IBM System/370 Extended Architecture

Principles of Operation

Publication Number
SA22-7085-0

File Number
S370-01

LzLabs GmbH. Ex. 1016-1

First Edition (March 1983)

Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM equipment, refer to the latest IBM System/370 and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

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 program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

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, comments may be addressed to IBM Corporation, Product Publications, Department B98, PO Box 390, Poughkeepsie, NY, U.S.A. 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1983

LzLabs GmbH. Ex. 1016-2

This publication provides, for reference purposes, a detailed definition of the machine functions performed by systems operating in the System/370 extended-architecture (370-XA) mode.

The publication applies only to systems operating in the 370-XA mode. The IBM System/370 Principles of Operation, GA22-7000, should be consulted regarding the functions of the architecture which apply to systems operating in the System/370 mode.

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 systems operating in the 370-XA mode 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 and, specifically, systems operating in the 370-XA mode, as found in Introduction to IBM Data Processing Systems, GC20-1684, and the IBM System/370 System Summary: Processors, GA22-7001. IBM publications relating to systems operating in the 370-XA mode are listed and described in the IBM System/370 and 4300 Processors Bibliography, GC20-0001.

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 channel subsystem, 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. The availability of facilities is summarized in the IBM System/370 System Summary: Processors, GA22-7001.

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 17 chapters and several appendixes:

Chapter 1, Introduction, highlights some of the major facilities of systems operating in the 370-XA mode.

Chapter 2, Organization, describes the major groupings within the system -- the central processing unit (CPU), storage, 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 in systems operating in the 370-XA mode. Dynamic address translation eliminates the need to assign a program to a fixed location in real storage and thus reduces the addressing constraints on system and problem programs.

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, program-event recording, timing facilities, resets, store status, and initial program loading.

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 also details the aspects of program execution on one CPU as observed by a channel program or another CPU program.

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

LzLabs GmbH. Ex. 1016-3

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 Instructions, contains detailed descriptions of floating-point data formats and the 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 370-XA I/O instructions.

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

Chapter 16, I/O Interruptions, covers I/O interruptions, interruption conditions, and the concluding of I/O operations.

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

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 the System/370 and 370-XA modes.
- A table of the powers of 2
- Tabular information helpful in dealing with hexadecimal numbers
- An EBCDIC chart

SIZE NOTATION

In this publication, the letters K, M, and G denote the multipliers 2^{10}, 2^{20}, and 2^{30}, respectively. Although the letters are borrowed from the decimal system and stand for kilo (10^3), mega (10^6), and giga (10^9), 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}

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

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.

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

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

LzLabs GmbH. Ex. 1016-4

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 the 370-XA mode 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 denotes a binary value. 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 I/O interface is described in the System Library publication IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information, GA22-6974.

The assists for MVS/XA are described in the System Library publication IBM Assists for MVS/XA, SA22-7092.

LzLabs GmbH. Ex. 1016-5

This page is intentionally left blank.

LzLabs GmbH. Ex. 1016-6

The meanings of abbreviations that are common in this publication are given in the following list. Only a few instruction mnemonics appear here; for a complete list of the mnemonics, see Appendix B. For other abbreviations not shown in this list, see the index.

AFT	ASN first table
AFTO	ASN-first-table origin
AFX	ASN-first-table index
AKM	authorization key mask
ASN	address-space number
AST	ASN second table
ASTE	AST entry
ASTO	AST origin
ASX	ASN-second-table index
AT	authority table
ATL	authority-table length
ATO	authority-table origin
AX	authority index
CBC	checking-block code
CCW	channel-command word
CHPID	channel-path identifier
CPU	central processing unit
CR	control register
CRW	channel-report word
DAT	dynamic address translation
EBCDIC	extended binary-coded-decimal interchange code
ECC	error-checking-and-correction code
EKM	entry key mask
ESW	extended-status word (word 3 of the IRB)
ET	entry table
ETL	entry-table length
ETO	entry-table origin
EX	entry index; execute
hex	hexadecimal
ID	identifier; identification
IDAW	indirect-data-address word
ILC	instruction-length code
IML	initial microprogram loading
I/O	input/output
IPL	initial program loading
IRB	interruption-response block
K	1,024 (bytes)
LPM	logical-path mask (in ORB)
LPUM	last-path-used mask (in SCHIB)
LT	linkage table
LTD	linkage-table designation
LTL	linkage-table length
LTO	linkage-table origin
LX	linkage index
M	1,048,576 (bytes)
OEMI	original equipment manufacturers' information
op code	operation code
ORB	operation-request block
PAM	path-available mask (in SCHIB)
PASN	primary ASN
PC	PROGRAM CALL
PC-cp	PROGRAM CALL to current primary
PC-ss	PROGRAM CALL with space switching
PCI	program-controlled interruption (flag in CCW or function)
PER	program-event recording
PFRA	page-frame real address
PIM	path-installed mask (in SCHIB)
PKM	PSW-key mask
PMCW	path-management-control word (words 0-6 of SCHIB)
PNOM	path-not-operational mask (in SCHIB)

LzLabs GmbH. Ex. 1016-7

POM	path-operational mask (in SCHIB)
PSTD	primary segment-table designation
PSTL	primary segment-table length
PSTO	primary segment-table origin
PSW	program-status word
PT	PROGRAM TRANSFER
PT-cp	PROGRAM TRANSFER to current primary
PT-ss	PROGRAM TRANSFER with space switching
PTL	page-table length
PTO	page-table origin
PX	page index
RR	register-and-register instruction format (or operation)
RRE	register-and-register instruction format (or operation) using an extended operation code
RS	register-and-storage instruction format (or operation)
RX	register-and-indexed-storage instruction format (or operation)
S	implied-operand-and-storage instruction format (or operation)
SASN	secondary ASN
SCHIB	subchannel-information block
SCSW	subchannel-status word
SI	storage-and-immediate-operand instruction format (or operation)
SID	subsystem-identification word
SLI	suppress length indication (flag in CCW)
SS	storage-and-storage instruction format (or operation)
SSAR	SET SECONDARY ASN
SSAR-cp	SET SECONDARY ASN to current primary
SSAR-ss	SET SECONDARY ASN with space switching
SSE	storage-and-storage instruction format (or operation) using an extended operation code
SSTD	secondary segment-table designation
SSTL	secondary segment-table length
SSTO	secondary segment-table origin
STD	segment-table designation
STL	segment-table length
STO	segment-table origin
sync	synchronization
TLB	translation-lookaside buffer
TOD	time of day
XA	extended architecture (architecture mode)

LzLabs GmbH. Ex. 1016-8

CHAPTER 1. INTRODUCTION 1-1
Highlights of 370-XA 1-1
Compatibility 1-3
Compatibility Among Systems in 370-XA Mode 1-3
Compatibility Between Systems in System/370 Mode and in 370-XA Mode 1-4
Control-Program Compatibility 1-4
Problem-State Compatibility 1-4
System Program 1-5
Availability 1-5

CHAPTER 2. ORGANIZATION 2-1
Main Storage 2-2
Central Processing Unit 2-2
Program-Status Word 2-3
General Registers 2-3
Floating-Point Registers . . . 2-3
Control Registers 2-3
Input and Output 2-5
Channel Subsystem 2-5
Input/Output Devices and Control Units 2-5
Operator Facilities 2-5
Service Processor 2-5

