask bit 11-26
CR14.4:
recovery subclass-mask bit 11-26
CR14.5:
degradation subclass-mask bit 11-26
CR14.6:
external-damage subclass-mask bit 11-26
CR14.7:
warning subclass-mask bit 11-26



LzLabs GmbH. Ex. 1008-992

control-register assignment *(continued)*
CR14.10:
TOD-clock-control-override control 4-30
CR14.12:
ASN-translation-control bit 3-18, 5-22
CR14.13-31:
ASN-first-table origin (AFTO) 3-19
CR15.1-28:
linkage-stack-entry address 5-68
control unit 2-7, 13-4
effect of I/O-system reset on 17-14
sharing of 13-4
type of 15-13
control unit defer time (I/O) 17-9
control unit defer time interval (in measurement block) 17-7
control-unit-defer time(I/O)
control-unit-queuing measurement (I/O) 17-9
control-unit-queuing-time interval (in measurement block) 17-6
conversion
between HFP and BFP data 9-8
binary-to-decimal 7-67
example A-18
decimal-to-binary 7-66
example A-18
hexadecimal-floating-point-number
basic example A-6
examples with instructions A-42
of floating-point format 19-7
CONVERT BFP TO HFP floating-point instructions 9-8
CONVERT FROM FIXED BFP instructions 19-26
CONVERT FROM FIXED HFP instructions 18-11
CONVERT HFP TO BFP floating-point instructions 9-9
CONVERT TO BINARY instruction 7-66
example A-18
CONVERT TO DECIMAL instruction 7-67
example A-18
CONVERT TO FIXED BFP instructions 19-26
CONVERT TO FIXED HFP instructions 18-12
CONVERT UNICODE TO UTF-8 instruction 7-67
CONVERT UTF-8 TO UNICODE instruction 7-70
Coordinated Universal Time (UTC) used in TOD epoch 4-32
COPY ACCESS instruction 7-72
count field
in CCW 15-29
invalid 16-24
in SCSW 16-29
counter updating (example) A-45
counting operations 7-18
coupling facility 1-10
CP (COMPARE DECIMAL) instruction 8-6
example A-34
CPA
*See* channel-program address
CPU (central processing unit) 2-2
address 4-45
assigned storage locations for 3-44
when stored during external interruptions 6-10
checkpoint 11-2
effect of power-on reset on 4-42
hangup due to string of interruptions 4-3
identification (ID) 10-90
machine-type number 10-90
model number 10-90
registers 2-3
save areas for 3-48
reset 4-40
signal-processor order 4-46
retry 11-2
serialization 5-91
signaling 4-45
state 4-1
check-stop 4-3
load 4-2
no effect on TOD clock 4-29
operating 4-2
stopped 4-2
version code 10-90
CPU timer 4-35
external interruption 6-11
save areas for 3-48
validity bit for 11-23
CPYA (COPY ACCESS) instruction 7-72
CR
*See* control register
CR (COMPARE) binary instruction 7-35
CRW (channel-report word) 17-23
chaining of 17-23, 17-24
error-recovery code (ERC) in 17-24
overflow in 17-24
reporting-source code (RSC) in 17-24
reporting-source ID (RSID) in 17-25
solicited 17-23
storing of 14-17
crypto-operation exception 6-21
cryptographic facility 1-8, 2-6
CS (COMPARE AND SWAP) instruction 7-40
examples A-44
CSCH (CLEAR SUBCHANNEL) instruction 14-5
CSP (COMPARE AND SWAP AND PURGE) instruction 10-17
current CCW 15-27
*See also* CCW
current PSW 4-3, 5-9
*See also* PSW
stored during interruption 6-2
CUSE (COMPARE UNTIL SUBSTRING EQUAL) instruction 7-51
CUTFU (CONVERT UTF-8 TO UNICODE) instruction 7-70


LzLabs GmbH. Ex. 1008-993

CUUTF (CONVERT UNICODE TO UTF-8) instruction 7-67
CVB (CONVERT TO BINARY) instruction 7-66
example A-18
CVD (CONVERT TO DECIMAL) instruction 7-67
example A-18
CXBR (COMPARE) BFP instruction 19-23
CXFBR (CONVERT FROM FIXED) BFP instruction 19-26
CXFR (CONVERT FROM FIXED) HFP instruction 18-11
CXR (COMPARE) HFP instruction 18-10

# D

D (DIVIDE) binary instruction 7-73
example A-19
D field of instruction 5-8
damage
channel-subsystem 11-19
code (external) 11-24
validity bit for 11-22
external 11-18
subclass-mask bit for 11-26
instruction-processing 11-17
processing 11-20
service-processor 11-19
system 11-16
timing-facility 11-17
DAT
*See* dynamic address translation
DAT mode (bit in PSW) 4-5
use in address translation 3-28
DAT-table format error 6-34
data
blocking of (I/O) 15-22
format for
binary-floating-point instructions 19-4
decimal instructions 8-1
general instructions 7-2
hexadecimal-floating-point instructions 18-3
indirect addressing of (I/O) 13-7, 15-36
measurement (I/O)
*See* measurement data
prefetching of for I/O operation 15-30
data address (I/O) 15-30
invalid 16-25
invalid specification of 16-25
data chaining of CCWs 15-33
flag in CCW for 15-28
overview of 13-8
data check
measurement-block 16-33
data exception 6-21
AFP-register 6-21
BFP-instruction 6-21
data exception *(continued)*
decimal-operand 6-21, 8-4
IEEE-exception-condition 6-21, 19-10
priority of program interruptions for 6-15
data-exception code
*See* DXC
data streaming (I/O) 13-3
effect of CCW count on 15-34
DCTI (device-connect-time interval)
in ESW 16-39
in extended-measurement word 16-41
in measurement block 17-5
DD (DIVIDE) HFP instruction 18-12
DDB (DIVIDE) BFP instruction 19-29
DDBR (DIVIDE) BFP instruction 19-29
DDR (DIVIDE) HFP instruction 18-12
DE (DIVIDE) HFP instruction 18-12
DEB (DIVIDE) BFP instruction 19-29
DEBR (DIVIDE) BFP instruction 19-29
decimal
arithmetic 8-2
comparison 8-6
digit codes 8-2
divide exception 6-22
instructions 8-1
examples A-34
number representation 8-1
examples A-4
operand overlap 8-3
overflow
exception 6-22
mask in PSW 4-6
sign codes 8-2
decimal-operand data exception 6-21, 8-4
decimal-to-binary conversion 7-66
example A-18
dedicated allegiance 15-12
default QNaN 19-6
deferred condition code 16-8
degradation (machine-check condition) 11-18
subclass-mask bit for 11-26
degradation, storage (machine-check condition) 11-21
delay in storing 5-85
delayed access exception (machine-check condition) 11-19
deletion of malfunctioning unit 11-4
denormalized numbers 19-8
DER (DIVIDE) HFP instruction 18-12
examples A-40
designation
access-list 5-44
authority-table 3-20
effective segment-table 3-32
entry-table 5-28
home segment-table 3-29
linkage-table 5-27
in AST entry 3-21



LzLabs GmbH. Ex. 1008-994

designation *(continued)*
of storage area for data (I/O) 15-30
page-table 3-30
primary segment-table 3-28
secondary segment-table 3-29
segment-table 3-28
in AST entry 3-20
destructive overlap 5-89, 7-84, 7-88, 7-91
in the access-register mode 5-80
device 2-7, 13-4
console 12-1
effect of I/O-system reset on 17-14
device-active bit 16-15
device-active-only measurement (I/O) 17-9
device-active-only-time interval (in measurement block) 17-6
device address 13-5
device-busy time (in extended measurement word) 16-42
device-busy time (in measurement block) 17-6
device-connect-time interval
*See* DCTI
device-connect-time measurement 17-10
effect of suspension on 15-40
enable 15-3
device-disconnect-time interval (in measurement block) 17-5
device identifier 13-5
device number 13-5
assignment of 13-5
in PMCW 15-4
device-number valid (bit in PMCW) 15-4
device status 16-23
field-validity flag for (in subchannel logout) 16-28, 16-34
with inappropriate bit combination 16-35
device status check 16-35
DIAGNOSE instruction 10-19
DIDBR (DIVIDE TO INTEGER) BFP instruction 19-30
DIEBR (DIVIDE TO INTEGER) BFP instruction 19-30
digit codes (decimal) 8-2
digit selector (in EDIT) 8-8
direct-access storage 3-1
disabling for interruptions 6-6
disallowed interruptions 6-6
dispatchable unit (DU) 5-36
access-list designation (DUALD) 5-44
control table (DUCT) 5-44
origin (DUCTO) 5-43
when branch-and-set-authority facility installed 10-7
when subspace-group facility installed 5-56
when trap facility installed 10-112
displacement (in relative addressing) 5-8
display (manual controls) 12-2
DIVIDE BFP instructions 19-29
DIVIDE binary instructions 7-73
example A-19
DIVIDE DECIMAL instruction 8-7
example A-34
divide exception
decimal 6-22
fixed-point 6-23
HFP 6-23
DIVIDE HFP instructions 18-12
examples A-40
DIVIDE LOGICAL instructions 7-73
DIVIDE TO INTEGER BFP instructions 19-30
remainder result of 19-9
divisible instruction execution 5-79
DL (DIVIDE LOGICAL) instruction 7-73
DLR (DIVIDE LOGICAL) instruction 7-73
doubleword 3-3
doubleword-concurrent storage references 5-88
DP (DIVIDE DECIMAL) instruction 8-7
example A-34
DR (DIVIDE) binary instruction 7-73
DU (dispatchable unit) 5-36
DUALD (dispatchable-unit access-list designation) 5-44
DUCT (dispatchable-unit control table) 5-44, 5-56
DUCTO (dispatchable-unit-control-table origin) 5-43
dump (standalone) 12-4
DXBR (DIVIDE) BFP instruction 19-29
DXC (data-exception code) 6-15, 19-14
summary figure 19-15
summary figure for IEEE 19-14
DXR (DIVIDE) HFP instruction 18-12
dynamic address translation (DAT) 3-26
by LOAD REAL ADDRESS instruction 10-38
control of 3-27
explicit and implicit 3-31
mode bit in PSW 4-5
use in address translation 3-28
sequence of table fetches 5-83
dynamic-reconnection feature 13-2

# E

E instruction format 5-5
EAR (EXTRACT ACCESS) instruction 7-75
early exception recognition 6-9
EAX
*See* extended authorization index
EBCDIC (Extended Binary-Coded-Decimal Interchange Code)
architecture designed for xxi
character code
table for I-1
ECC (error checking and correction) 11-2



LzLabs GmbH. Ex. 1008-995

ECW (extended-control word) 16-40
indication in SCSW 16-11
ED (EDIT) instruction 8-7
examples A-35
EDIT AND MARK instruction 8-12
example A-36
EDIT instruction 8-7
examples A-35
editing instructions 8-3
*See also* ED instruction, EDMK instruction
EDMK (EDIT AND MARK) instruction 8-12
example A-36
effective access-list designation 5-44
effective address 3-5
controlled by addressing mode 5-7
generation 5-7
used for storage interlocks 5-80
effective segment-table designation 3-32
EFPC (EXTRACT FPC) instruction 19-34
EKM (entry key mask) 5-29
use by stacking PROGRAM CALL 5-65
emergency signal (external interruption) 6-11
signal-processor order 4-45
EMIF (ESCON-multiple-image facility) 1-9
EMW (extended-measurement word) 16-40
in IRB 16-40
enabled (bit for TRAP) 10-112
enabled (bit in PMCW) 15-2
enabling for interruptions 6-6
subchannel 16-5
enabling of subchannel 15-2, 16-5
endian 7-81
ending of instruction execution 5-16
Enterprise Systems Connection Architecture (ESCON) I/O interface
publication referenced xxi
entry
addressing-mode bit 5-65
extended authorization index 5-66
instruction address 5-65
key 5-66
parameter 5-65
problem-state bit 5-65
entry (for tracing) 4-11
entry descriptor 5-69
entry index (EX) 5-27
entry key mask (EKM) 5-29
use by stacking PROGRAM CALL 5-65
entry table (ET)
designation 5-28
length (ETL) 5-28
origin (ETO) 5-28
entry-table entry (ETE)
basic (16 byte) 5-28
extended (32 byte) 5-64
entry-type code 5-69
EPAR (EXTRACT PRIMARY ASN) instruction 10-19
epoch (for TOD clock) 4-32
EPSW (EXTRACT PSW) instruction 7-76
equipment check
in signal-processor status 4-52
ERC (error-recovery code) 17-24
*See also* CRW
EREG (EXTRACT STACKED REGISTERS) instruction 10-20
error
checking and correction 11-2
from DIAGNOSE instruction 10-19
I/O-error alert 16-35
indirect storage 11-21
intermittent 11-5
PSW-format 6-9
secondary (I/O) 16-35
solid 11-5
state of TOD clock 4-30
storage 11-20
storage-key 11-21
error-recovery code (ERC) 17-24
*See also* CRW
ERW (extended-report word) 16-32, 16-36
as result of channel-control check 16-27
as result of channel-data check 16-27
ESA/370 architecture 1-10
architectural-mode controls 12-2
comparison of facilities with 370-XA E-1
comparison with ESA/390 D-1
facilities E-1
ESA/390 architecture
comparison of facilities with ESA/370 D-1
highlights of 1-1
ESAR (EXTRACT SECONDARY ASN) instruction 10-20
ESCON (Enterprise Systems Connection Architecture) 13-2
ESCON (Enterprise Systems Connection Architecture) I/O interface
publication referenced xxi
ESCON channel-to-channel adapter
publication referenced xxi
ESCON-multiple-image facility (EMIF) 1-9
ESTA (EXTRACT STACKED STATE) instruction 10-21
ESW (extended-status word) 16-32
*See also* extended status
ESW format bit (in SCSW) 16-8
ET
*See* entry table
ETE
*See* entry-table entry
ETL (entry-table length) 5-28