CHAPTER 3. STORAGE 3-1
Storage Addressing 3-2
Information Formats 3-2
Integral Boundaries 3-3
Address Types and Formats . . . 3-4
Address Types 3-4
Absolute Address 3-4
Real Address 3-4
Virtual Address 3-5
Primary Virtual Address . . 3-5
Secondary Virtual Address . 3-5
Logical Address 3-5
Instruction Address 3-5
Effective Address 3-5
Address Size and Wraparound . 3-6
Address Wraparound 3-6
Storage Key 3-8
Protection 3-8
Key-Controlled Protection . . 3-9
Fetch-Protection-Override Control 3-10
Page Protection 3-10
Low-Address Protection 3-11
Reference Recording 3-11
Change Recording 3-11
Prefixing 3-12
Address Spaces 3-13
ASN Translation 3-14
ASN-Translation Controls . . . 3-14
ASN-Translation Tables 3-15
ASN-First-Table Entries . . 3-15
ASN-Second-Table Entries . . 3-15
ASN-Translation Process . . . 3-16
ASN-First-Table Lookup . . . 3-17
ASN-Second-Table Lookup . . 3-17
Recognition of Exceptions during ASN Translation . . 3-18
ASN Authorization 3-18
ASN-Authorization Controls . . 3-18
Control Register 4 3-18
ASN-Second-Table Entry . . . 3-18
Authority-Table Entries . . 3-19
ASN-Authorization Process . . 3-19
Authority-Table Lookup . . . 3-20
Recognition of Exceptions during ASN Authorization . 3-21
Dynamic Address Translation . . 3-21
Translation Control 3-22
PSW 3-22
Control Register 0 3-23
Control Register 1 3-23
Control Register 7 3-24
Translation Tables 3-24
Segment-Table Entries . . . 3-24
Page-Table Entries 3-25
Summary of Segment-Table and Page-Table Sizes 3-25
Translation Process 3-26
Effective Segment-Table Designation 3-26
Inspection of Control Register 0 3-29
Segment-Table Lookup 3-29
Page-Table Lookup 3-29
Formation of the Real Address 3-30
Recognition of Exceptions during Translation 3-30
Translation-Lookaside Buffer . 3-30
Use of the Translation-Lookaside Buffer 3-31
Modification of Translation Tables 3-34
Address Summary 3-36
Addresses Translated 3-36
Handling of Addresses 3-37
Assigned Storage Locations . . . 3-38

CHAPTER 4. CONTROL 4-1
Stopped, Operating, Load, and Check-Stop States 4-2
Stopped State 4-2
Operating State 4-2
Load State 4-3
Check-Stop State 4-3
Program-Status Word 4-3
Program-Status-Word Format . . 4-5
Control Registers 4-6
Tracing 4-8
Control-Register Allocation . 4-8
Trace Entries 4-9
Operation 4-10
Program-Event Recording 4-11
Control-Register Allocation . 4-11
Operation 4-12
Identification of Cause . . 4-12
Priority of Indication . . . 4-13
Storage-Area Designation . . . 4-14
PER Events 4-14
Successful Branching 4-14
Instruction Fetching 4-14
Storage Alteration 4-15
General-Register Alteration 4-15
Indication of Events Concurrently with Other Interruption Conditions . . . 4-16
Timing 4-18
Time-of-Day Clock 4-18
Format 4-18
States 4-19

LzLabs GmbH. Ex. 1016-9

Changes in Clock State . . . 4-19
Setting and Inspecting the Clock . . . 4-20
TOD-Clock Synchronization . . 4-21
Clock Comparator . . . 4-21
CPU Timer . . . 4-22
Externally Initiated Functions . 4-23
Service Signal . . . 4-23
Resets . . . 4-24
CPU Reset . . . 4-27
Initial CPU Reset . . . 4-27
Subsystem Reset . . . 4-28
Clear Reset . . . 4-28
Power-On Reset . . . 4-28
Initial Program Loading . . . 4-29
Store Status . . . 4-29
Multiprocessing . . . 4-30
Shared Main Storage . . . 4-30
CPU-Address Identification . . 4-30
CPU Signaling and Response . . . 4-31
Signal-Processor Orders . . . 4-31
Conditions Determining Response 4-33
Conditions Precluding Interpretation of the Order Code . . . 4-33
Status Bits . . . 4-34

CHAPTER 5. PROGRAM EXECUTION . 5-1
Instructions . . . 5-2
Operands . . . 5-2
Instruction Format . . . 5-2
Register Operands . . . 5-4
Immediate Operands . . . 5-4
Storage Operands . . . 5-4
Address Generation . . . 5-4
Bimodal Addressing . . . 5-4
Sequential Instruction-Address Generation . . . 5-5
Operand-Address Generation . . 5-5
Formation of the Intermediate Value . . . 5-5
Formation of the Address . . 5-5
Branch-Address Generation . . 5-6
Formation of the Branch Address 5-6
Instruction Execution and Sequencing . . . 5-6
Decision-Making . . . 5-6
Loop Control . . . 5-7
Subroutine Linkage . . . 5-7
Interruptions . . . 5-11
Types of Instruction Ending . 5-11
Completion . . . 5-11
Suppression . . . 5-11
Nullification . . . 5-11
Termination . . . 5-11
Interruptible Instructions . . 5-11
Point of Interruption . . . 5-11
Execution of Interruptible Instructions . . . 5-12
Exceptions to Nullification and Suppression . . . 5-12
Storage Change and Restoration for DAT-Associated Access Exceptions . . . 5-13
Modification of DAT-Table Entries . . . 5-13
Trial Execution for Editing Instructions and TRANSLATE 5-14
Interlocked Update for Nullification and Suppression 5-14
Authorization Mechanisms . . . 5-14
Mode Requirements . . . 5-15
Extraction-Authority Control 5-15
PSW-Key Mask . . . 5-15
Secondary-Space Control . . 5-15
Subsystem-Linkage Control . 5-15
ASN-Translation Control . . 5-15
Authorization Index . . . 5-16
PC-Number Translation . . . 5-17
PC-Number Translation Control 5-18
PC-Number Translation Tables . 5-18
Linkage-Table Entries . . . 5-18
Entry-Table Entries . . . 5-18
PC-Number-Translation Process 5-19
Linkage-Table Lookup . . . 5-20
Entry-Table Lookup . . . 5-21
Recognition of Exceptions During PC-Number Translation 5-21
Sequence of Storage References . 5-21
Interlocks for Virtual-Storage References . . . 5-22
Instruction Fetching . . . 5-23
DAT-Table Fetches . . . 5-24
Storage-Key Accesses . . . 5-24
Storage-Operand References . . 5-25
Storage-Operand Fetch References . . . 5-25
Storage-Operand Store References . . . 5-25
Storage-Operand Update References . . . 5-26
Storage-Operand Consistency . 5-27
Single-Access References . . 5-27
Multiple-Access Operands . . 5-27
Block-Concurrent References 5-27
Consistency Specification . 5-27
Relation Between Operand Accesses . . . 5-28
Other Storage References . . . 5-29
Serialization . . . 5-29
CPU Serialization . . . 5-29
Channel-Program Serialization 5-30

CHAPTER 6. INTERRUPTIONS . . . 6-1
Interruption Action . . . 6-2
Interruption Code . . . 6-5
Enabling and Disabling . . . 6-5
Handling of Floating Interruption Conditions . . . 6-6
Instruction-Length Code . . . 6-6
Zero ILC . . . 6-6
ILC on Instruction-Fetching Exceptions . . . 6-6
Exceptions Associated with the PSW . . . 6-8
Early Exception Recognition 6-8
Late Exception Recognition . 6-8
External Interruption . . . 6-9
Clock Comparator . . . 6-9
CPU Timer . . . 6-10
Emergency Signal . . . 6-10
External Call . . . 6-10
Interrupt Key . . . 6-10
Malfunction Alert . . . 6-11
Service Signal . . . 6-11
TOD-Clock Sync Check . . . 6-11
Input/Output Interruption . . . 6-11
Machine-Check Interruption . . . 6-12
Program Interruption . . . 6-12
Program-Interruption Conditions 6-13
Addressing Exception . . . 6-13
AFX-Translation Exception . 6-15
ASN-Translation-Specification Exception . . . 6-15
ASX-Translation Exception . 6-15
Data Exception . . . 6-15
Decimal-Divide Exception . . 6-16
Decimal-Overflow Exception . 6-16

LzLabs GmbH. Ex. 1016-10

Execute Exception 6-16
Exponent-Overflow Exception 6-16
Exponent-Underflow Exception 6-16
EX-Translation Exception . . 6-17
Fixed-Point-Divide Exception 6-17
Fixed-Point-Overflow Exception 6-17
Floating-Point-Divide Exception 6-17
LX-Translation Exception . . 6-17
Monitor Event 6-18
Operand Exception 6-18
Operation Exception 6-18
Page-Translation Exception . 6-19
PC-Translation-Specification Exception 6-19
PER Event 6-19
Primary-Authority Exception 6-19
Privileged-Operation Exception 6-20
Protection Exception 6-20
Secondary-Authority Exception 6-21
Segment-Translation Exception 6-21
Significance Exception . . . 6-21
Space-Switch Event 6-21
Special-Operation Exception 6-22
Specification Exception . . 6-22
Programming Note 6-23
Trace-Table Exception . . . 6-23
Translation-Specification Exception 6-23
Collective Program-Interruption Names 6-24
Recognition of Access Exceptions 6-24
Multiple Program-Interruption Conditions 6-26
Access Exceptions 6-29
ASN-Translation Exceptions . 6-30
Trace Exceptions 6-31
Restart Interruption 6-31
Supervisor-Call Interruption . . 6-31
Priority of Interruptions . . . 6-32

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 7-3
Unsigned Binary Arithmetic . . 7-3
Signed and Logical Comparison . 7-4
Instructions 7-4
ADD 7-8
ADD HALFWORD 7-8
ADD LOGICAL 7-9
AND 7-9
BRANCH AND LINK 7-10
BRANCH AND SAVE 7-10
BRANCH AND SAVE AND SET MODE . 7-11
BRANCH AND SET MODE 7-11
BRANCH ON CONDITION 7-12
BRANCH ON COUNT 7-13
BRANCH ON INDEX HIGH 7-13
BRANCH ON INDEX LOW OR EQUAL . 7-13
COMPARE 7-14
COMPARE AND SWAP 7-14
COMPARE DOUBLE AND SWAP . . . 7-14
COMPARE HALFWORD 7-16
COMPARE LOGICAL 7-16
COMPARE LOGICAL CHARACTERS UNDER MASK 7-17
COMPARE LOGICAL LONG 7-17
CONVERT TO BINARY 7-19
CONVERT TO DECIMAL 7-20
DIVIDE 7-20
EXCLUSIVE OR 7-21
EXECUTE 7-22
INSERT CHARACTER 7-23
INSERT CHARACTERS UNDER MASK . 7-23
INSERT PROGRAM MASK 7-23
LOAD 7-24
LOAD ADDRESS 7-24
LOAD AND TEST 7-24
LOAD COMPLEMENT 7-24
LOAD HALFWORD 7-25
LOAD MULTIPLE 7-25
LOAD NEGATIVE 7-25
LOAD POSITIVE 7-26
MONITOR CALL 7-26
MOVE 7-27
MOVE LONG 7-27
MOVE NUMERICS 7-30
MOVE WITH OFFSET 7-31
MOVE ZONES 7-31
MULTIPLY 7-32
MULTIPLY HALFWORD 7-32
OR 7-33
PACK 7-33
SET PROGRAM MASK 7-34
SHIFT LEFT DOUBLE 7-34
SHIFT LEFT DOUBLE LOGICAL . . 7-35
SHIFT LEFT SINGLE 7-35
SHIFT LEFT SINGLE LOGICAL . . 7-36
SHIFT RIGHT DOUBLE 7-36
SHIFT RIGHT DOUBLE LOGICAL . . 7-36
SHIFT RIGHT SINGLE 7-37
SHIFT RIGHT SINGLE LOGICAL . . 7-37
STORE 7-37
STORE CHARACTER 7-38
STORE CHARACTERS UNDER MASK . 7-38
STORE CLOCK 7-38
STORE HALFWORD 7-39
STORE MULTIPLE 7-39
SUBTRACT 7-40
SUBTRACT HALFWORD 7-40
SUBTRACT LOGICAL 7-40
SUPERVISOR CALL 7-41
TEST AND SET 7-41
TEST UNDER MASK 7-42
TRANSLATE 7-42
TRANSLATE AND TEST 7-43
UNPACK 7-44

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
Instructions 8-3
ADD DECIMAL 8-5
COMPARE DECIMAL 8-5
DIVIDE DECIMAL 8-5
EDIT 8-6
EDIT AND MARK 8-9
MULTIPLY DECIMAL 8-10
SHIFT AND ROUND DECIMAL . . . 8-10
SUBTRACT DECIMAL 8-11
ZERO AND ADD 8-11

CHAPTER 9. FLOATING-POINT INSTRUCTIONS 9-1
Floating-Point Number Representation 9-1

LzLabs GmbH. Ex. 1016-11

Normalization 9-2
Floating-Point-Data Format . . . 9-2
Instructions 9-4
ADD NORMALIZED 9-6
ADD UNNORMALIZED 9-7
COMPARE 9-8
DIVIDE 9-8
HALVE 9-10
LOAD 9-10
LOAD AND TEST 9-11
LOAD COMPLEMENT 9-11
LOAD NEGATIVE 9-11
LOAD POSITIVE 9-12
LOAD ROUNDED 9-12
MULTIPLY 9-13
STORE 9-14
SUBTRACT NORMALIZED 9-14
SUBTRACT UNNORMALIZED 9-15

CHAPTER 10. CONTROL INSTRUCTIONS 10-1
DIAGNOSE 10-4
EXTRACT PRIMARY ASN 10-5
EXTRACT SECONDARY ASN 10-5
INSERT ADDRESS SPACE CONTROL . 10-6
INSERT PSW KEY 10-7
INSERT STORAGE KEY EXTENDED . 10-7
INSERT VIRTUAL STORAGE KEY . . 10-7
INVALIDATE PAGE TABLE ENTRY . 10-8
LOAD ADDRESS SPACE PARAMETERS 10-10
LOAD CONTROL 10-17
LOAD PSW 10-17
LOAD REAL ADDRESS 10-18
MOVE TO PRIMARY 10-19
MOVE TO SECONDARY 10-19
MOVE WITH KEY 10-20
PROGRAM CALL 10-22
PROGRAM TRANSFER 10-28
PURGE TLB 10-33
RESET REFERENCE BIT EXTENDED 10-33
SET ADDRESS SPACE CONTROL . 10-33
SET CLOCK 10-34
SET CLOCK COMPARATOR 10-35
SET CPU TIMER 10-35
SET PREFIX 10-36
SET PSW KEY FROM ADDRESS . . 10-36
SET SECONDARY ASN 10-37
SET STORAGE KEY EXTENDED . . 10-40
SET SYSTEM MASK 10-40
SIGNAL PROCESSOR 10-41
STORE CLOCK COMPARATOR . . . 10-42
STORE CONTROL 10-42
STORE CPU ADDRESS 10-43
STORE CPU ID 10-43
STORE CPU TIMER 10-43
STORE PREFIX 10-44
STORE THEN AND SYSTEM MASK . 10-44
STORE THEN OR SYSTEM MASK . 10-44
TEST BLOCK 10-45
TEST PROTECTION 10-47
TRACE 10-48