LzLabs GmbH. Ex. 1008-996

ETO (entry-table origin) 5-28
ETR
  external interruption 6-12
ETR (external time reference) 2-6
ETR (external time reference) facility 1-8
ETR sync check (machine-check condition) 11-25
event 6-14
  monitor 7-82
  PER 4-14
  space-switch 6-29
EX (entry index) 5-27
  translation exception 6-22
EX (EXECUTE)
  *See* EXECUTE instruction
exception access identification 3-46
exception-extension code 6-15
exceptions 6-14
  access (collective program-interruption name) 6-35, 6-40
  addressing 6-16
  AFX-translation 6-19
  ALE-sequence 6-19
  ALEN-translation 6-19
  ALET-specification 6-19
  ASN-translation (collective program-interruption name) 6-45
  ASN-translation-specification 6-19
  associated with
    ART 5-53
    stacking process 5-75
    unstacking process 5-78
  ASTE-sequence 6-20
  ASTE-validity 6-20
  ASX-translation 6-21
  comparison of ESA/370 with 370-XA E-4
  crypto-operation 6-21
  data 6-21
  decimal-divide 6-22
  decimal-overflow 6-22
  delayed access (machine-check condition) 11-19
  during translation 3-35
  EX-translation 6-22
  execute 6-22
  extended-authority 6-22
  fixed-point-divide 6-23
  fixed-point-overflow 6-23
  HFP-divide 6-23
  HFP-exponent-overflow 6-23
  HFP-exponent-underflow 6-24
  HFP-significance 6-24
  HFP-square-root 6-24
  IEEE 19-10
  LX-translation 6-24
  operand (of I/O instruction) 6-25
  operation 6-25
  page-translation 6-26
exceptions *(continued)*
  PC-translation-specification 6-27
  primary-authority 6-27
  privileged-operation 6-27
  protection 6-28
  PSW-related 6-9
  recognition of
    early and late 6-9
  secondary-authority 6-29
  segment-translation 6-29
  special-operation 6-30
  specification 6-31
  stack-empty 6-33
  stack-full 6-33
  stack-operation 6-33
  stack-specification 6-33
  stack-type 6-34
  subspace-replacement (collective program-interruption name) 6-45
  trace (collective program-interruption name) 6-45
  trace-table 6-34
  translation-specification 6-34
  unnormalized-operand 6-35
  vector-operation 6-35
EXCLUSIVE OR instructions 7-74
  examples A-19
execute exception 6-22
EXECUTE instruction 7-74
  effect of address comparison on 12-1
  example A-21
  exceptions while fetching target of 6-8
  PER event for target of 4-22
exigent machine-check conditions 11-11
expanded storage 2-2
  accessed by MOVE PAGE 2-2, 7-93
  block number 2-2
  control failure (machine-check condition) 11-24
  not operational (machine-check condition) 11-24
explicit address translation 3-31
explicit-trace-control bit 4-11
exponent 18-1
  *See also* characteristic, floating point
  *See also* floating point
  overflow
    HFP 18-1
  underflow
    HFP 18-1
    mask in PSW 4-6
exponent bias 19-4
extended AST entry 5-47
extended-authority exception 6-22
  as an access exception 6-35
extended authorization 5-52
extended authorization index (EAX) 5-43
  control bit 5-65
  in entry-table entry 5-66



LzLabs GmbH. Ex. 1008-997

extended authorization index (EAX) *(continued)*
in linkage-stack state entry 5-72
extended binary-floating-point number 19-4
extended control (bit in SCSW) 16-11
extended-control word 16-40
*See also* ECW
extended entry-table entry 5-64
extended hexadecimal-floating-point number 18-3
extended-measurement-word 17-11
extended measurement word (I/O)
update enable 15-8
extended-measurement-word enable (I/O)
extended measurement word mode enable (I/O) 15-8
extended-report word
*See* ERW
extended save area 4-44
address 3-47
extended-sorting facility 1-9
extended status
*See also* ESW
flags in subchannel logout for 16-32
format-0 16-32
format-1 16-38
format-2 16-38
format-3 16-39
secondary-CCW address 16-38
extended-status word 16-32
*See also* extended status
extended-status-word-format bit 16-8
external call
external interruption 6-12
pending (signal-processor status) 4-52
signal-processor order 4-45
external damage 11-18
subclass-mask bit for 11-26
external-damage code 11-24
assigned storage locations for 3-48
validity bit for 11-22
external interruption 6-10
clock-comparator 4-35, 6-11
CPU-timer 4-36, 6-11
direct conditions 6-10
emergency-signal 6-11
ETR 6-12
external-call 6-12
interrupt-key 6-12
malfunction-alert 6-12
mask in PSW 4-5
parameter 6-10
assigned storage locations for 3-44
pending conditions 6-10
priority of conditions 6-10
service-signal 6-12
TOD-clock-sync-check 6-13
external time reference (ETR) 2-6
external-time-reference (ETR) facility 1-8
externally initiated functions 4-37
I/O 17-17
EXTRACT ACCESS instruction 7-75
EXTRACT FPC instruction 19-34
EXTRACT PRIMARY ASN instruction 10-19
EXTRACT PSW instruction 7-76
EXTRACT SECONDARY ASN instruction 10-20
EXTRACT STACKED REGISTERS instruction 10-20
EXTRACT STACKED STATE instruction 10-21
extraction-authority-control bit 5-22

# F

facilities of 370-XA (compared with System/370) F-1
facilities of ESA/370 (compared with 370-XA) E-1
facilities of ESA/390 (compared with ESA/370) D-1
failing-storage address 11-25
assigned storage locations for 3-48
in ESW 16-32, 16-37
as result of channel-control check 16-27
as result of channel-data check 16-27
validity bit for 11-22
validity flag for (in ERW) 16-37
failing-storage-address format
indicator for (in ERW) 16-37
failure
vector-facility 11-18
fetch-only bit 5-46
fetch protection 3-9
bit in storage key 3-8
override-control bit 3-11
fetch reference 5-85
access exceptions for 6-38
fetching
handling of invalid CBC in storage keys during 11-8
of ART-table and DAT-table entries 5-83
of instructions 5-82
of PSWs during interruptions 5-90
of storage operands 5-85
FICON I/O interface
publication referenced xxi
FIDBR (LOAD FP INTEGER) BFP instruction 19-36
FIDR (LOAD FP INTEGER) HFP instruction 18-15
FIEBR (LOAD FP INTEGER) BFP instruction 19-36
field 3-2
field separator (in EDIT) 8-8
field-validity flags (in subchannel logout) 16-34
relation to channel-control check of 16-28
FIER (LOAD FP INTEGER) HFP instruction 18-15
FIFO (first in first out) queuing
example for lock and unlock A-47
fill byte (in EDIT) 8-8
FIXBR (LOAD FP INTEGER) BFP instruction 19-36
fixed-length field 3-2



LzLabs GmbH. Ex. 1008-998

fixed logout
assigned storage locations for 3-48
machine-check 11-27
fixed point
*See also* binary
divide exception 6-23
overflow exception 6-23
mask in PSW 4-6
FIXR (LOAD FP INTEGER) HFP instruction 18-15
flags
for BFP arithmetic exceptions 19-3
for floating-point arithmetic exceptions 19-3
for IEEE exception conditions 19-3
floating interruption conditions 6-6, 11-25
clearing of 4-41
floating point
*See also* exponent
binary (BFP) 9-1
binary data format 19-4
conversion
between formats 19-7
conversion from binary integer 18-11, 19-26
conversion to binary integer 18-12, 19-26
data
lengthening format of 18-16, 19-38
shortening format of 18-18, 19-39
data class 19-5
hexadecimal (HFP) 9-1
hexadecimal data format 18-3
instructions 9-1
numbers 19-4
registers 2-3, 9-2
clearing of 9-11
save areas for 3-48
validity bit for 11-23
shifting
*See* normalization
floating-point-control (FPC) register 19-2
format
address 3-2
binary-floating-point data 19-4
CCW
*See* CCW format control
decimal data 8-1
error
in DAT-table entry 6-34
in PSW 6-9
general data 7-2
hexadecimal-floating-point data 18-3
information 3-2
instruction 5-3
PSW 4-5
format-0 access-list designation 5-45
format-0 and format-1 CCWs 15-28
format-1 access-list designation 5-45
format-2-IDAW control 15-25
format control 15-24
forward-section-header address 5-70
forward-section validity bit 5-70
FPC (floating-point-control) register 19-2
fraction 18-1
free-pool manipulation
programming example A-48
fullword
*See* word
function control (I/O) 16-12
function-pending time 17-3
in extended measurement word 16-41
in measurement block 17-5

# G

G (giga) xxi
general instructions 7-2
examples A-7
general registers 2-3
alteration-event-mask bit 4-15
alteration of (PER event) 4-23
PER-mask bits 4-16
save areas for 3-49
validity bit for 11-23
glue module 5-15
GMT (Greenwich Mean Time) obsolete term for UTC 4-32
Greenwich Mean Time (GMT) obsolete term for Coordinated Universal Time 4-32
guard digit 18-4

# H

halfword 3-3
halfword-concurrent storage references 5-88
halt function 15-15
bit in SCSW for 16-12
completion of 15-16
initiated by HALT SUBCHANNEL 14-6
path management for 15-15
pending 16-15
signaling for 15-16
halt signal 17-12
issued as part of halt function 15-16
HALT SUBCHANNEL instruction 14-6
*See also* halt function
effect on SCSW count field 15-18
function initiated by 15-15
use of after RESET CHANNEL PATH 14-9
HALVE HFP instructions 18-13
example A-41
HDR (HALVE) HFP instruction 18-13
example A-41



LzLabs GmbH. Ex. 1008-999

header entry 5-70
HER (HALVE) HFP instruction 18-13
hex
  *See* hexadecimal
hexadecimal (hex) representation 5-6
hexadecimal floating point
  conversion
    examples with instructions A-42
  instructions
    examples A-39
hexadecimal floating point (HFP) 9-1
  conversion
    basic example A-6
hexadecimal-floating-point number
  examples A-5
HFP (hexadecimal floating point) 9-1
HFP data 18-3
  conversion of 9-8
HFP exponent
  overflow
    exception 6-23
  underflow
    exception 6-24
    mask in PSW 4-6
HFP significance
  exception 6-24
  mask (in PSW) 4-6
HFP square root
  exception 6-24
high-speed data transfer (I/O) 13-3
home address space 3-16, 5-33
  facilities 5-33
home segment table
  designation (HSTD) 3-29
  length (HSTL) 3-30
  origin (HSTO) 3-30
home-space mode 3-28
home space-switch-event-control bit 3-29
home storage-alteration-event-control bit 3-30
home virtual address 3-4
  effective segment-table designation for 3-32
HSCH (HALT SUBCHANNEL) instruction 14-6
HSTD (home segment-table designation) 3-29
HSTL (home segment-table length) 3-30
HSTO (home segment-table origin) 3-30

## I

I field of instruction 5-6
I instruction format 5-5
I/O (input/output) 2-6
  basic functions of 15-1
  blocking of data for 15-22
  comparison of 370-XA with System/370 F-5
  effect on CPU timer 4-36
  sense data
    *See* sense data
I/O (input/output) *(continued)*
  support functions of 17-1
I/O addressing 13-5
I/O commands
  *See also* commands
  publication referenced xxi
I/O device
  *See* device
I/O-error alert (in subchannel logout) 16-35
I/O instructions 14-1, 14-2
  deferred condition code for 16-8
  operand access by 14-1
  role of in I/O operations 13-6
I/O interface
  ESCON publication referenced
  OEMI publication referenced xxi
I/O interruption 6-13, 16-1
  *See also* interruption
  action for 16-5
  masking of 13-9
  priority of 16-4
  program-controlled interruption
    *See* PCI
I/O-interruption code 6-13
I/O-interruption condition 13-9, 16-2
  alert 16-4
  intermediate 16-4
  primary 13-8, 16-4
  secondary 13-8, 16-4
  solicited 16-3
  unsolicited 16-3
I/O-interruption-identification word
  assigned storage locations for 3-47
I/O-interruption parameter
  assigned storage locations for 3-47
  in ORB 15-22
  in PMCW 15-2
  used for IPL 17-18
I/O-interruption request
  clearing of 13-9
  from subchannels 16-5
I/O-interruption subclass 13-9
I/O-interruption subclass mask 6-14, 16-5
  relation to priority 16-4
I/O mask in PSW 4-5
I/O operations 13-6
  conclusion of
    *See* conclusion of I/O operations
  immediate 15-43
  initiated indication for 16-11
  termination of
    *See* conclusion of I/O operations
I/O-system reset 17-13
  as part of subsystem reset 4-41
I/O-interruption code 14-19
  interruption-identification word in 14-19