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 . . . 11-4
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 . . . 11-9
Check-Stop State 11-10
System Check Stop 11-11
Machine-Check Interruption . . 11-11
Exigent Conditions 11-11
Repressible Conditions . . . 11-11
Interruption Action 11-12
Point of Interruption . . . 11-13
Machine-Check-Interruption Code 11-14
Subclass 11-15
System Damage 11-15
Instruction-Processing Damage 11-15
System Recovery 11-16
Timing-Facility Damage . . 11-16
Degradation 11-16
Warning 11-16
Channel Report Pending . . 11-17
Service-Processor Damage . 11-17
Channel-Subsystem Damage . 11-17
Time of Interruption Occurrence 11-17
Backed Up 11-17
Synchronous Machine-Check Interruption Conditions . . 11-17
Processing Backup 11-17
Processing Damage 11-18
Storage-Error Type 11-18
Storage Error Uncorrected 11-18
Storage Error Corrected . 11-18
Storage-Key Error Uncorrected 11-18
Machine-Check Interruption-Code Validity Bits 11-19
PSW-MWP Validity 11-19
PSW Mask and Key Validity 11-19
PSW Program-Mask and Condition-Code Validity . 11-19
PSW-Instruction-Address Validity 11-19
Failing-Storage-Address Validity 11-19
Floating-Point-Register Validity 11-19
General-Register Validity 11-19
Control-Register Validity 11-20
Storage Logical Validity . 11-20
CPU-Timer Validity 11-20
Clock-Comparator Validity 11-20
Machine-Check Extended Interruption Information . . 11-20
Register-Save Areas 11-20
Failing-Storage Address . . 11-21
Handling of Machine-Check Conditions 11-21
Floating Interruption Conditions 11-21
Floating Machine-Check-Interruption Conditions 11-21
Floating I/O Interruptions 11-21
Machine-Check Masking 11-21
Channel-Report-Pending Subclass Mask 11-22
Recovery Subclass Mask . . 11-22
Degradation Subclass Mask 11-22

LzLabs GmbH. Ex. 1016-12

Timing-Facility-Damage Subclass Mask 11-22
Warning Subclass Mask . . 11-22
Machine-Check Logout 11-22
Summary of Machine-Check Masking 11-22

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 . 12-5

CHAPTER 13. I/O OVERVIEW . . . 13-1
Comparison with System/370 . . . 13-1
The Channel Subsystem 13-2
Subchannels 13-2
Attachment of Input/Output Devices 13-3
Channel Paths 13-3
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
Addresses Dependent on Channel-Path Type 13-5
I/O Operations 13-6
Start-Function Initiation . . 13-6
Path Management 13-6
Channel-Program Execution . . 13-7
Conclusion of I/O Operations . 13-7
I/O Interruptions 13-8

CHAPTER 14. I/O INSTRUCTIONS . 14-1
Introduction 14-1
I/O-Instruction Formats 14-1
I/O-Instruction Execution . . . 14-1
Serialization 14-1
Operand Access 14-2
Condition Code 14-2
Program Exceptions 14-2
Instructions 14-2
CLEAR SUBCHANNEL 14-3
HALT SUBCHANNEL 14-4
MODIFY SUBCHANNEL 14-6
RESET CHANNEL PATH 14-6
RESUME SUBCHANNEL 14-8
SET ADDRESS LIMIT 14-9
SET CHANNEL MONITOR 14-10
START SUBCHANNEL 14-11
STORE CHANNEL PATH STATUS . 14-13
STORE CHANNEL REPORT WORD . 14-13
STORE SUBCHANNEL 14-14
TEST PENDING INTERRUPTION . 14-15
TEST SUBCHANNEL 14-16

CHAPTER 15. BASIC I/O FUNCTIONS 15-1
Control of Basic I/O Functions . 15-1
Subchannel-Information Block (SCHIB) 15-1
Path-Management-Control Word 15-2
Subchannel-Status Word . . . 15-6
Model-Dependent Area 15-6
Summary of Modifiable Fields 15-6
Channel-Path Allegiance 15-9
Working Allegiance 15-9
Active Allegiance 15-9
Dedicated Allegiance 15-10
Channel-Path Availability . 15-10
Control-Unit Type 15-11
Clear-Function Execution . . . 15-11
Halt-Function Execution . . . 15-13
Start-Function and Resume-Function Execution . . 15-15
Execution of I/O Operations . 15-17
Programming Note 15-17
Blocking of Data 15-18
Operation-Request Block (ORB) 15-19
Channel-Command Word 15-20
Command Code 15-22
Designation of Storage Area 15-22
Chaining 15-23
Data Chaining 15-26
Command Chaining 15-27
Skipping 15-28
Program-Controlled Interruption 15-29
CCW Indirect Data Addressing 15-30
Suspension of Channel-Program Execution 15-31
Commands 15-32
Write 15-33
Read 15-34
Read Backward 15-34
Control 15-35
Sense 15-36
Sense ID 15-38
Transfer in Channel . . . 15-39
Command Retry 15-39
Concluding I/O Operations During Initiation 15-39
Immediate Conclusion of I/O Operations 15-40
Concluding I/O Operations During Data Transfer 15-41
Channel-Path-Reset-Function Execution 15-42

CHAPTER 16. I/O INTERRUPTIONS . 16-1
Interruption Conditions 16-2
Unsolicited Interruption Condition 16-3
Solicited Interruption Condition 16-3
Intermediate Interruption Condition 16-4
Primary Interruption Condition 16-4
Secondary Interruption Condition 16-4
Alert Interruption Condition . 16-5
Priority of Interruptions . . . 16-5
Interruption Action 16-6
Interruption-Response Block (IRB) 16-6
Subchannel-Status Word (SCSW) 16-7
Extended-Status Word 16-8
Extended-Control Word . . . 16-8
Subchannel-Status-Word Contents 16-8

LzLabs GmbH. Ex. 1016-13

Subchannel Key 16-8
Suspend Control (S) 16-8
Extended-Status-Word Format (L) 16-8
Deferred-Condition-Code Contents 16-9
Format (F) 16-11
Prefetch (P) 16-11
Initial-Status-Interruption Request (I) 16-12
Address-Limit-Checking Control (A) 16-12
Suppress-Suspended Interruption (U) 16-12
Subchannel-Control-Field Contents 16-12
Zero Condition Code (Z) . 16-12
Extended Control (E) . . . 16-12
Path Not Operational (N) . 16-13
Function Control 16-13
Activity Control 16-14
Status Control 16-16
Device-Status Conditions . . 16-18
Attention 16-19
Status Modifier 16-19
Control-Unit End 16-19
Busy 16-21
Channel End 16-21
Device End 16-21
Unit Check 16-22
Unit Exception 16-23
Subchannel-Status Conditions 16-23
Program-Controlled Interruption 16-24
Incorrect Length 16-24
Program Check 16-24
Protection Check 16-25
Channel-Data Check 16-26
Channel-Control Check . . 16-26
Interface-Control Check . 16-27
Chaining Check 16-27
CCW-Address-Field Contents . 16-28
Count-Field Contents 16-33
Extended-Status-Word Contents 16-35
Extended-Status Format 0 . . 16-36
Extended-Status Format 1 . . 16-40
Extended-Status Format 2 . . 16-40
Extended-Status Format 3 . . 16-41
Extended-Control Word 16-43

CHAPTER 17. I/O SUPPORT FUNCTIONS 17-1
Address-Limit Checking 17-1
Channel-Subsystem-Monitoring Facilities 17-2
Channel-Subsystem-Timing Facility 17-2
Channel-Subsystem Timer . . 17-2
Measurement-Block-Update Facility 17-4
Measurement Block 17-4
Measurement-Block Origin . . 17-5
Measurement-Block Key . . . 17-5
Measurement-Block Index . . 17-6
Measurement-Block-Update Mode 17-6
Measurement-Block-Update Enable 17-6
Time-Interval-Measurement Accuracy 17-6
Device-Connect-Time-Measurement Facility 17-7
Device-Connect-Time-Measurement Mode 17-7
Device-Connect-Time-Measurement Enable 17-7
Signals and Resets 17-8
Signals 17-8
Halt Signal 17-8
Clear Signal 17-8
Reset Signal 17-8
Resets 17-9
Channel-Path Reset 17-9
I/O-System Reset 17-9
Externally Initiated Functions 17-12
Initial Program Loading . . 17-12
Reconfiguration of the I/O System 17-14
Channel-Subsystem Recovery . . 17-14
Channel-Report Word (CRW) . 17-15
Channel Report 17-15
CRW Contents 17-16

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-4
Decimal Integers A-5
Floating-Point Numbers A-5
Conversion Example A-7
Instruction-Use Examples A-7
Machine Format A-7
Assembler-Language Format . . A-7
Addressing Mode in Examples A-8
General Instructions A-8
ADD HALFWORD (AH) A-8
AND (N, NR, NI, NC) A-8
And (NI) A-8
BRANCH AND LINK (BAL, BALR) . A-9
BRANCH ON CONDITION (BC, BCR) A-9
BRANCH ON COUNT (BCT, BCTR) . A-10
BRANCH ON INDEX HIGH (BXH) . . A-10
BXH Example 1 . . . . . A-10
BXH Example 2 . . . . . A-10
BRANCH ON INDEX LOW OR EQUAL (BXLE) A-11
COMPARE HALFWORD (CH) A-11
COMPARE LOGICAL (CL, CLC, CLI, CLR) A-11
Compare Logical (CLC) . . . A-11
Compare Logical (CLI) . . . A-12
Compare Logical (CLR) . . . A-12
COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) A-12
COMPARE LOGICAL LONG (CLCL) . A-13
CONVERT TO BINARY (CVB) . . . A-14
CONVERT TO DECIMAL (CVD) . . . A-14
DIVIDE (D, DR) A-15
EXCLUSIVE OR (X, XC, XI, XR) . A-15
Exclusive Or (XC) A-15
Exclusive Or (XI) A-16
EXECUTE (EX) A-17
INSERT CHARACTERS UNDER MASK (ICM) A-17
LOAD (L, LR) A-18
LOAD ADDRESS (LA) A-18
LOAD HALFWORD (LH) A-19
MOVE (MVC, MVI) A-19
Move (MVC) A-19
Move (MVI) A-20
MOVE LONG (MVCL) A-20
MOVE NUMERICS (MVN) A-21
MOVE WITH OFFSET (MVO) A-21
MOVE ZONES (MVZ) A-22
MULTIPLY (M, MR) A-22
MULTIPLY HALFWORD (MH) A-23
OR (O, OR, OI, OC) A-23
Or (OI) A-23
PACK (PACK) A-23

LzLabs GmbH. Ex. 1016-14

SHIFT LEFT DOUBLE (SLDA) . . . A-24
SHIFT LEFT SINGLE (SLA) . . . A-24
STORE CHARACTERS UNDER MASK (STCM) A-24
STORE MULTIPLE (STM) A-25
TEST UNDER MASK (TM) A-25
TRANSLATE (TR) A-26
TRANSLATE AND TEST (TRT) . . . A-26
UNPACK (UNPK) A-28
Decimal Instructions A-28
ADD DECIMAL (AP) A-28
COMPARE DECIMAL (CP) A-29
DIVIDE DECIMAL (DP) A-29
EDIT (ED) A-29
EDIT AND MARK (EDMK) A-31
MULTIPLY DECIMAL (MP) A-32
SHIFT AND ROUND DECIMAL (SRP) A-32
Decimal Left Shift A-32
Decimal Right Shift A-33
Decimal Right Shift and Round A-33
Multiplying by a Variable Power of 10 . . . . . A-33
ZERO AND ADD (ZAP) A-34
Floating-Point Instructions . . A-34
ADD NORMALIZED (AD, ADR, AE, AER, AXR) A-34
ADD UNNORMALIZED (AU, AUR, AW, AWR) A-34
COMPARE (CD, CDR, CE, CER) . . A-35
DIVIDE (DD, DDR, DE, DER) . . A-35
HALVE (HDR, HER) A-36
MULTIPLY (MD, MDR, ME, MER, MXD, MXDR, MXR) A-36
Floating-Point-Number Conversion A-36
Fixed Point to Floating Point A-37
Floating Point to Fixed Point A-37
Multiprogramming and Multiprocessing Examples . . . A-38
Example of a Program Failure Using OR Immediate A-38
COMPARE AND SWAP (CS, CDS) . . A-38
Setting a Single Bit A-39
Updating Counters A-39
Bypassing POST AND WAIT . . . A-40
BYPASS POST Routine A-40
BYPASS WAIT Routine A-40
LOCK/UNLOCK A-40
LOCK/UNLOCK with LIFO Queuing for Contentions A-41
LOCK/UNLOCK with FIFO Queuing for Contentions A-42
Free-Pool Manipulation A-44

APPENDIX B. LISTS OF INSTRUCTIONS B-1

APPENDIX C. CONDITION-CODE SETTINGS C-1

APPENDIX D. COMPARISON BETWEEN SYSTEM/370 AND 370-XA MODES . . D-1
New Facilities in 370-XA Mode . D-1
Bimodal Addressing D-1
31-Bit Logical Addressing . . D-1
31-Bit Real and Absolute Addressing D-1
Page Protection D-2
Tracing D-2
Comparison of Facilities D-2
Summary of Changes D-4
Changes in Instructions Provided D-4
Input/Output Comparison . . . D-6
Comparison of PSW Formats . . D-7
Changes in Control-Register Assignment D-7
Changes in Assigned Storage Locations D-9
SIGNAL PROCESSOR Changes . . . D-9
Machine-Check Changes D-9
Changes to Addressing Wraparound D-10
Changes to LOAD REAL ADDRESS . D-10
Changes to 31-Bit Real Operand Addresses D-10

APPENDIX E. TABLE OF POWERS OF 2 E-1

APPENDIX F. HEXADECIMAL TABLES F-1

APPENDIX G. EBCDIC CHART . . . G-1

INDEX X-1

LzLabs GmbH. Ex. 1016-15

CHAPTER 1. INTRODUCTION

Highlights of 370-XA 1-1
Compatibility 1-3
  Compatibility Among Systems in 370-XA Mode 1-3
  Compatibility Between Systems in System/370 Mode and in 370-XA Mode 1-4
    Control-Program Compatibility 1-4
    Problem-State Compatibility 1-4
System Program 1-5
Availability 1-5

This publication describes the architecture of systems operating in the IBM System/370 extended-architecture (370-XA) mode.

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 programs running on different machine implementations produce the results that are defined by a single architecture, the implementations are considered to be compatible.

HIGHLIGHTS OF 370-XA

The 370-XA mode has evolved from the System/370 architecture, with special attention paid to the implementation of large systems. It incorporates a number of significant new facilities beyond System/370. Some facilities available in the System/370 mode are changed or not provided in the 370-XA mode. A detailed comparison of the differences in the facilities and functions which are offered in the System/370 mode and in the 370-XA mode appears in Appendix D.

The most significant change from System/370 is in the I/O facilities provided by the channel subsystem. It includes these significant new capabilities:

- Path-independent addressing of I/O devices, which permits the initiation of I/O operations with any device 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 what 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. These functions are performed without interaction with the program.
- 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. This capability complements the path-management capability; together, they permit the channel subsystem and the I/O device to choose the first available path to initiate or continue execution of a chain of operations.
- 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. The format of the CCWs is specified when an I/O operation is initiated.
- Address-limit checking, which provides an additional storage-protection facility to prevent data access to storage locations above or below a specified absolute address. The absolute address-limit value can be set by an I/O instruction, and individual subchannels can be set up by another I/O instruction to allow data accesses to locations only at or

LzLabs GmbH. Ex. 1016-17

above, or only below, the limit address.

- Monitoring facilities, which can be invoked by the program to cause the channel subsystem to measure and accumulate, in main storage, key I/O-resource usage parameters for individual subchannels. The accumulated data-transfer time for a channel-program execution can be passed to the program with the ending status for that channel program.

- A set of 13 new I/O instructions, with associated control blocks, which are provided for the control of the channel subsystem.

The following is a summary of the other extensions incorporated in the 370-XA mode:

- Bimodal addressing provides two modes of operation: a 24-bit addressing mode for running old programs and a 31-bit addressing mode. The mode is controlled by a bit 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 extends the virtual address space from the 16M bytes addressable with 24-bit addresses to 2G bytes (2,147,483,648 bytes). In the 31-bit mode, address arithmetic and all logical addresses specified by instructions, as well as the address appearing in the program-status word (PSW), are expanded to 31 bits. Addresses appearing in control registers and permanently assigned storage locations are 31 bits, independent of the addressing mode.