LzLabs GmbH. Ex. 1008-1000

I/O-interruption code *(continued)*
stored by TPI 14-19
I/O-interruption parameter
in I/O-interruption code 14-19
I/O-interruption-identification word 14-19
IAC (INSERT ADDRESS SPACE CONTROL) instruction 10-23
IC (INSERT CHARACTER) instruction 7-76
IC (instruction counter)
*See* instruction address
ICM (INSERT CHARACTERS UNDER MASK) instruction 7-76
examples A-21
ID
*See* CPU identification, sense ID
IDA (indirect-data address) 15-36
flag in CCW 15-29
IDAW (indirect-data-address word) 15-36
check (in subchannel logout) 16-33
invalid address of 16-24
invalid address specification in 16-24
invalid address specification of 16-25
idle state for subchannel 16-13
IEEE-exception-condition data exception 6-21, 19-10
IEEE exception conditions 19-14
summary figure 19-14
IEEE standard 1-4
IFCC (interface-control check) 16-28
ILC (instruction-length code) 6-7
assigned storage locations for 3-44
for program interruptions 6-14
for supervisor-call interruption 6-46
IML (initial machine loading) controls 12-3
immediate operand 5-6
immediate operation
SLI flag in CCW for 15-31
immediate operation (I/O) 15-43
implicit address translation 3-31
incorrect length (subchannel status) 16-23
for immediate operations 15-31
incorrect-length-indication mode 15-25
incorrect-length-indication-suppression facility 17-21, F-2
effect on immediate operation 15-31
incorrect-length-suppression mode 15-25
incorrect state (signal-processor status) 4-52
index
for address generation 5-8
instructions for branching on 7-18
into access list 5-44
into ASN first and second tables 3-18
into authority table 5-23
into entry and linkage tables 5-27
register for 2-3
indicator
check-stop 12-2
indicator *(continued)*
load 12-3
manual 12-3
mode 12-2
test 12-5
wait 12-5
indirect-data address
*See* IDA
indirect-data-address word
*See* IDAW
indirect storage error 11-21
infinities 19-6
information format 3-2
inhibition of unit of operation 5-18
initial-command-response measurement (I/O) 17-10
initial-command-response time (in measurement block) 17-6
initial CPU reset 4-41
signal-processor order 4-46
initial-machine-loading (IML) controls 12-3
initial program loading
*See* IPL
initial-status-interruption control 15-24, 16-11
relation to Z bit 16-11
used for IPL 17-18
inoperative (signal-processor status) 4-53
input/output
*See* I/O
INSERT ADDRESS SPACE CONTROL instruction 10-23
INSERT CHARACTER instruction 7-76
INSERT CHARACTERS UNDER MASK instruction 7-76
examples A-21
INSERT PROGRAM MASK instruction 7-77
INSERT PSW KEY instruction 10-24
INSERT STORAGE KEY EXTENDED instruction 10-25
INSERT VIRTUAL STORAGE KEY instruction 10-25
installation 2-1
instruction address
as a type of address 3-5
handling by DAT 3-28
in entry-table entry 5-29
in PSW 4-6
validity bit for 11-22
instruction-length code
*See* ILC
instruction-processing damage 11-17
resulting in processing backup 11-20
resulting in processing damage 11-20
instructions
*See also* instruction lists and page numbers in Appendix B
backing up of 11-20
classes of 2-2


LzLabs GmbH. Ex. 1008-1001

instructions *(continued)*
comparison of 370-XA with System/370 F-3
comparison of ESA/370 with 370-XA E-2
control 10-1
damage to 11-17, 11-20
decimal 8-1
examples A-34
divisible execution of 5-79
ending of 5-16
examples of use A-6
execution of 5-9
fetching of 5-82
access exception for 6-37
PER event for 4-22
PER-event mask for 4-15
floating-point 9-1
format of 5-3
general 7-2
examples A-7
hexadecimal-floating-point
examples A-39
I/O
*See* I/O instructions
interruptible
*See* interruptible instructions
length of 5-5
list of B-1
modification by EXECUTE instruction 7-74
prefetching of 5-82
privileged 4-6
for control 10-1
semiprivileged 4-6, 10-1
sequence of execution of 5-2
stepping of (rate control) 12-4
effect on CPU state 4-2
effect on CPU timer 4-36
unprivileged 4-6, 7-2
vector 2-6
integer
binary 7-2
address as 5-8
conversion from floating point 18-12, 19-26
conversion to floating point 18-11, 19-26
examples A-2
decimal 8-2
integer quotient 19-30
integral boundary 3-3
interface
ESCON I/O
publication referenced xxi
parallel-I/O
OEMI publication referenced xxi
serial-I/O
publication referenced xxi
interface-control check (subchannel status) 16-28
interlocked-update storage reference 5-86
interlocks for virtual storage references 5-79
intermediate interruption condition (I/O) 16-4
intermediate-status bit (I/O) 16-17
intermittent errors 11-5
International Atomic Time (TAI) related to Coordinated Universal Time 4-32
interpretive execution
publication referenced xxii
interpretive-execution facility 1-10
interrupt key 12-3
external interruption 6-12
interruptible instructions 5-17
COMPARE AND FORM CODEWORD 7-36
COMPARE LOGICAL LONG 7-44
COMPARE UNTIL SUBSTRING EQUAL 7-53
MOVE LONG 7-84
PER event affecting the ending of 4-20
stopping of 4-2
TEST BLOCK 10-108
UPDATE TREE 7-141
vector instructions 5-17
interruption 6-1
*See also* masks
action 6-2
I/O 16-5
machine-check 11-12
classes of 6-5
effect on instruction sequence 5-16
external
*See* external interruption
I/O
*See* I/O interruption
machine-check
*See* machine-check interruption
masking of 6-6
pending 6-6
external 6-10
machine-check 11-13
relation to CPU state 4-2
priority of
*See* priority
program
*See* program interruption
program-controlled (I/O)
*See* PCI
restart 6-46
string
*See* string of interruptions
supervisor-call 6-46
interruption code 6-5
external 6-10
I/O
*See* I/O-interruption code
machine-check (MCIC) 3-48, 11-15
program 6-14



LzLabs GmbH. Ex. 1008-1002

interruption code *(continued)*
summary of 6-2
supervisor-call 6-46
interruption conditions 6-1
clearing of 4-40
floating 6-6, 11-25
I/O
*See* I/O-interruption condition
interruption parameter
external (assigned storage locations) 3-44
I/O
*See* I/O-interruption parameter
interruption-response block
*See* IRB
interruption subclass
*See* I/O-interruption subclass
invalid
access-list entry 5-46
address 6-16
bit in ASN-first-table entry 3-19
bit in ASN-second-table entry 3-20
bit in linkage-table entry 5-28
bit in page-table entry 3-31
bit in segment-table entry 3-30
CBC 11-2
in registers 11-10
in storage 11-6
in storage keys 11-7
operation code 6-25
order (signal-processor status) 4-53
parameter (signal-processor status) 4-52
translation address 3-35
translation format 3-28
exception recognition 3-35
invalid address specification
in channel-program address 16-24
in IDAW 16-25
of data in CCW 16-25
of IDAW 16-24
of TIC CCW 16-24
invalid CCW field
command code 16-24
count 16-24
data address 16-25
suspend flag 16-25
invalid format
of CCW 16-25
of ORB 16-26
invalid sequence of CCWs 16-26
INVALIDATE PAGE TABLE ENTRY instruction 10-26
effect of when CPU is stopped 4-2
inverse move
*See* MOVE INVERSE instruction, move-inverse facility
IPK (INSERT PSW KEY) instruction 10-24
IPL (initial program loading) 4-43, 17-17
assigned storage locations for 3-43
effect on CPU state 4-2
IPM (INSERT PROGRAM MASK) instruction 7-77
IPTE (INVALIDATE PAGE TABLE ENTRY) instruction 10-26
IRB (interruption-response block) 16-6
*See also* ECW, ERW, ESW, SCSW
storage requirements for 16-11
ISC (I/O-interruption-subclass code) 15-2
ISC (I/O-interruption-subclass code)
enhanced 14-19
ISKE (INSERT STORAGE KEY EXTENDED) instruction 10-25
isolated state 16-36
IVSK (INSERT VIRTUAL STORAGE KEY) instruction 10-25

# K

K (kilo) xxi
KDB (COMPARE AND SIGNAL) BFP instruction 19-24
KDBR (COMPARE AND SIGNAL) BFP instruction 19-24
KEB (COMPARE AND SIGNAL) BFP instruction 19-24
KEBR (COMPARE AND SIGNAL) BFP instruction 19-24
key
access
*See* access key
manual
*See* manual operation
PSW
*See* PSW key
storage
*See* storage key
subchannel
*See* subchannel key
key check (in subchannel logout) 16-32
key-controlled protection 3-9
exception for 6-28
key mask
authorization 5-29
entry 5-29
PSW (PKM) 5-22
KIMD (COMPUTE INTERMEDIATE MESSAGE DIGEST) instruction 7-55
KLMD (COMPUTE LAST MESSAGE DIGEST) instruction 7-55
KM (CIPHER MESSAGE) instruction 7-26
KMAC (COMPUTE MESSAGE AUTHENTICATION CODE) instruction 7-61
KMC (CIPHER MESSAGE WITH CHAINING) instruction 7-26
KXBR (COMPARE AND SIGNAL) BFP instruction 19-24


LzLabs GmbH. Ex. 1008-1003

# L


L (LOAD) binary instruction 7-77
  example A-22
L fields of instruction 5-7
LA (LOAD ADDRESS) instruction 7-78
  examples A-22
LAE (LOAD ADDRESS EXTENDED) instruction 7-78
LAM (LOAD ACCESS MULTIPLE) instruction 7-77
LARL (LOAD ADDRESS RELATIVE LONG) instruction 7-79
LASP (LOAD ADDRESS SPACE PARAMETERS) instruction 10-28
last-path-used mask
  *See* LPUM
late exception recognition 6-9
LCDBR (LOAD COMPLEMENT) BFP instruction 19-35
LCDR (LOAD COMPLEMENT) HFP instruction 18-15
LCEBR (LOAD COMPLEMENT) BFP instruction 19-35
LCER (LOAD COMPLEMENT) HFP instruction 18-15
LCR (LOAD COMPLEMENT) binary instruction 7-79
LCTL (LOAD CONTROL) instruction 10-36
LCXBR (LOAD COMPLEMENT) BFP instruction 19-35
LCXR (LOAD COMPLEMENT) HFP instruction 18-15
LD (LOAD) floating-point instruction 9-10
LDE (LOAD LENGTHENED) HFP instruction 18-16
LDEB (LOAD LENGTHENED) BFP instruction 19-38
LDEBR (LOAD LENGTHENED) BFP instruction 19-38
LDER (LOAD LENGTHENED) HFP instruction 18-16
LDR (LOAD) floating-point instruction 9-10
LDXBR (LOAD ROUNDED) BFP instruction 19-39
LDXR (LOAD ROUNDED) HFP instruction 18-18
LE (LOAD) floating-point instruction 9-10
LEDBR (LOAD ROUNDED) BFP instruction 19-39
LEDR (LOAD ROUNDED) HFP instruction 18-18
left-to-right addressing 3-2
length
  field 3-2
  instruction 5-5
  of BFP data
    decreasing 19-39
    increasing 19-38
  of HFP data
    decreasing 18-18
    increasing 18-16
  register-operand 5-6
  second operand same as first 5-6
  variable (storage operand) 5-7
LER (LOAD) floating-point instruction 9-10
LEXBR (LOAD ROUNDED) BFP instruction 19-39
LEXR (LOAD ROUNDED) HFP instruction 18-18
LFPC (LOAD FPC) instruction 19-37
LH (LOAD HALFWORD) instruction 7-80
  examples A-23
LHI (LOAD HALFWORD IMMEDIATE) instruction 7-80
LIFO (last in first out) queuing
  example for lock and unlock A-46
light
  *See* indicator
limit mode (I/O) 15-2
link information
  for BRANCH AND LINK instruction 7-14
  for BRANCH AND SAVE AND SET MODE instruction 7-16
  for BRANCH AND SAVE instruction 7-15
linkage for subroutines 5-10
linkage index (LX) 5-27
linkage stack 5-60, 5-68
  associated PER events 5-64
  associated trace entries 5-64
  branch state entry 10-10
  entry address 5-68
  entry descriptor 5-69
  entry-type code 5-69
  handling of information in 5-63
  header entry 5-70
  instructions 5-60
  introduction 5-66
  next-entry size 5-69
  operations 5-66
    control 5-68
  program-call state entry 10-54
  remaining free space 5-69
  section 5-66
    identification 5-69
  state entry 5-71
  trailer entry 5-70
linkage-stack functions 5-61
linkage table (LT) 5-28
  designation (LTD) 5-27
    in AST entry 3-21
  length (LTL) 5-28
    in primary AST entry 5-28
  origin (LTO) 5-28
    in primary AST entry 5-28
little endian 7-81
LM (LOAD MULTIPLE) instruction 7-80
LNDBR (LOAD NEGATIVE) BFP instruction 19-38
LNDR (LOAD NEGATIVE) HFP instruction 18-16
LNEBR (LOAD NEGATIVE) BFP instruction 19-38
LNER (LOAD NEGATIVE) HFP instruction 18-16
LNR (LOAD NEGATIVE) binary instruction 7-80
LNXBR (LOAD NEGATIVE) BFP instruction 19-38
LNXR (LOAD NEGATIVE) HFP instruction 18-16
LOAD ACCESS MULTIPLE instruction 7-77
LOAD ADDRESS EXTENDED instruction 7-78
LOAD ADDRESS instruction 7-78
  examples A-22
LOAD ADDRESS RELATIVE LONG instruction 7-79
LOAD ADDRESS SPACE PARAMETERS instruction 10-28