- 31-bit real and absolute addressing provides addressability for up to 2G bytes of main storage. Associated with this extension, a number of formats are changed to provide for 31-bit address fields. These include the dynamic-address-translation and other table entries, the associated control registers, and the prefix register. The 31-bit-real-and-absolute-addressing facility replaces the extended-real-addressing facility of System/370, where page-table-entry bits 13 and 14 are used to extend the real address to 26 bits.

- The 370-XA protection facilities reflect the adoption of the 4K-byte block as the basic unit of storage allocation. Only one storage key is allocated to a 4K-byte protection block of storage; that is, the System/370 2K-byte block is not provided. Associated with the 4K-byte protection block is a control, called the fetch-protection-override control, that eliminates fetch protection for locations 0-2047 so as to permit access to status and control information located in the first 2K bytes of storage. Page protection, which is controlled by a bit in the page-table entry, replaces segment protection introduced for later models of System/370. The page-protection facility permits establishing read-only pages. As in System/370, low-address protection provides additional protection for the contents of storage locations 0 through 511.

- The tracing facility assists in the determination of system problems by providing an on-going record in storage of significant events. Branch tracing and ASN tracing may implicitly form entries in the trace table, whereas entries may be explicitly formed by the TRACE instruction. Each of the three types of tracing is separately controllable. A separate trace table is associated with each CPU. This facility replaces the MVS-oriented System/370 dual-address-space tracing.

- The two orders set prefix and store status at address provide additional capability for communication between CPUs by means of the SIGNAL PROCESSOR instruction.

- The DIVIDE (DXR) instruction provides for an extended-precision dividend, divisor, and quotient and thus rounds out the set of extended-precision floating-point instructions.

The following is a summary of the facilities appearing in the System/370 mode but not provided in the 370-XA mode:

- The System/370 I/O instructions and I/O interruptions, including all 10 System/370 I/O instructions, channel masks in control register 2, the block-multiplexing control in control register 0, and channel-set switching with the associated two instructions. These facilities are replaced by the 370-XA channel subsystem.

- The System/370 formats containing 24-bit addresses, which have been replaced by formats providing for 31-bit addresses. These include tables and control registers asso-

LzLabs GmbH. Ex. 1016-18

ciated with dynamic address translation and the dual-address-space facility.

- The basic-control mode and the associated PSW format, as well as the controls and information formats of the interruption mechanism. In the 370-XA mode, only the functions and format of the System/370 extended-control mode are available.
- The interval timer at storage location 80.
- The 2K-byte block associated with a storage key and the instructions INSERT STORAGE KEY, RESET REFERENCE BIT, and SET STORAGE KEY.
- Direct control, including the instructions READ DIRECT and WRITE DIRECT and the external signals.
- Certain System/370 machine-check and I/O-recovery facilities. In the 370-XA mode, these conditions either are encoded differently or the associated error-recording and recovery functions are performed by the machine without a need for bringing the associated information to the attention of the program. The facilities include the I/O extended logout and the associated control in control register 14, machine-check extended logout and the associated controls in control registers 14 and 15, limited-channel-logout extensions, and some machine-check indications.

Additionally, the 370-XA mode differs from the System/370 mode in that (1) the control-register assignment has been changed, (2) storage addresses for channel programs in the 24-bit mode cause an I/O program check instead of wraparound, (3) the extended-key instructions and TEST BLOCK are subject to the 24-bit and 31-bit addressing modes, and (4) it is unpredictable whether prefixing is applied to addressing of dynamic-address-translation tables.

Except for the facilities specifically identified as not provided, the 370-XA mode includes all facilities that are defined in the System/370 Principles of Operation. Most of the facilities that are considered features in the System/370 mode (because they are optional or unavailable on some models) are a standard part of the 370-XA mode.

Specifically, the 370-XA mode incorporates dynamic address translation, including the common-segment bit and the instructions INVALIDATE PAGE TABLE ENTRY and TEST PROTECTION introduced for later models of System/370. The table formats are modified to accommodate 31-bit real addresses, and, in contrast to the facility in the System/370 mode, this facility is available only with 1M-byte segments and 4K-byte pages, reflecting the larger virtual and real storage available on systems operating in the 370-XA mode.

Similarly, the 370-XA mode includes all of the functions (except for DAS tracing) of the System/370 dual-address-space facility. The 370-XA mode thus permits establishing addressability for up to 65,536 address spaces of 2G bytes each. A number of control-register and table formats, however, are changed to accommodate the 31-bit address fields.

The System/370 multiprocessing facilities, which include prefixing, CPU-address identification, CPU signaling and response, and TOD-clock synchronization are a basic part of the 370-XA mode. Thus, the instructions SET PREFIX, STORE PREFIX, STORE CPU ADDRESS, and SIGNAL PROCESSOR are operative even when no other CPU is in the configuration.

Even though the System/370 I/O facilities have generally been replaced by the channel subsystem in the 370-XA mode, and although a new channel-command-word (CCW) format is introduced to accommodate 31-bit addresses, the System/370 24-bit format, including the command codes and flags, is carried into the 370-XA mode. Similarly, the 370-XA mode incorporates the functions of the suspend-and-resume facility available on the later System/370 models. Compatibility with System/370 is maintained also in the physical attachment of I/O control units via the System/370 I/O interface.

COMPATIBILITY

COMPATIBILITY AMONG SYSTEMS IN 370-XA MODE

Although systems operating in the 370-XA mode 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 to operate in the 370-XA mode gives identical results on any system operating in that mode, 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

LzLabs GmbH. Ex. 1016-19

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 logout. 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 in this publication to be unpredictable or model-dependent, or on special-purpose functions (such as assists) that are not described in this publication. This includes the requirement that the 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 defined in this publication.

COMPATIBILITY BETWEEN SYSTEMS IN SYSTEM/370 MODE AND IN 370-XA MODE

Control-Program Compatibility

Control programs that were written to run on systems operating in the System/370 mode cannot be directly transferred from those systems to systems operating in the 370-XA mode. This is because in the 370-XA mode the BC mode is not present, new facilities for I/O are included, and the dynamic-address-translation facility is modified. (See Appendix D for a detailed comparison between the System/370 and 370-XA modes.)

To provide full control-program compatibility for the System/370 mode, all models which provide the 370-XA mode also offer manual controls that place the machine in the System/370 mode. When the system is in this mode, the operation of the system is as described in the IBM System/370 Principles of Operation, GA22-7000.

Problem-State Compatibility

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

A program written to run in the problem state on systems operating in the System/370 mode will run on a system operating in the 370-XA mode, provided that the program:

1. Observes the limitations described in the section "Compatibility Among Systems in 370-XA Mode."

2. Is not dependent on control-program facilities which are unavailable on the system.

3. Takes into account other changes made to the System/370 architectural definition that affect compatibility between the System/370 mode and the 370-XA mode. These changes are described in Appendix D.

Programming Note

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 run 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 this location to be changed.

LzLabs GmbH. Ex. 1016-20

SYSTEM PROGRAM

The system is designed to operate 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.

AVAILABILITY

Availability is the capability of a system to accept and successfully process an individual job. Systems operating in the 370-XA mode 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, 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.
- 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 running 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.