LzLabs GmbH. Ex. 1008-1004

LOAD AND TEST BFP instructions 19-35
LOAD AND TEST binary instruction 7-79
LOAD AND TEST HFP instructions 18-14
LOAD binary instructions 7-77
example A-22
load-clear key 12-3
LOAD COMPLEMENT BFP instructions 19-35
LOAD COMPLEMENT binary instruction 7-79
LOAD COMPLEMENT HFP instructions 18-15
LOAD CONTROL instruction 10-36
LOAD floating-point instructions 9-10
LOAD FP INTEGER BFP instructions 19-36
LOAD FP INTEGER HFP instructions 18-15
LOAD FPC instruction 19-37
LOAD HALFWORD IMMEDIATE instruction 7-80
LOAD HALFWORD instruction 7-80
examples A-23
load indicator 12-3
LOAD LENGTHENED BFP instructions 19-38
LOAD LENGTHENED HFP instructions 18-16
LOAD MULTIPLE instruction 7-80
LOAD NEGATIVE BFP instructions 19-38
LOAD NEGATIVE binary instruction 7-80
LOAD NEGATIVE HFP instructions 18-16
load-normal key 12-3
LOAD POSITIVE BFP instructions 19-39
LOAD POSITIVE binary instruction 7-81
LOAD POSITIVE HFP instructions 18-17
LOAD PSW instruction 10-37
LOAD REAL ADDRESS instruction 10-38
LOAD REVERSED instructions 7-81
LOAD ROUNDED BFP instructions 19-39
LOAD ROUNDED HFP instructions 18-18
load state 4-1, 4-2
during IPL 4-43
load-unit-address controls 12-3
LOAD USING REAL ADDRESS instruction 10-40
LOAD ZERO floating-point instructions 9-11
loading, initial
*See* IML, IPL
location 3-2
*See also* address
not available in configuration 6-16
lock A-46
example with FIFO queuing A-48
example with LIFO queuing A-47
lock used by PERFORM LOCKED OPERATION instruction 7-111
logical
arithmetic (unsigned binary) 7-4
comparison 7-4
connective
AND 7-14
EXCLUSIVE OR 7-74
OR 7-100
data 7-2
logical address 3-4
handling by DAT 3-28
logical-path mask
*See* LPM
I/O-interruption
*See* I/O-interruption subclass mask
logical string assist 1-2
logically partitioned (LPAR) mode 1-10, 1-12
logout
fixed
assigned storage locations for 3-48
machine-check 11-27
subchannel (I/O) 16-32
long binary-floating-point number 19-4
long hexadecimal-floating-point number 18-3
long I/O block 16-23
loop control 5-10
loop of interruptions
*See* string of interruptions
low-address protection 3-12
control bit 3-12
exception for 6-28
LPAR (logically partitioned) mode 1-10, 1-12
LPDBR (LOAD POSITIVE) BFP instruction 19-39
LPDR (LOAD POSITIVE) HFP instruction 18-17
LPEBR (LOAD POSITIVE) BFP instruction 19-39
LPER (LOAD POSITIVE) HFP instruction 18-17
LPM (logical-path mask) 15-4, 15-25
effect on system performance of 15-11
used for IPL 17-18
LPR (LOAD POSITIVE) binary instruction 7-81
LPSW (LOAD PSW) instruction 10-37
LPUM (last-path-used mask) 15-5
field-validity flag for (in subchannel logout) 16-34
in ESW 16-34
LPXBR (LOAD POSITIVE) BFP instruction 19-39
LPXR (LOAD POSITIVE) HFP instruction 18-17
LR (LOAD) binary instruction 7-77
LRA (LOAD REAL ADDRESS) instruction 10-38
LRDR (LOAD ROUNDED) HFP instruction 18-18
LRER (LOAD ROUNDED) HFP instruction 18-18
LRV (LOAD REVERSED) instruction 7-81
LRVH (LOAD REVERSED) instruction 7-81
LRVR (LOAD REVERSED) instruction 7-81
LT (linkage table) 5-28
LTD (linkage-table designation) 5-27
LTDBR (LOAD AND TEST) BFP instruction 19-35
LTDR (LOAD AND TEST) HFP instruction 18-14
LTEBR (LOAD AND TEST) BFP instruction 19-35
LTER (LOAD AND TEST) HFP instruction 18-14
LTL (linkage-table length) 5-28
in primary AST entry 5-28
LTO (linkage-table origin) 5-28
in primary AST entry 5-28
LTR (LOAD AND TEST) binary instruction 7-79


LzLabs GmbH. Ex. 1008-1005

LTXBR (LOAD AND TEST) BFP instruction 19-35
LTXR (LOAD AND TEST) HFP instruction 18-14
LURA (LOAD USING REAL ADDRESS)
instruction 10-40
LX (linkage index) 5-27
invalid bit 5-28
translation exception 6-24
LXD (LOAD LENGTHENED) HFP instruction 18-16
LXDB (LOAD LENGTHENED) BFP instruction 19-38
LXDBR (LOAD LENGTHENED) BFP instruction 19-38
LXDR (LOAD LENGTHENED) HFP instruction 18-16
LXE (LOAD LENGTHENED) HFP instruction 18-16
LXEB (LOAD LENGTHENED) BFP instruction 19-38
LXEBR (LOAD LENGTHENED) BFP instruction 19-38
LXER (LOAD LENGTHENED) HFP instruction 18-16
LXR (LOAD) floating-point instruction 9-10
LZDR (LOAD ZERO) floating-point instruction 9-11
LZER (LOAD ZERO) floating-point instruction 9-11
LZXR (LOAD ZERO) floating-point instruction 9-11

# M

M (mega) xxi
M (MULTIPLY) binary instruction 7-98
example A-27
machine check 11-1
*See also* malfunction
comparison of 370-XA with System/370 F-7
extended save area 11-24
handling of malfunction detected as part of I/O 11-5
interruption 6-14, 11-11
action 11-12
code (MCIC) 3-48, 11-15
floating conditions 11-26
machine check interruption 11-26
mask in PSW 4-5
subclass masks in control register 11-26
logout 11-27
mask
in PSW 4-5
machine-check architectural-mode identification 3-47
machine-type number (in CPU ID) 10-90
MAD (MULTIPLY AND ADD) HFP instruction 18-20
MADB (MULTIPLY AND ADD) BFP instruction 19-42
MADBR (MULTIPLY AND ADD) BFP instruction 19-42
MADR (MULTIPLY AND ADD) HFP instruction 18-20
MAE (MULTIPLY AND ADD) HFP instruction 18-20
MAEB (MULTIPLY AND ADD) BFP instruction 19-42
MAEBR (MULTIPLY AND ADD) BFP instruction 19-42
MAER (MULTIPLY AND ADD) HFP instruction 18-20
main storage 3-1
*See also* storage
effect of power-on reset on 4-42
shared (in multiprocessing) 4-44
malfunction 11-1
at channel subsystem 16-27
malfunction *(continued)*
at I/O device 16-28
correction of 11-2
effect on manual operation 12-1
from DIAGNOSE instruction 10-19
indication of 11-5
machine-check handling for when detected as part of I/O 11-5
malfunction alert (external interruption) 6-12
when entering check-stop state 11-11
manual indicator 12-3
*See also* stopped state
manual operation 12-1
controls
address-compare 12-1
alter-and-display 12-2
IML 12-3
load-unit-address 12-3
power 12-3
rate 12-4
TOD-clock 12-5
effect on CPU signaling 4-50
keys
interrupt 12-3
load-clear 12-3
load-normal 12-3
restart 12-4
start 12-4
stop 12-4
store-status 12-4
system-reset-clear 12-5
system-reset-normal 12-5
masks 6-6
*See also* I/O interruption, interruption
for BFP arithmetic exceptions 19-3
for IEEE exception conditions 19-3
in BRANCH ON CONDITION instruction 7-17
in BRANCH RELATIVE ON CONDITION instruction 7-20
in COMPARE LOGICAL CHARACTERS UNDER MASK instruction 7-43
in INSERT CHARACTERS UNDER MASK instruction 7-76
in PSW 4-5
in STORE CHARACTERS UNDER MASK instruction 7-122
in TEST UNDER MASK HIGH instruction 7-130
in TEST UNDER MASK instruction 7-130
in TEST UNDER MASK LOW instruction 7-130
monitor 6-24
path-management 15-2, 15-25
PER-event 4-15
PER general-register 4-16
program-interruption 6-14
subclass
*See* subclass-mask bits



LzLabs GmbH. Ex. 1008-1006

maximum negative number 7-3
MBA (measurement-block address) 15-8
MC (MONITOR CALL) instruction 7-82
MCIC (machine-check-interruption code) 3-48, 11-15
MD (MULTIPLY) HFP instruction 18-18
MDB (MULTIPLY) BFP instruction 19-40
MDBR (MULTIPLY) BFP instruction 19-40
MDE (MULTIPLY) HFP instruction 18-18
MDEB (MULTIPLY) BFP instruction 19-40
MDEBR (MULTIPLY) BFP instruction 19-40
MDER (MULTIPLY) HFP instruction 18-18
MDR (MULTIPLY) HFP instruction 18-18
  example A-41
ME (MULTIPLY) HFP instruction 18-18
measurement
  block (I/O)
    address 17-8
    format 17-7
    origin 17-7
  device-connect-time 17-10
  extended-measurement-word 17-11
  measurement-block update (I/O) 17-3
measurement block (I/O) 17-3
  data check 16-33
  index 15-6
  key (MBK)
    used as access key 3-9
  multiple use of 15-11
  program check 16-33
  protection check 16-33
  update enable 15-3
measurement-block-format control (I/O) 15-7
measurement data (I/O)
  accumulated 17-3
  effect of CSCH on 14-5
  effect of HSCH on 14-6
measurement-mode control (I/O) 15-3
MEE (MULTIPLY) HFP instruction 18-18
MEEB (MULTIPLY) BFP instruction 19-40
MEEBR (MULTIPLY) BFP instruction 19-40
MEER (MULTIPLY) HFP instruction 18-18
MER (MULTIPLY) HFP instruction 18-18
message byte (in EDIT) 8-8
MH (MULTIPLY HALFWORD) instruction 7-98
  example A-27
MHI (MULTIPLY HALFWORD IMMEDIATE) instruction 7-98
ML (MULTIPLY LOGICAL) instruction 7-99
MLR (MULTIPLY LOGICAL) instruction 7-99
mode
  access-register 3-28
  addressing
    *See* addressing mode
  architectural
    *See* architectural mode
  burst (channel-path operation) 13-3
mode *(continued)*
  byte-multiplex (channel-path operation) 13-3
  home-space 3-28
  incorrect-length-indication 15-25
  incorrect-length-suppression 15-25
  indicator
    architectural 12-2
  multipath
    *See* multipath mode
  primary-space 3-28
  real 3-28
  requirements for semiprivileged instructions 5-21
  rounding 19-7
  secondary-space 3-28
  single-path 15-3, 15-21
  translation 3-28
model number (in CPU ID) 10-90
modifiable area (in linkage-stack state entry) 5-73
modification control 15-23
MODIFY STACKED STATE instruction 10-40
MODIFY SUBCHANNEL instruction 14-7
MONITOR CALL instruction 7-82
monitor-class number 6-25
  assigned storage locations for 3-46
monitor code 6-25
  assigned storage locations for 3-46
monitor event 6-24
monitor masks 6-24
monitoring
  *See also* measurement
  channel-subsystem 17-1
  for PER events
    *See* PER
  with MONITOR CALL 6-24, 7-82
MOVE instructions 7-83
  examples A-21, A-23
move-inverse facility 7-83
MOVE INVERSE instruction 7-83
  example A-24
MOVE LONG EXTENDED instruction 7-87
MOVE LONG instruction 7-83
  examples A-25
MOVE LONG UNICODE instruction 7-90
MOVE NUMERICS instruction 7-93
  example A-25
move-page facility 2 D-3
MOVE PAGE instruction 7-93, 10-42
  facility 1 7-93
  facility 2 10-42
MOVE STRING instruction 7-95
  example A-26
MOVE TO PRIMARY instruction 10-45
MOVE TO SECONDARY instruction 10-45
MOVE WITH DESTINATION KEY instruction 10-47
MOVE WITH KEY instruction 10-47


LzLabs GmbH. Ex. 1008-1007

MOVE WITH OFFSET instruction 7-97
example A-26
MOVE WITH SOURCE KEY instruction 10-48
MOVE ZONES instruction 7-97
example A-27
MP (MULTIPLY DECIMAL) instruction 8-12
example A-36
MR (MULTIPLY) binary instruction 7-98
example A-27
MS (MULTIPLY SINGLE) instruction 7-99
MSCH (MODIFY SUBCHANNEL) instruction 14-7
MSD (MULTIPLY AND SUBTRACT) HFP instruction 18-20
MSDB (MULTIPLY AND SUBTRACT) BFP instruction 19-42
MSDBR (MULTIPLY AND SUBTRACT) BFP instruction 19-42
MSDR (MULTIPLY AND SUBTRACT) HFP instruction 18-20
MSE (MULTIPLY AND SUBTRACT) HFP instruction 18-20
MSEB (MULTIPLY AND SUBTRACT) BFP instruction 19-42
MSEBR (MULTIPLY AND SUBTRACT) BFP instruction 19-42
MSER (MULTIPLY AND SUBTRACT) HFP instruction 18-20
MSR (MULTIPLY SINGLE) instruction 7-99
MSTA (MODIFY STACKED STATE) instruction 10-40
multipath mode 15-3
entering 15-21
multiple-access storage references 5-87
MULTIPLY AND ADD BFP instructions 19-42
MULTIPLY AND ADD HFP instructions 18-20
MULTIPLY AND SUBTRACT BFP instructions 19-42
MULTIPLY AND SUBTRACT HFP instructions 18-20
MULTIPLY BFP instructions 19-40
MULTIPLY binary instructions 7-98
examples A-27
MULTIPLY DECIMAL instruction 8-12
example A-36
MULTIPLY HALFWORD IMMEDIATE instruction 7-98
MULTIPLY HALFWORD instruction 7-98
example A-27
MULTIPLY HFP instructions 18-18
example A-41
MULTIPLY LOGICAL instructions 7-99
MULTIPLY SINGLE instructions 7-99
multiprocessing 4-44
manual operations for 12-6
programming considerations for 8-3, A-43
programming examples A-43
timing-facility interruptions for 4-34
TOD clock for 4-29
multiprogramming examples A-43
MVC (MOVE) instruction 7-83
examples A-21, A-23
MVCDK (MOVE WITH DESTINATION KEY) instruction 10-47
MVCIN (MOVE INVERSE) instruction 7-83
example A-24
MVCK (MOVE WITH KEY) instruction 10-47
MVCL (MOVE LONG) instruction 7-83
examples A-25
MVCLE (MOVE LONG EXTENDED) instruction 7-87
MVCLU (MOVE LONG UNICODE) instruction 7-90
MVCP (MOVE TO PRIMARY) instruction 10-45
MVCS (MOVE TO SECONDARY) instruction 10-45
MVCSK (MOVE WITH SOURCE KEY) instruction 10-48
MVI (MOVE) instruction 7-83
example A-24
MVN (MOVE NUMERICS) instruction 7-93
example A-25
MVO (MOVE WITH OFFSET) instruction 7-97
example A-26
MVPG (MOVE PAGE) instruction 7-93, 10-42
facility 1 7-93
facility 2 10-42
MVST (MOVE STRING) instruction 7-95
example A-26
MVZ (MOVE ZONES) instruction 7-97
example A-27
MXBR (MULTIPLY) BFP instruction 19-40
MXD (MULTIPLY) HFP instruction 18-18
MXDB (MULTIPLY) BFP instruction 19-40
MXDBR (MULTIPLY) BFP instruction 19-40
MXDR (MULTIPLY) HFP instruction 18-18
MXR (MULTIPLY) HFP instruction 18-18

## N

N (AND) instruction 7-13
N condition (I/O) 16-12
NaN (not-a-number) 19-6
NC (AND) instruction 7-13
near-valid CBC 11-2
in storage 11-5
negative zero
binary 7-3
decimal 8-3
example A-5
new PSW 4-3
assigned storage locations for 3-43
fetched during interruption 6-2
next-entry size (in linkage stack) 5-69
NI (AND) instruction 7-13
example A-8
no-operation
instruction (BRANCH ON CONDITION) 7-17
instruction (BRANCH RELATIVE ON CONDITION) 7-20



LzLabs GmbH. Ex. 1008-1008

node (of tree structure) 7-141
noninterlocked-update storage reference 5-86
nonnumeric entities
  binary 19-6
nonvolatile storage 3-2
normalization
  of BFP numbers 19-8
  of HFP numbers 18-3
not-a-number (NaN) 19-6
not operational
  as channel-path state 16-12
    *See also* path-not-operational bit
  as CPU state 4-50
  as TOD-clock state 4-30
not set (TOD-clock state) 4-30
NR (AND) instruction 7-13
nullification
  exceptions to 5-19
  for exigent machine-check conditions 11-11
  of instruction execution 5-17
  of unit of operation 5-18
numbering
  of addresses (byte locations) 3-2
  of bits 3-2
numbers
  binary 7-2
    examples A-2
  binary-floating-point 19-4
  CPU-model 10-90
  decimal 8-1
    examples A-4
  device 13-5
  hexadecimal 5-6
  hexadecimal-floating-point 18-3
  hexadecimal-floating-point
    examples A-5
  machine-type 10-90
numeric bits 8-1
  moving of 7-93

## O

O (OR) instruction 7-100
OC (OR) instruction 7-100
OEMI (original equipment manufacturers information) for I/O interface xxi
  publication referenced xxi
OI (OR) instruction 7-100
  example A-28
  example of problem with A-43
old PSW 6-2
  assigned storage locations for 3-43
one's complement binary notation 7-2
  used for SUBTRACT LOGICAL instruction 7-128
  used for SUBTRACT LOGICAL WITH BORROW instruction 7-128
op code
  *See* operation code
operand 5-3
  access identification 3-47
  access of 5-85
    for I/O instructions 14-1
  address generation for 5-8
  exception 6-25
  immediate 5-6
  length of 5-3
  overlap of
    for decimal instructions 8-3
    for general instructions 7-2
  register for 5-6
  sequence of references for 5-85
  storage 5-6
  types of (fetch, store, update) 5-85
  used for result 5-3
operating state 4-1, 4-2
operation
  I/O
    *See* I/O operations
  unit of 5-17
operation code (op code) 5-2
  invalid 6-25
operation exception 6-25
operation-request block
  *See* ORB
operator facilities 2-7, 12-1
  basic 12-1
operator intervening (signal-processor status) 4-52
OR instructions 7-100
  example of problem with OR immediate A-43
  examples A-28
ORB (operation-request block) 15-22
  CSS priority 15-26
  extension control in 15-26
  interruption parameter in 15-22
  invalid 16-26
  logical-path mask (LPM) in 15-25
orders (I/O) 13-6, 15-29
orders (signal-processor) 4-45
  conditions precluding response to 4-50
  CPU reset 4-46
  emergency signal 4-45
  external call 4-45
  initial CPU reset 4-46
  restart 4-46
  sense 4-45
  set architecture 4-49
  set prefix 4-46
  start 4-46
  stop 4-46
  stop and store status 4-46
  store extended status at address 4-47
  store status at address 4-47


LzLabs GmbH. Ex. 1008-1009

overflow
binary 7-3
example A-2
decimal 6-22
exponent
*See* exponent overflow
fixed-point 6-23, 7-4
in CRW 17-24
overlap
destructive 7-84, 7-88, 7-91
operand 5-80
for decimal instructions 8-3
for general instructions 7-2
operation 5-79

# P

PACK ASCII instruction 7-101
PACK instruction 7-100
example A-28
PACK UNICODE instruction 7-102
packed decimal numbers 8-1
conversion of to zoned format 7-139
conversion to from zoned format 7-100
examples A-4
padding byte
for COMPARE LOGICAL LONG EXTENDED instruction 7-45
for COMPARE LOGICAL LONG instruction 7-43
for MOVE LONG EXTENDED instruction 7-87
for MOVE LONG instruction 7-84
page 3-27
page-frame real address (PFRA) 3-31
PAGE IN instruction 10-50
page index (PX) 3-27
page-invalid bit (in page-table entry) 3-31
PAGE OUT instruction 10-51
page protection 3-11, F-2
bit for 3-31
exception for 6-28
page swapping 3-26
page table 3-30
designation 3-30
length (PTL) 3-30
lookup 3-35
origin (PTO) 3-30
page-translation exception 6-26
as an access exception 6-35, 6-40
PALB (PURGE ALB) instruction 10-76
PAM (path-available mask) 15-7
effect of reconfiguration on 15-11
effect of resetting on 15-11
effect on allegiance of 15-11
parallel-I/O channel-to-channel adapter
publication referenced xxi
parallel-I/O interface 13-3
OEMI publication referenced xxi
parameter
external-interruption 6-10
assigned storage locations for 3-44
I/O-interruption
*See* I/O-interruption parameter
register for SIGNAL PROCESSOR 4-46, 10-88
translation 3-27
parity bit 11-2
partial completion of instruction execution 5-17
PASN (primary address-space number) 3-17
in trace entry 4-13
PASTE (primary AST entry) 5-28
PASTEO (primary-AST-entry origin) 5-28, 5-43
path
*See* channel path
path available for selection 15-13
path management 13-6
for clear function 15-14
for halt function 15-15
for start function and resume function 15-18
path-management-control word
*See* PMCW
path-management masks
last-path-used mask
*See* LPUM
logical-path mask
*See* LPM
path-available mask
*See* PAM
path-installed mask
*See* PIM
path-not-operational mask
*See* PNOM
path-operational mask
*See* POM
path-not-operational bit (N) in SCSW 16-12
path-not-operational condition 15-4
path verification required
indicator for (in ERW) 16-37
pattern (in EDIT) 8-7
PC (PROGRAM CALL) instruction 10-52
PC-cp (PROGRAM CALL instruction, to current primary) 10-54
PC number 10-53
in linkage-stack state entry 5-73
in trace entry 4-13
translation 5-27
PC-ss (PROGRAM CALL instruction, with space switching) 10-55
PC-translation-specification exception 6-27
PC-type bit 5-65
PCF (PROGRAM CALL FAST) instruction 10-63
PCI (program-controlled interruption) 15-35
as flag in CCW 15-29



LzLabs GmbH. Ex. 1008-1010

PCI (program-controlled interruption) *(continued)*
intermediate interruption condition for 16-17
subchannel status for 16-23
pending channel reports (effect of I/O-system reset on) 17-16
pending interruption
*See* interruption pending
PER (program-event recording) 4-14
access identification 3-46, 4-18
address 4-18
assigned storage locations for 3-46
ATMID (addressing-and-translation-mode identification) 4-17
code 4-17
assigned storage locations for 3-46
events 4-14
extensions 1-9
general-register-alteration event 4-23
mask bits 4-16
instruction-fetching event 4-22
masks
bit in PSW 4-5
general-register 4-16
PER-event 4-15
priority of indication 4-20
program-interruption condition 6-27
STD (segment-table-designation) identification 4-18
storage-alteration event 4-22
storage-area designation 4-21
ending address 4-16
starting address 4-16
wraparound 4-21
store-using-real-address event 4-24
successful-branching event 4-22
PER 1 (program-event recording 1) 4-14
PER 2 (program-event recording 2) 4-14
PER 2 (program event recording 2) facility D-4
PERFORM LOCKED OPERATION instruction 7-103
example A-50
PFRA (page-frame real address) 3-31
PGIN (PAGE IN) instruction 10-50
PGOUT (PAGE OUT) instruction 10-51
piecemeal steps of instruction execution 5-79
PIM (path-installed mask) 15-6
PKA (PACK ASCII) instruction 7-101
PKM (PSW-key mask) 5-22
PKU (PACK UNICODE) instruction 7-102
PLO (PERFORM LOCKED OPERATION) instruction 7-103
example A-50
PMCW (path-management-control word) 15-2
channel-path identifiers (CHPID) in 15-7
PNOM (path-not-operational mask) 15-4
effect on POM of 15-11
indicated in SCSW 16-12
point of damage 11-14
point of interruption 5-17
for machine check 11-14
POM (path-operational mask) 15-6
effect on PNOM of 15-11
POST (SVC)
example of routine to bypass A-45
postnormalization 18-3
power controls 12-3
power-on reset 4-42
powers of 2
table of G-1
PR (PROGRAM RETURN) instruction 10-67
PR-cp (PROGRAM RETURN instruction, to current primary) 10-67
PR-ss (PROGRAM RETURN instruction, with space switching) 10-67
PR/SM (Processor Resource/Systems Manager) 1-10, 1-12
precision (floating-point) 9-1
preferred sign codes 8-2
prefetch control 15-24
prefetching
*See also* CCW prefetch control
access exceptions not recognized for 6-37
channel-control check during 16-27
channel-data check during 16-27
handling of invalid CBC in storage keys during 11-8
of ART-table and DAT-table entries 5-83
of data for I/O 15-30
of instructions 5-82
of operands 5-85
prefix 3-15
set by signal-processor order 4-46
store-status save area for 3-48
prefix area 3-15
prenormalization 18-3
primary address space 3-16
primary ASN (PASN) 3-17
in linkage-stack state entry 5-72
primary AST entry (PASTE)
origin (PASTEO) 5-28, 5-43
primary authority 3-24
exception 6-27
primary interruption condition (I/O) 16-4
primary-list bit 5-43
primary segment table
designation (PSTD) 3-28
length (PSTL) 3-29
origin (PSTO) 3-28
primary-space access-list designation (PSALD) 5-45
primary-space mode 3-28
primary space-switch-event-control bit 3-28
primary-status bit (I/O) 16-18
primary storage-alteration-event-control bit 3-29