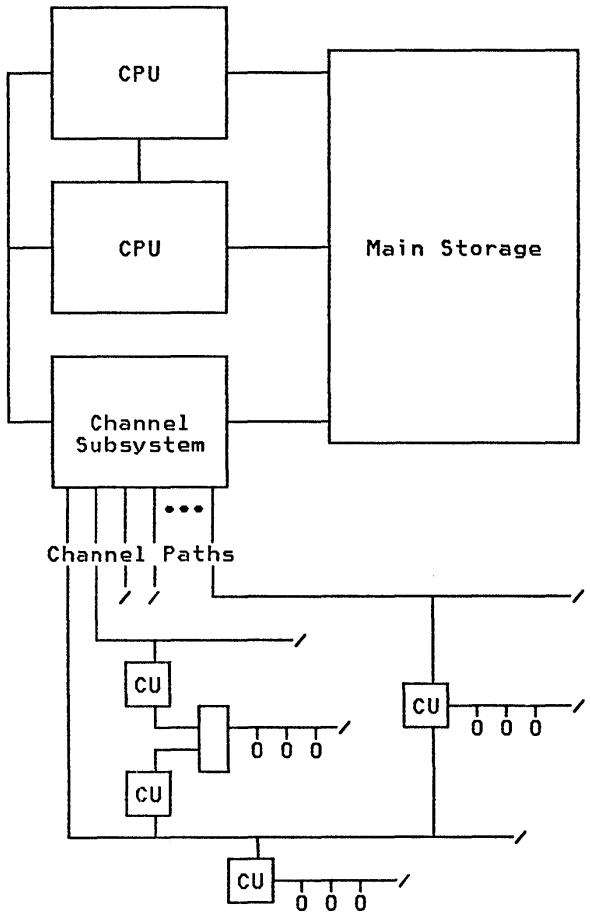
LzLabs GmbH. Ex. 1016-21

CHAPTER 2. ORGANIZATION

Main Storage 2-2
Central Processing Unit 2-2
  Program-Status Word 2-3
  General Registers 2-3
  Floating-Point Registers 2-3
  Control Registers 2-3
Input and Output 2-5
  Channel Subsystem 2-5
  Input/Output Devices and Control Units 2-5
Operator Facilities 2-5
Service Processor 2-5

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. The physical identity of these functions may vary among implementations, called "models." The figure "Logical Structure of a 370-XA System With Two CPUs" depicts the logical structure of a two-CPU multiprocessing system.

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 storage, and the representation of the operator facilities. The differences in internal characteristics are apparent to the observer only as differences in machine performance.



Logical Structure of a 370-XA System with Two CPUs

LzLabs GmbH. Ex. 1016-23

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. Model-dependent reconfiguration controls may be provided to change the amount of main 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 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 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, and main-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 on 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 in time, 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 be either physically integrated with a CPU or constructed as standalone units. Additionally, main storage may be composed of large-capacity storage and 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.

CENTRAL PROCESSING UNIT

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 compatibility rules are observed.

The CPU, in executing instructions, can process binary integers and floating-point numbers 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 CPU to another without affecting the logical results.

Instructions which the CPU executes fall into five classes: general, decimal, floating-point, control, and I/O 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, and the floating-point instructions on data in the floating-point format. 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.

To perform its functions, the CPU may use a certain amount of internal storage. Although this internal storage may use the same physical storage medium as main storage, it is not considered part of main storage and is not addressable by programs.

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, the control 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

LzLabs GmbH. Ex. 1016-24

operation. See the figure "General, Floating-Point, and Control Registers" later in this chapter for the format of those registers.

PROGRAM-STATUS WORD

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 interrupting program reloads 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 bits. 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

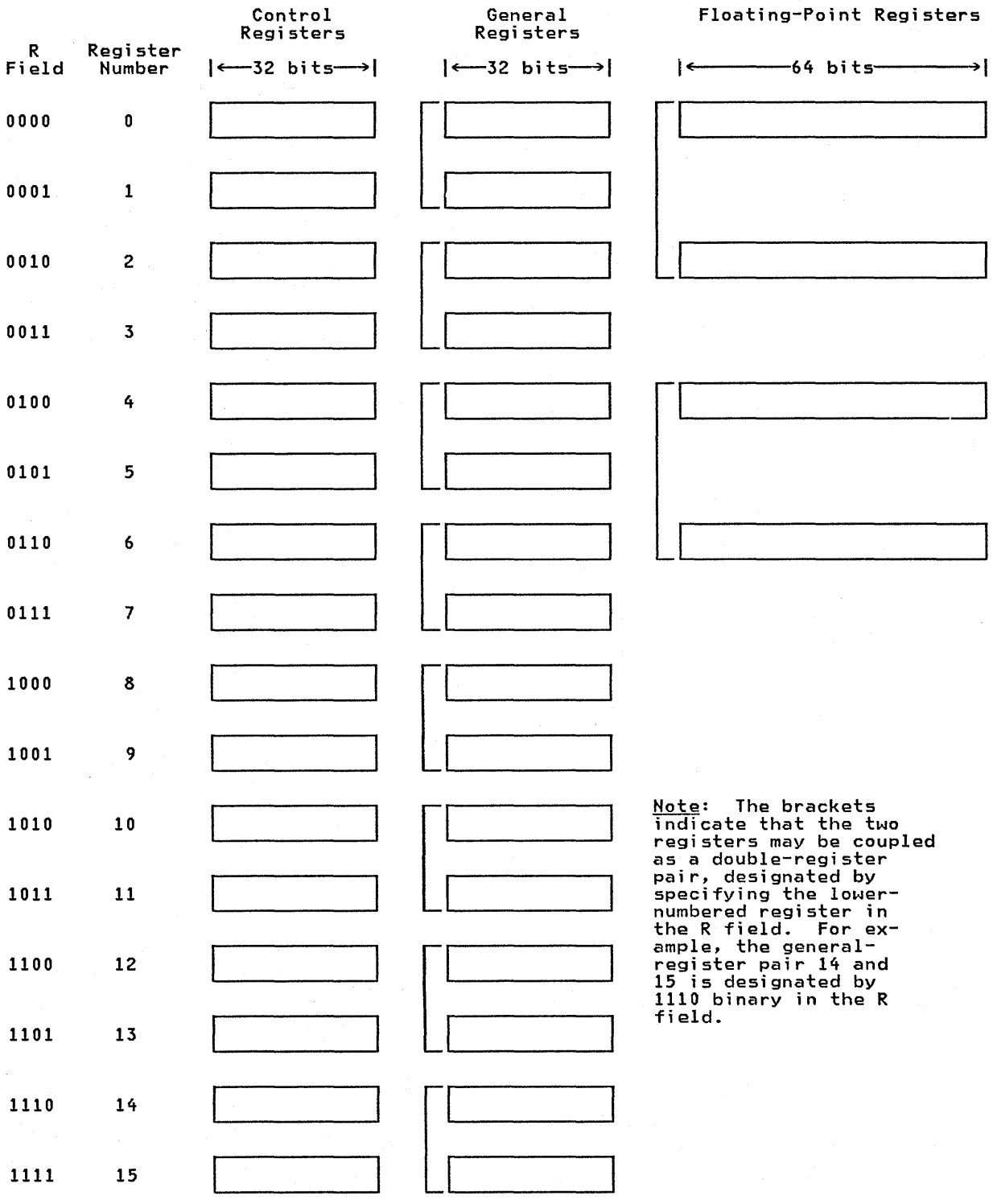
Four floating-point registers are available for floating-point operations. They are identified by the numbers 0, 2, 4, and 6 and are designated by a four-bit R field in floating-point instructions. Each floating-point register is 64 bits long and can contain either a short (32-bit) or a long (64-bit) floating-point operand. A short operand occupies the leftmost bit positions of a floating-point register. The rightmost portion of the register is ignored in operations that use short operands and remains unchanged in operations that produce short results. Two pairs of adjacent floating-point registers can be used for extended operands: registers 0 and 2, and registers 4 and 6. Each of these pairs, identified by the numbers 0 and 4, provides for a 128-bit format.

CONTROL REGISTERS

The CPU has provisions for 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.

LzLabs GmbH. Ex. 1016-25



General, Floating-Point, and Control Registers

LzLabs GmbH. Ex. 1016-26

INPUT AND OUTPUT

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 64K.

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 the 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.

INPUT/OUTPUT DEVICES AND CONTROL UNITS

Input/output devices include such equipment as card readers and punches, magnetic-tape units, direct-access storage, displays, keyboards, printers, teleprocessing devices, communications controllers, and sensor-based equipment. Many I/O devices function with an external medium, such as punched cards or magnetic tape. Some I/O devices handle only electrical signals, such as those found in sensor-based networks. In either case, I/O-device operation is regulated by a control unit. In all cases, the control-unit function 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.