LzLabs GmbH. Ex. 1008-1011

primary virtual address 3-4
effective segment-table designation for 3-32
priority
of access exceptions 6-40
of ASN-translation exceptions 6-45
of data exceptions 6-15
of external-interruption conditions 6-10
of I/O interruptions 16-4
of interruptions (CPU) 6-46
of PER events 4-20
of program-interruption conditions 6-38
for arithmetic exceptions 6-15
of subspace-replacement exceptions 6-45
of trace exceptions 6-45
private bit 5-46
private-space control
effect on
fetch-protection override 3-11
low-address protection 3-12
use of common segments 3-30
private-space-control bit 3-29
home 3-30
primary 3-29
secondary 3-29
privileged instructions 4-6
control 10-1
I/O 14-1
privileged-operation exception 6-27
problem state 4-6
bit in entry-table entry 5-29
bit in PSW 4-6
compatibility 1-13
processing backup (synchronous machine-check condition) 11-20
processing damage (synchronous machine-check condition) 11-20
processor
*See* CPU
processor-availability facility 1-9
Processor Resource/Systems Manager (PR/SM) 1-10, 1-12
program 5-36
channel
*See* channel program
exceptions 6-14
execution of 5-2
fields of SCHIB modifiable by 15-9
initial loading of 4-43, 17-17
interruption 6-14
priority of 6-15, 6-38
mask (in PSW) 4-6
program-call-fast-control bit 5-23
PROGRAM CALL FAST instruction 10-63
PROGRAM CALL instruction 10-52
trace entry for 4-13
type of 5-65
program-call state entry 5-71, 10-54
program check
as subchannel status 16-24
measurement-block 16-33
program-controlled interruption (I/O)
*See* PCI
program-event recording
*See* PER
program-event-recording facility 2 D-4
program events
*See* PER events
program mask
validity bit for 11-22
PROGRAM RETURN instruction 10-67
program-status word
*See* PSW
PROGRAM TRANSFER instruction 10-70
trace entry for 4-13
programmable field of TOD clock 4-32
protection (storage) 3-8
access-list-controlled
*See* access-list-controlled protection
during tracing 4-14
fetch
*See* fetch protection
key-controlled
*See* key-controlled protection
low-address
*See* low-address protection
page
*See* page protection
protection check
as subchannel status 16-26
measurement-block 16-33
protection exception 6-28
as an access exception 6-35, 6-40
PSALD (primary-space access-list designation) 5-45
pseudo AST entry 3-17
PSTD (primary segment-table designation) 3-28
PSTL (primary segment-table length) 3-29
PSTO (primary segment-table origin) 3-28
PSW (program-status word) 2-3, 4-3
assigned storage locations for 3-43
comparison of 370-XA with System/370 F-5
comparison of ESA/370 with 370-XA E-3
current 4-3, 5-9
stored during interruption 6-2
exceptions associated with 6-9
format error 6-9
in linkage-stack state entry 5-72
in program execution 5-9
saved 4-41
store-status save area for 3-48
validity bits for 11-22
PSW key 4-5
control bit 5-65



LzLabs GmbH. Ex. 1008-1012

PSW key *(continued)*
in entry-table entry 5-66
in trace entry 4-13
used as access key 3-9
validity bit for 11-22
PSW-key mask (PKM) 5-22
control bit 5-65
in linkage-stack state entry 5-72
PT (PROGRAM TRANSFER) instruction 10-70
PT-cp (PROGRAM TRANSFER instruction, to current primary) 10-71
PT-ss (PROGRAM TRANSFER instruction, with space switching) 10-71
PTL (page-table length) 3-30
PTLB (PURGE TLB) instruction 10-76
PTO (page-table origin) 3-30
publications
other related documents xxi
PURGE ALB instruction 10-76
PURGE TLB instruction 10-76
PX (page index) 3-27

# Q

QNaN (quiet NaN) 19-6
queuing
FIFO
example for lock and unlock A-47
LIFO
example for lock and unlock A-46
quiet NaN (QNaN) 19-6

# R

R field of instruction 5-6
radix
binary 9-1
hexadecimal 9-1
rate control 12-4
RCHP (RESET CHANNEL PATH) instruction 14-9
real address 3-4
real mode 3-28
real storage 3-4
receiver check (signal-processor status) 4-53
reconfiguration of I/O system 17-20
recovery
as class of machine-check condition 11-12
channel-subsystem 17-21
system 11-17
subclass-mask bit for 11-26
reduced-authority state 10-7
redundancy 11-2
reference
bit in storage key 3-8
multiple-access 5-87
recording 3-14
reference *(continued)*
sequence for storage 5-78
*See also* sequence
single-access 5-87
to expanded storage by MOVE PAGE 7-93
register
access 2-4
base-address 2-3
control 2-4
designation of 5-6
floating-point 2-3, 9-2
floating-point-control 19-2
general 2-3
index 2-3
prefix 3-15
save areas for 3-48, 11-24
validation of 11-10
vector-facility 2-6
relative branching 5-9
remainder 19-9
result of DIVIDE TO INTEGER 19-30
remaining free space (in linkage stack) 5-69
remote operating stations 12-1
reporting-source code (RSC) 17-24
reporting-source ID (RSID) 17-25
repressible machine-check conditions 11-12
reset 4-37, 17-13
channel-path 17-13
clear 4-42
CPU 4-40
effect on CPU state 4-2
effect on TOD clock 4-29
I/O-system 17-13
as part of subsystem reset 4-41
initial CPU 4-41
power on 4-42
subsystem 4-41
summary of functions 4-39
summary of functions performed by manual initiation of 4-38
system-reset-clear key 12-5
system-reset-normal key 12-5
RESET CHANNEL PATH instruction 14-9
*See also* channel-path-reset function
function initiated by 15-45
RESET REFERENCE BIT EXTENDED instruction 10-76
reset signal (I/O) 17-13
in channel-path reset 17-13
in I/O-system reset 17-14, 17-15
issued as part of RCHP 15-45
resetting event
*See* path verification required
resolution
of clock comparator 4-35
of CPU timer 4-35


LzLabs GmbH. Ex. 1008-1013

resolution *(continued)*
of TOD clock 4-29
restart
interruption 6-46
key 12-4
signal-processor order 4-46
result operand 5-3
resume function 13-8, 15-18
*See also* start function
initiated by RESUME SUBCHANNEL 14-10
path management for 15-19
pending 16-13
RESUME PROGRAM instruction 10-77
RESUME SUBCHANNEL instruction 14-10
*See also* resume function
channel-program requirements for 14-11
count of in measurement block 17-4
function initiated by 15-18
retry
CPU 11-2
I/O command
*See* command retry
RI instruction format 5-5
RIL instruction format 5-5
RLL (ROTATE LEFT SINGLE LOGICAL) instruction 7-116
ROTATE LEFT SINGLE LOGICAL instruction 7-116
rounding (decimal) 8-13
example A-38
rounding (floating-point)
of BFP result 19-7
of HFP result 18-18
rounding action
summary of 9-3
RP (RESUME PROGRAM) instruction 10-77
RR instruction format 5-5
RRBE (RESET REFERENCE BIT EXTENDED) instruction 10-76
RRE instruction format 5-5
RRF instruction format 5-5
RS instruction format 5-5
RSC (reporting-source code) 17-24
RSCH (RESUME SUBCHANNEL) instruction 14-10
RSE instruction format 5-5
RSI instruction format 5-5
RSID (reporting-source ID) 17-25
RSL instruction format 5-5
running (state of TOD clock) 4-30
RX instruction format 5-5
RXE instruction format 5-5
RXF instruction format 5-5

# S

S (SUBTRACT) binary instruction 7-127
S instruction format 5-5
SAC (SET ADDRESS SPACE CONTROL) instruction 10-79
SACF (SET ADDRESS SPACE CONTROL FAST) instruction 10-79
SAL (SET ADDRESS LIMIT) instruction 14-11
SAM24 (SET ADDRESSING MODE) instruction 7-117
SAM31 (SET ADDRESSING MODE) instruction 7-117
sample count (in ESW) 17-4
SAR (SET ACCESS) instruction 7-117
SASN (secondary address-space number) 3-17
in trace entry 4-13
save areas for registers 3-48, 4-44, 11-24
saved PSW 4-41
SCHIB (subchannel-information block) 15-1
as operand of
MODIFY SUBCHANNEL 14-7
STORE SUBCHANNEL 14-17
Measurement-Block Address (MBA) in 15-8
model-dependent area in 15-8
path-management-control word (PMCW) in 15-2
subchannel-status word (SCSW) in 15-8
summary of modifiable fields in 15-9
SCHM (SET CHANNEL MONITOR) instruction 14-12
SCK (SET CLOCK) instruction 10-81
SCKC (SET CLOCK COMPARATOR) instruction 10-82
SCKPF (SET CLOCK PROGRAMMABLE FIELD) instruction 10-82
SCP-initiated reset 1-9
SCSW (subchannel-status word) 16-6
activity-control field in 16-13
CCW address in 16-18
count in 16-29
device-status field in 16-23
function-control field in 16-12
in IRB 16-6
in SCHIB 15-8
status-control field in 16-16
subchannel-control field in 16-11
subchannel-status field in 16-23
SD (SUBTRACT NORMALIZED) HFP instruction 18-23
SDB (SUBTRACT) BFP instruction 19-45
SDBR (SUBTRACT) BFP instruction 19-45
SDR (SUBTRACT NORMALIZED) HFP instruction 18-23
SE (SUBTRACT NORMALIZED) HFP instruction 18-23
SEARCH STRING instruction 7-116
examples A-29
SEB (SUBTRACT) BFP instruction 19-45
SEBR (SUBTRACT) BFP instruction 19-45
secondary address space 3-16
secondary ASN (SASN) 3-17
control bit 5-66



LzLabs GmbH. Ex. 1008-1014

secondary ASN (SASN) *(continued)*
in linkage-stack state entry 5-72
secondary authority 3-24
exception 6-29
secondary-CCW address validity (in ERW) 16-37
secondary error (in subchannel logout) 16-35
secondary interruption condition (I/O) 16-4
secondary segment table
designation (SSTD) 3-29
length (SSTL) 3-29
origin (SSTO) 3-29
secondary-space-control bit 3-28, 5-22
secondary-space mode 3-28
secondary-status bit (I/O) 16-18
secondary storage-alteration-event-control bit 3-29
secondary virtual address 3-4
effective segment-table designation for 3-32
segment 3-27
segment index (SX) 3-27
segment-invalid bit (in segment-table entry) 3-30
segment table 3-30
length (STL) 3-28
lookup 3-34
origin (STO) 3-28
segment-table designation (STD) 3-28
effective 3-32
home 3-29
obtaining of in access-register translation 5-35
primary 3-28
secondary 3-29
use after ART 5-48
segment-translation exception 6-29
as an access exception 6-35, 6-40
self-describing block of I/O data 15-34
semiprivileged
instructions 4-6
descriptions of 10-1
program authorization 5-21
summary of 5-25
programs 4-6, 5-21
sense
as signal-processor order 4-45
sequence
conceptual 5-78
instruction-execution 5-2
of CCWs that is invalid 16-26
of storage references 5-78
ART-table and DAT-table entries 5-83
for floating-point data 9-2
instructions 5-82
operands 5-85
storage keys 5-84
sequence code (in subchannel logout) 16-35
field-validity flag for 16-34
SER (SUBTRACT NORMALIZED) HFP instruction 18-23
serial-I/O channel-to-channel adapter
publication referenced xxi
serial-I/O interface 13-2
publication referenced xxi
serialization 5-91
caused by I/O instructions 14-1
channel-program 5-92
CPU 5-91
in completion of store operations 5-86
service-call-logical-processor (SCLP) facility 1-11
service-processor damage 11-19
service processor inoperative (signal-processor status) 4-53
service-signal external interruption 6-12
subclass-mask bit for 6-13
SET ACCESS instruction 7-117
SET ADDRESS LIMIT instruction 14-11
SET ADDRESS SPACE CONTROL FAST instruction 10-79
SET ADDRESS SPACE CONTROL instruction 10-79
SET ADDRESSING MODE instructions 7-117
set architecture
signal-processor order 4-49
SET CHANNEL MONITOR instruction 14-12
effect on measurement modes of 17-1
SET CLOCK COMPARATOR instruction 10-82
SET CLOCK instruction 10-81
SET CLOCK PROGRAMMABLE FIELD instruction 10-82
SET CPU TIMER instruction 10-82
SET FPC instruction 19-44
set prefix (signal-processor order) 4-46
SET PREFIX instruction 10-83
SET PROGRAM MASK instruction 7-118
SET PSW KEY FROM ADDRESS instruction 10-83
SET ROUNDING MODE (SRNM) 19-44
SET SECONDARY ASN instruction 10-84
access registers 5-40
set state (of TOD clock) 4-30
SET STORAGE KEY EXTENDED instruction 10-87
SET SYSTEM MASK instruction 10-87
SFPC (SET FPC) instruction 19-44
SH (SUBTRACT HALFWORD) instruction 7-127
shared storage
*See* storage sharing
shared TOD clock 4-29
SHIFT AND ROUND DECIMAL instruction 8-13
examples A-37
SHIFT LEFT DOUBLE instruction 7-118
example A-29
SHIFT LEFT DOUBLE LOGICAL instruction 7-119
SHIFT LEFT SINGLE instruction 7-119
example A-30
SHIFT LEFT SINGLE LOGICAL instruction 7-120
SHIFT RIGHT DOUBLE instruction 7-120


LzLabs GmbH. Ex. 1008-1015

SHIFT RIGHT DOUBLE LOGICAL instruction 7-121
SHIFT RIGHT SINGLE instruction 7-121
SHIFT RIGHT SINGLE LOGICAL instruction 7-121
shifting
  floating-point
    *See* normalization
short binary-floating-point number 19-4
short hexadecimal-floating-point number 18-3
short I/O block 16-23
SI instruction format 5-5
SID
  *See* subsystem-identification word
sign bit
  binary 7-3
  floating-point 18-1
sign codes (decimal) 8-2
signal (I/O) 17-12
  clear
    *See* clear signal
  halt
    *See* halt signal
  reset
    *See* reset signal
SIGNAL PROCESSOR instruction 10-88
  comparison of 370-XA with System/370 F-6
  orders 4-45
  status 4-51
signaling NaN (SNaN) 19-6
signed binary
  arithmetic 7-3
  comparison 7-4
  integer 7-2
    examples A-2
significance
  loss 18-1
    in HFP addition 18-9
  mask (in PSW) 4-6
  starter (in EDIT) 8-8
significand 19-4
SIGP
  *See* SIGNAL PROCESSOR instruction
SIGP (SIGNAL PROCESSOR) instruction 10-88
single-access reference 5-87
single-path mode 15-3, 15-21
size notation xxi
size of address 3-5
  controlled by addressing mode 5-7
  in CCW 15-28
skip flag in CCW 15-29
  effect on data transfer of 15-35
SL (SUBTRACT LOGICAL) instruction 7-127
SLA (SHIFT LEFT SINGLE) instruction 7-119
  example A-30
SLB (SUBTRACT LOGICAL WITH BORROW) instruction 7-128
SLBR (SUBTRACT LOGICAL WITH BORROW) instruction 7-128
SLDA (SHIFT LEFT DOUBLE) instruction 7-118
  example A-29
SLDL (SHIFT LEFT DOUBLE LOGICAL) instruction 7-119
SLI (suppress-length-indication) flag in CCW 15-28
  for immediate operations 15-31
SLL (SHIFT LEFT SINGLE LOGICAL) instruction 7-120
SLR (SUBTRACT LOGICAL) instruction 7-127
SNaN (signaling NaN) 19-6
solicited interruption condition (I/O) 16-3
solid errors 11-5
sorting
  extended 1-9
sorting instructions
  *See also* COMPARE AND FORM CODEWORD instruction, UPDATE TREE instruction
  example A-51
source
  vector-facility (machine-check condition) 11-19
source of interruption
  identified by interruption code 6-5
SP (SUBTRACT DECIMAL) instruction 8-14
space-switch event 6-29
  control
    home, in control register 13 3-29
  control bit
    in ASTE 3-20
    primary, in control register 1 3-28
special-operation exception 6-30
special QNaN 19-6
specification exception 6-31
SPKA (SET PSW KEY FROM ADDRESS) instruction 10-83
SPM (SET PROGRAM MASK) instruction 7-118
SPT (SET CPU TIMER) instruction 10-82
SPX (SET PREFIX) instruction 10-83
SQD (SQUARE ROOT) HFP instruction 18-21
SQDB (SQUARE ROOT) BFP instruction 19-45
SQDBR (SQUARE ROOT) BFP instruction 19-45
SQDR (SQUARE ROOT) HFP instruction 18-21
SQE (SQUARE ROOT) HFP instruction 18-21
SQEB (SQUARE ROOT) BFP instruction 19-45
SQEBR (SQUARE ROOT) BFP instruction 19-45
SQER (SQUARE ROOT) HFP instruction 18-21
square root D-5
SQUARE ROOT BFP instructions 19-45
SQUARE ROOT HFP instructions 18-21
SQXBR (SQUARE ROOT) BFP instruction 19-45
SQXR (SQUARE ROOT) HFP instruction 18-21
SR (SUBTRACT) binary instruction 7-127
SRA (SHIFT RIGHT SINGLE) instruction 7-121
SRDA (SHIFT RIGHT DOUBLE) instruction 7-120



LzLabs GmbH. Ex. 1008-1016

SRDL (SHIFT RIGHT DOUBLE LOGICAL) instruction 7-121
SRL (SHIFT RIGHT SINGLE LOGICAL) instruction 7-121
SRNM (SET ROUNDING MODE) 19-44
SRP (SHIFT AND ROUND DECIMAL) instruction 8-13
  examples A-37
SRST (SEARCH STRING) instruction 7-116
  examples A-29
SS instruction format 5-5
SSAR (SET SECONDARY ASN) instruction 10-84
  access registers 5-40
SSAR-cp (SET SECONDARY ASN instruction, to current primary) 10-84
SSAR-ss (SET SECONDARY ASN instruction, with space switching) 10-84
SSASTEO (subspace-AST-entry origin) 5-56
SSASTESN (subspace-AST-entry sequence number) 5-57
SSCH (START SUBCHANNEL) instruction 14-14
SSE instruction format 5-5
SSKE (SET STORAGE KEY EXTENDED) instruction 10-87
SSM (SET SYSTEM MASK) instruction 10-87
SSM-suppression-control bit 6-30, 10-87
SSTD (secondary segment-table designation) 3-29
SSTL (secondary segment-table length) 3-29
SSTO (secondary segment-table origin) 3-29
ST (STORE) binary instruction 7-122
stack-empty exception 6-33
stack-full exception 6-33
stack-operation exception 6-33
stack-specification exception 6-33
stack-type exception 6-34
stacking process 5-73
stacking PROGRAM CALL 5-61
STAM (STORE ACCESS MULTIPLE) instruction 7-122
standalone dump 12-4
standard epoch (for TOD clock) 4-32
STAP (STORE CPU ADDRESS) instruction 10-90
start (CPU)
  function 4-2
  key 12-4
  signal-processor order 4-46
start function (I/O) 13-6, 15-18
  bit in SCSW for 16-12
  initiated by START SUBCHANNEL 14-14
  path management for 15-19
  pending 16-14
START SUBCHANNEL instruction 14-14
  *See also* start function for I/O
  count of in measurement block 17-4
  deferred condition code for (in SCSW) 16-8
  function initiated by 15-18
  operation-request block (ORB) used by 15-22
state
  CPU
    *See* CPU state
  TOD-clock 4-30
state entry 5-71
status
  alert 16-16
  device 16-23
    effect of clear function on 15-15
    field-validity flag for (in subchannel logout) 16-34
    with inappropriate bit combination 16-35
  device-status check 16-35
  for SIGNAL PROCESSOR 4-45, 10-88
  initial-status interruption
    *See* initial-status-interruption control
  intermediate 16-17
  primary 16-18
  program
    *See* PSW
  resulting from signal-processor orders 4-51
  secondary 16-18
  storing of 4-43
    manual key for 12-4
  subchannel 16-23
status-control field (in SCSW) 16-16
status modifier (device status)
  effect of in command chaining 15-35
status pending 16-18
status-verification facility 17-20, F-2
STC (STORE CHARACTER) instruction 7-122
STCK (STORE CLOCK) instruction 7-123
STCKC (STORE CLOCK COMPARATOR) instruction 10-89
STCKE (STORE CLOCK EXTENDED) instruction 7-124
STCM (STORE CHARACTERS UNDER MASK) instruction 7-122
  examples A-30
STCPS (STORE CHANNEL PATH STATUS) instruction 14-16
STCRW (STORE CHANNEL REPORT WORD) instruction 14-17
STCTL (STORE CONTROL) instruction 10-89
STD
  *See* segment-table designation
STD (STORE) floating-point instruction 9-11
STE (STORE) floating-point instruction 9-11
STFL (STORE FACILITY LIST) instruction 10-91
STFL facility list 3-47
STFPC (STORE FPC) instruction 19-45
STH (STORE HALFWORD) instruction 7-126
STIDP (STORE CPU ID) instruction 10-90
STL (segment-table length) 3-28
STM (STORE MULTIPLE) instruction 7-126
  example A-30



LzLabs GmbH. Ex. 1008-1017

STNSM (STORE THEN AND SYSTEM MASK) instruction 10-103
STO (segment-table origin) 3-28
stop
  function 4-2
  key 12-4
  signal-processor order 4-46
stop and store status (signal-processor order) 4-46
stopped (signal-processor status) 4-52
stopped state
  of CPU 4-1
    effect on completion of store operations 5-86
  of TOD clock 4-30
storage 3-1, 3-29
  absolute 3-4
  address wraparound
    *See* wraparound
  addressing 3-2
    *See also* address
  alteration
    space-control bit 4-15
  alteration manual controls 12-2
  alteration PER event 3-29, 4-22
    bits for 3-29
    mask for 4-15
  assigned locations in 3-43
    comparison of 370-XA with System/370 F-6
    comparison of ESA/370 with 370-XA E-3
  auxiliary 3-1, 3-26
  block 3-4
    testing for usability of 10-107
  buffer (cache) 3-2
  clearing of
    *See* clearing operation
  concurrency of access for references to 5-88
  configuration of 3-4
  direct-access 3-1
  display 12-2
  error 11-20
    indirect 11-21
  expanded 2-2
    accessed by MOVE PAGE 7-93
  failing address in
    *See* failing-storage address
  interlocked update 5-86
  interlocks for virtual references 5-79
  main 3-1
  noninterlocked update of 5-86
  nonvolatile 3-2
  operand 5-6
    reference to (fetch, store, update) 5-85
    update reference 5-86
  operand consistency 5-87
    examples A-47, A-49
  prefixing for 3-15
  real 3-4
storage *(continued)*
  sequence of references to 5-78
    for floating-point data 9-2
  size
    notation for xxi
  validation of 11-6
  virtual 3-26
  volatile 3-2
    effect of power-on reset on 4-42
storage-access code (in subchannel logout) 16-34
storage-alteration-event bit 4-16
storage-alteration-event-control bit 3-29
  home 3-30
  primary 3-29
  secondary 3-29
storage-area designation
  for I/O operations 15-30
  for PER events 4-21
storage degradation (machine-check condition) 11-21
storage key 3-8
  error in 11-21
  sequence of references to 5-84
  testing for usability of 10-107
  validation of 11-7
storage-key function 1-9
storage-logical-validity bit 11-23
storage protection 3-8
  during tracing 4-14
storage-protection override D-5
storage-protection-override-control bit 3-10
storage reconfiguration 1-9
storage sharing
  by address spaces 3-27
  by CPUs and the channel subsystem 3-4
  examples A-43
  in multiprocessing 4-44
STORE ACCESS MULTIPLE instruction 7-122
STORE binary instruction 7-122
STORE CHANNEL PATH STATUS instruction 14-16
STORE CHANNEL REPORT WORD instruction 14-17
  channel-report word (CRW) stored by 17-23
STORE CHARACTER instruction 7-122
STORE CHARACTERS UNDER MASK instruction 7-122
  examples A-30
STORE CLOCK COMPARATOR instruction 10-89
STORE CLOCK EXTENDED instruction 7-124
STORE CLOCK instruction 7-123
STORE CONTROL instruction 10-89
STORE CPU ADDRESS instruction 10-90
STORE CPU ID instruction 10-90
STORE CPU TIMER instruction 10-91
store extended status at address (signal-processor order) 4-47
STORE FACILITY LIST instruction 10-91



LzLabs GmbH. Ex. 1008-1018

STORE floating-point instructions 9-11
STORE FPC instruction 19-45
STORE HALFWORD instruction 7-126
STORE MULTIPLE instruction 7-126
example A-30
STORE PREFIX instruction 10-92
store reference 5-85
access exceptions for 6-38
STORE REVERSED instructions 7-126
store status 4-43
extended save area 4-44
key 12-4
signal-processor order for 4-46
store-status architectural-mode identification 3-47
store status at address (signal-processor order) 4-47
STORE SUBCHANNEL instruction 14-17
STORE SYSTEM INFORMATION instruction 10-92
STORE THEN AND SYSTEM MASK instruction 10-103
STORE THEN OR SYSTEM MASK instruction 10-103
store using real address (PER event) 4-24
store-using-real-address-event mask 4-15
STORE USING REAL ADDRESS instruction 10-104
STOSM (STORE THEN OR SYSTEM MASK) instruction 10-103
STPT (STORE CPU TIMER) instruction 10-91
STPX (STORE PREFIX) instruction 10-92
streaming-mode control 15-23
string of interruptions 4-3, 6-47
caused by clock comparator 4-35
caused by CPU timer 4-36
STRV (STORE REVERSED) instruction 7-126
STRVH (STORE REVERSED) instruction 7-126
STSCH (STORE SUBCHANNEL) instruction 14-17
STSI (STORE SYSTEM INFORMATION) instruction 10-92
STURA (STORE USING REAL ADDRESS) instruction 10-104
SU (SUBTRACT UNNORMALIZED) HFP instruction 18-23
subchannel 13-2
active allegiance for 15-12
dedicated allegiance for 15-12
effect of I/O-system reset on 17-15
idle 16-13
working allegiance for 15-12
subchannel-active bit 16-15
subchannel addressing 13-5
subchannel control information in SCSW 16-11
subchannel enabled bit in PMCW 15-2
subchannel-information block
*See* SCHIB
subchannel key 15-22, 16-8
used as access key 3-9
used for IPL 17-18
subchannel key check (in subchannel logout) 16-32
subchannel logout 16-32
subchannel number 13-5
subchannel status 16-23
subchannel-status word
*See* SCSW
subclass-mask bits
external-interruption 6-10
I/O-interruption
*See* I/O-interruption subclass mask
machine-check 11-26
subroutine linkage 5-10
subspace-active bit 5-56
subspace-AST-entry origin (SSASTEO) 5-56
subspace-AST-entry sequence number (SSASTESN) 5-57
subspace-group control 3-29
subspace-group-control bit
primary 3-29
secondary 3-29
subspace groups 5-55
introduction to 5-11
subspace-replacement
exceptions 6-45
operations 5-59
subsystem-identification word (SID)
subsystem-identification word (SID)
assigned storage locations for 3-47
subsystem-linkage-control bit 5-22, 5-27
in primary AST entry 5-28
subsystem reset 4-41
subsystem-identification word (SID) 14-1
SUBTRACT BFP instructions 19-45
SUBTRACT binary instructions 7-127
SUBTRACT DECIMAL instruction 8-14
SUBTRACT HALFWORD instruction 7-127
SUBTRACT LOGICAL instructions 7-127
SUBTRACT LOGICAL WITH BORROW instructions 7-128
SUBTRACT NORMALIZED
*See* SUBTRACT BFP instructions
SUBTRACT NORMALIZED HFP instructions 18-23
SUBTRACT UNNORMALIZED HFP instructions 18-23
successful-branching PER event 4-22
mask for 4-15
SUPERVISOR CALL instruction 7-129
supervisor-call interruption 6-46
supervisor state 4-6
support functions (I/O) 17-1
suppress-length-indication flag in CCW
*See* SLI
suppress-suspended-interruption control (I/O) 15-25, 16-11
used for IPL 17-18
suppression
exceptions to 5-19


LzLabs GmbH. Ex. 1008-1019

suppression *(continued)*
of instruction execution 5-16
of unit of operation 5-18
suppression on protection 3-12
virtual-address enhancement of 3-13
SUR (SUBTRACT UNNORMALIZED) HFP instruction 18-23
suspend control 15-23
suspend-control bit 16-8
used for IPL 17-18
suspend flag in CCW 15-29
invalid 16-25
suspend function 13-8
suspended bit (in SCSW) 16-16
suspension of channel-program execution 15-38
effect on DCTI of 15-40
intermediate interruption condition for 16-17
SVC (SUPERVISOR CALL) instruction 7-129
SW (SUBTRACT UNNORMALIZED) HFP instruction 18-23
swapping
by COMPARE (DOUBLE) AND SWAP instructions 7-40
by EXCLUSIVE OR instruction 7-74
SWR (SUBTRACT UNNORMALIZED) HFP instruction 18-23
SX (segment index) 3-27
SXBR (SUBTRACT) BFP instruction 19-45
SXR (SUBTRACT NORMALIZED) HFP instruction 18-23
synchronization
checkpoint 11-3
of CPU timer with TOD clock 4-36
of TOD clocks 4-30, 4-34
synchronization control 15-24
synchronous machine-check-interruption conditions 11-20
system
manual control of 12-1
organization of 2-1
system check stop 11-11
system damage 11-16
system mask (in PSW) 4-3
validity bit for 11-22
system recovery 11-17
system reset
*See* reset
I/O
*See* I/O-system reset
system-reset-clear key 12-5
system-reset-normal key 12-5
System/360 and System/370 I/O interface
*See* parallel-I/O interface
System/370
comparison with 370-XA F-1
compatibility with ESA/390 1-13
System/370 *(continued)*
effect of PER 2 on 4-15

# T

T (tera) xxi
table of powers of 2 G-1
tables
ASN
*See* ASN first table, ASN second table
authority
*See* authority table
DAT
*See* page table, segment table
entry
*See* entry table
linkage
*See* linkage table
page
*See* page table
segment
*See* segment table
trace 4-10
translation 3-30
TAI (International Atomic Time) related to UTC 4-32
TAM (TEST ADDRESSING MODE) instruction 7-129
TAR (TEST ACCESS) instruction 10-104
target instruction 7-74
TB (TEST BLOCK) instruction 10-107
TBDER (CONVERT HFP TO BFP) floating-point instruction
TBDR (CONVERT HFP TO BFP) floating-point instruction 9-9
TBEDR (CONVERT HFP TO BFP) floating point instruction 9-9
TCDB (TEST DATA CLASS) BFP instruction 19-46
TCEB (TEST DATA CLASS) BFP instruction 19-46
TCXB (TEST DATA CLASS) BFP instruction 19-46
termination
of I/O operations
*See* conclusion of I/O operations
of instruction execution 5-17
for exigent machine-check conditions 11-11
of unit of operation 5-18
for exigent machine-check conditions 11-11
termination code (in subchannel logout) 16-34
field-validity flag for 16-34
TEST ACCESS instruction 10-104
TEST ADDRESSING MODE instruction 7-129
TEST AND SET instruction 7-129
TEST BLOCK instruction 10-107
TEST DATA CLASS BFP instructions 19-46
TEST DECIMAL instruction 8-14
test indicator 12-5
TEST PENDING INTERRUPTION instruction 14-18, 14-19



LzLabs GmbH. Ex. 1008-1020

TEST PROTECTION instruction 10-109
TEST SUBCHANNEL instruction 14-20
  interruption-response block (IRB)used by 16-6
TEST UNDER MASK HIGH instruction 7-130
TEST UNDER MASK instruction 7-130
  examples A-31
TEST UNDER MASK LOW instruction 7-130
testing for storage-block and storage-key usability 10-107
THDER (CONVERT BFP TO HFP) floating-point instruction 9-8
THDR (CONVERT BFP TO HFP) floating-point instruction 9-8
TIC (transfer in channel) 15-41
  invalid sequence of 16-26
time-of-day clock
  *See* TOD clock
timer
  *See* CPU timer
timing
  channel-subsystem 17-2
timing facilities 4-29
timing-facility bit (in PMCW) 15-4
timing-facility damage 11-17
  for TOD clock 4-30
TLB (translation-lookaside buffer) 3-35
  entries 3-36
    attachment of 3-36
    clearing of 3-38
    effect of translation changes on 3-38
    usable state 3-37
TM (TEST UNDER MASK) instruction 7-130
  examples A-31
TMH (TEST UNDER MASK HIGH) instruction 7-130
TML (TEST UNDER MASK LOW) instruction 7-130
TOD clock 4-29
  effect of power-on reset on 4-42
  effect on clock-comparator interruption 6-11
  effect on CPU-timer decrementing 4-36
  effect on CPU-timer interruption 6-11
  manual control of 4-30, 12-5
  unique values of 4-31
  validation of 11-10
  value in trace entry 4-13
TOD-clock-control-override control 4-30
TOD-clock programmable field 4-32
TOD-clock programmable register 4-32
TOD-clock sync check (external interruption) 6-13
TOD-clock-sync-control bit 4-30, 4-34
TOD-clock-synchronization facility 4-34
TP (TEST DECIMAL) instruction 8-14
TPI (TEST PENDING INTERRUPTION) instruction 14-18, 14-19
  interruption code stored by 14-19
TPROT (TEST PROTECTION) instruction 10-109
TR (TRANSLATE) instruction 7-131
  example A-31
trace 4-10, F-2
  entries 4-11
  entry address 4-11
  exceptions 6-45
  table exception 6-34
TRACE instruction 10-111
  trace entry for 4-13
trailer entry 5-70
transfer in channel
  *See* TIC
transferring program control 5-61
TRANSLATE AND TEST instruction 7-132
  example A-32
TRANSLATE EXTENDED instruction 7-132
TRANSLATE instruction 7-131
  example A-31
TRANSLATE ONE TO ONE instruction 7-134
TRANSLATE ONE TO TWO instruction 7-135
TRANSLATE TWO TO ONE instruction 7-135
TRANSLATE TWO TO TWO instruction 7-135
translation
  address 3-26
    *See also* dynamic address translation
  exception identification 3-44
  lookaside buffer
    *See* TLB
  PC-number 5-27
  specification exception 6-34
  tables for 3-30
translation format 3-28
translation modes 3-28
translation parameters 3-27
trap control block 10-113
TRAP instruction 10-112
trap save area 10-113
TRAP2 (TRAP) instruction 10-112
TRAP4 (TRAP) instruction 10-112
TRE (TRANSLATE EXTENDED) instruction 7-132
tree structure for sorting 7-141
  example A-51
trial execution
  for editing instructions and TRANSLATE instruction 5-21
  for PER 4-17
TROO (TRANSLATE ONE TO ONE) instruction 7-134
TROT (TRANSLATE ONE TO TWO) instruction 7-135
TRT (TRANSLATE AND TEST) instruction 7-132
  example A-32
TRTO (TRANSLATE TWO TO ONE) instruction 7-135
TRTT (TRANSLATE TWO TO TWO) instruction 7-135
true zero (HFP number) 18-1
TS (TEST AND SET) instruction 7-129
TSCH (TEST SUBCHANNEL) instruction 14-20



LzLabs GmbH. Ex. 1008-1021

two's complement binary notation 7-3
examples A-2
type of PROGRAM CALL 5-65

# U

ulp (unit in the last place) 19-4
underflow
*See* exponent underflow
unit check (device status)
in establishing dedicated allegiance 15-12
unit of operation 5-17
unlock A-46
example with FIFO queuing A-48
example with LIFO queuing A-47
unnormalized floating-point number 18-3
HFP data only 9-1
unnormalized-operand exception 6-35
unordered (comparison to a NaN) 19-8
unordered comparison 19-23
UNPACK ASCII instruction 7-139
UNPACK instruction 7-139
example A-33
UNPACK UNICODE instruction 7-140
UNPK (UNPACK) instruction 7-139
example A-33
UNPKA (PACK ASCII) instruction 7-139
UNPKU (PACK UNICODE) instruction 7-140
unprivileged instructions 4-6, 7-2
unsigned binary
arithmetic 7-4
integer 7-2
examples A-3
in address generation 5-8
unsolicited interruption condition (I/O) 16-3
unstack-suppression bit 5-69
unstacking process 5-75
update reference 5-86
UPDATE TREE instruction 7-141
example A-51
UPT (UPDATE TREE) instruction 7-141
example A-51
usable ALB entry 5-54
usable TLB entry 3-37
UTC (Coordinated Universal Time) used in TOD epoch 4-32

# V

valid ART-table entry 5-54
valid CBC 11-2
valid floating-point-register numbers 9-2
valid segment-table or page-table entry 3-36
validation 11-5
of registers 11-10
of storage 11-6
validation *(continued)*
of storage key 11-7
of TOD clock 11-10
validity bit for backward stack-entry address 5-70
validity bit for forward-section-header address 5-70
validity bits
in machine-check-interruption code 11-22
in subchannel logout 16-34
variable-length field 3-3
vector facility 1-10, 2-6
effect of power-on reset on 4-42
vector-facility failure (machine-check condition) 11-18
vector-facility source (machine-check condition) 11-19
vector-operation exception 6-35
vector operations
publication referenced xxii
version code 10-90
virtual address 3-4
virtual machine
extensions for 1-9
virtual storage 3-26
virtual-address enhancement of suppression on protection 3-13
VM-data-space facility 1-9
volatile storage 3-2
effect of power-on reset on 4-42

# W

WAIT (SVC)
example of routine to bypass A-46
wait indicator 12-5
wait-state bit
in PSW 4-5
warning (machine-check condition) 11-18
subclass-mask bit for 11-26
word 3-3
word-concurrent storage references 5-88
working allegiance (I/O) 15-12
wraparound
of instruction addresses 5-7
of PER addresses 4-21
of register numbers
for LOAD MULTIPLE instruction 7-80
for STORE MULTIPLE instruction 7-126
of storage addresses 3-5
comparison of 370-XA with System/370 for F-7
controlled by addressing mode 3-5
for MOVE INVERSE instruction 7-83
for MOVE LONG EXTENDED instruction 7-88
for MOVE LONG instruction 7-84
for MOVE LONG UNICODE instruction 7-91
of TOD clock 4-29



LzLabs GmbH. Ex. 1008-1022

# X

X (EXCLUSIVE OR) instruction 7-74
X field of instruction 5-8
XA (extended architecture)
  *See* 370-XA architecture
XC (EXCLUSIVE OR) instruction 7-74
  examples A-19
XI (EXCLUSIVE OR) instruction 7-74
  example A-20
XR (EXCLUSIVE OR) instruction 7-74
XSCH (CANCEL SUBCHANNEL) instruction 14-4

# Z

Z bit (zero condition-code bit) 16-11
  as cause of intermediate interruption
    condition 16-17
ZAP (ZERO AND ADD) instruction 8-14
  example A-38
zero
  instruction-length code 6-7
  negative
    *See* negative zero
  normal meaning for byte value xxi
  setting floating-point register to 9-11
  true (HFP number) 18-1
ZERO AND ADD instruction 8-14
  example A-38
zero condition code (Z bit in SCSW) 16-11
zone bits 8-1
  moving of 7-97
zoned decimal numbers 8-1
  examples A-4



LzLabs GmbH. Ex. 1008-1023

# Communicating Your Comments to IBM

Enterprise Systems Architecture/390
Principles of Operation

Publication No. SA22-7201-08

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a reader's comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - FAX: (International Access Code)+1+845+432-9405
- If you prefer to send comments electronically, use one of these network IDs:
  - Internet e-mail: mhvrcfs@us.ibm.com
  - World Wide Web: http://www.ibm.com/s390/os390/webqs.html

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

LzLabs GmbH. Ex. 1008-1024

# Reader's Comments — We'd Like to Hear from You

**Enterprise Systems Architecture/390**
**Principles of Operation**

**Publication No. SA22-7201-08**

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

**Note:** Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: _____

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ] As an introduction
[ ] As a reference manual
[ ] For another purpose (explain)
[ ] As a text (student)
[ ] As a text (instructor)

_____

_____

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

_____ _____
Name Address

_____ _____
Company or Organization

_____ _____
Phone No.

LzLabs GmbH. Ex. 1008-1025

**Reader's Comments — We'd Like to Hear from You**
SA22-7201-08

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

SA22-7201-08

Cut or Fold
Along Line

LzLabs GmbH. Ex. 1008-1026



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SA22-7201-08



LzLabs GmbH. Ex. 1008-1028