SERVICE PROCESSOR

Depending on the model, a service processor may be provided. The service processor, which is intended primarily for maintenance of the system, may perform (1) some or all of the functions associated with initial microprogram loading, resets, and other operator facilities, (2) CPU retry and other recovery actions associated with machine-check handling, and (3) reconfiguration operations.

Normally, the existence of the service processor is not apparent to the program since the functions involved could be implemented by any physical unit in the system with the same logical results. However, the service processor can generate two interruptions: a service-signal external interruption and a service-processor-damage machine-check interruption.

LzLabs GmbH. Ex. 1016-27

Storage Addressing 3-2
Information Formats 3-2
Integral Boundaries 3-3
Address Types and Formats 3-4
Address Types 3-4
Absolute Address 3-4
Real Address 3-4
Virtual Address 3-5
Primary Virtual Address 3-5
Secondary Virtual Address 3-5
Logical Address 3-5
Instruction Address 3-5
Effective Address 3-5
Address Size and Wraparound 3-6
Address Wraparound 3-6
Storage Key 3-8
Protection 3-8
Key-Controlled Protection 3-9
Fetch-Protection-Override Control 3-10
Page Protection 3-10
Low-Address Protection 3-11
Reference Recording 3-11
Change Recording 3-11
Prefixing 3-12
Address Spaces 3-13
ASN Translation 3-14
ASN-Translation Controls 3-14
ASN-Translation Tables 3-15
ASN-First-Table Entries 3-15
ASN-Second-Table Entries 3-15
ASN-Translation Process 3-16
ASN-First-Table Lookup 3-17
ASN-Second-Table Lookup 3-17
Recognition of Exceptions During ASN Translation 3-18
ASN Authorization 3-18
ASN-Authorization Controls 3-18
Control Register 4 3-18
ASN-Second-Table Entry 3-18
Authority-Table Entries 3-19
ASN-Authorization Process 3-19
Authority-Table Lookup 3-20
Recognition of Exceptions During ASN Authorization 3-21
Dynamic Address Translation 3-21
Translation Control 3-22
PSW 3-22
Control Register 0 3-23
Control Register 1 3-23
Control Register 7 3-24
Translation Tables 3-24
Segment-Table Entries 3-24
Page-Table Entries 3-25
Summary of Segment-Table and Page-Table Sizes 3-25
Translation Process 3-26
Effective Segment-Table Designation 3-26
Inspection of Control Register 0 3-29
Segment-Table Lookup 3-29
Page-Table Lookup 3-29
Formation of the Real Address 3-30
Recognition of Exceptions During Translation 3-30
Translation-Lookaside Buffer 3-30
Use of the Translation-Lookaside Buffer 3-31
Modification of Translation Tables 3-34
Address Summary 3-36
Addresses Translated 3-36
Handling of Addresses 3-37
Assigned Storage Locations 3-38

LzLabs GmbH. Ex. 1016-29

This chapter discusses the representation of information in storage, the addressing of information, 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 storage (from input devices) before they can be processed.

Main storage may consist of standalone units or be integrated with a CPU. Additionally, main storage may be composed of a large-capacity storage and a smaller 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 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 that assigns highest priority to references by the channel subsystem and that rotates priority among CPUs. 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 by the channel subsystem when power is being turned off. In both types of main storage, the contents of the storage key 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 direct-access storage devices provide. "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 the section "Address Size and Wraparound" in this chapter.

INFORMATION FORMATS

Information is transmitted between storage and the 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

LzLabs GmbH. Ex. 1016-30

controlled 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.

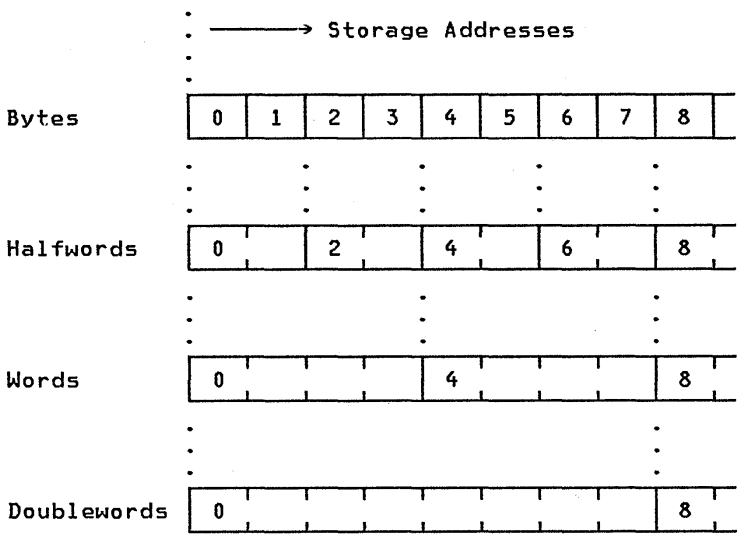
INTEGRAL BOUNDARIES

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 the figure "Integral Boundaries with Storage Addresses.")

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.

LzLabs GmbH. Ex. 1016-31



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.

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

LzLabs GmbH. Ex. 1016-32

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 and instruction addresses are treated as primary virtual addresses when in primary-space 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 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.

Logical Address

Except where otherwise specified, the storage-operand addresses for instructions are logical addresses. Logical addresses are treated as real addresses in real mode, treated as primary virtual addresses in primary-space mode, and treated as secondary virtual addresses in secondary-space mode. Some instructions have storage-operand addresses or storage accesses associated with the instruction which do not follow the rules for logical addresses. 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 real mode, treated as primary virtual addresses in primary-space mode, and treated as either primary virtual addresses or secondary virtual addresses in secondary-space mode. The instruction address in the current PSW and the target address of EXECUTE are instruction addresses.

Note: When the CPU is in the secondary-space mode, it is unpredictable whether instructions, and the target of EXECUTE, are fetched from the primary address space or the secondary address space. However, all copies of an instruction used in a single execution are fetched from a single address space, and the machine can change to or from interpreting instruction addresses as primary virtual or secondary virtual only between instructions and only by issuing a checkpoint-synchronizing function.

Programming Notes

1. Predictable program operation is ensured in secondary-space mode only when the instructions are fetched from virtual-address locations which translate to the same real address by means of both the primary and secondary segment tables. Thus, a program should not enter secondary-space mode if it is not aware of the virtual-to-real address mapping in both the primary and secondary address spaces.

2. The requirement limiting when the CPU can change the address space used for fetching instructions eliminates problems with CPU retry, DAT pretesting, and trial execution of instructions for the purposes of determining PER events.

Effective Address

In some situations, it is convenient to use the term "effective address." An effective address is the address which results from address arithmetic, before address translation, if any, is performed. Address arithmetic is the

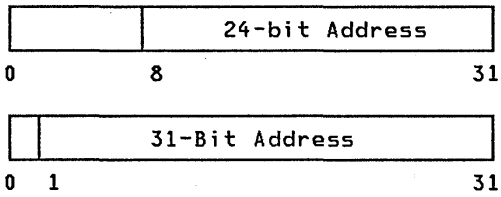
LzLabs GmbH. Ex. 1016-33

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 the section "Address Generation" in Chapter 5, "Program Execution").

The size of the real addresses yielded by the ASN-translation, PC-number-translation ASN-authorization, 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 specified 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 generates an address when it increments an address to fetch a CCW, to fetch an IDAW, or to transfer data.

When, during the generation of the address, an address is obtained that exceeds the value 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 always wrap, except for addresses generated for DAT-table entries. For DAT-table entries, it is unpredictable whether the address wraps or whether 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.

The figure "Address-Space Wraparound" 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. 1016-34

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	A or R[2]	X31
DAT-table entries when used for LRA	A or R[2]	X31
ASN-first-table, ASN-second-table, authorization-table, linkage-table, and entry-table entries	R	W31
I/O measurement block	A	P31
For a channel program with format-0 CCWs:		
Channel-program address in ORB	A	P24
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:		
Channel-program address in ORB	A	P31
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

Address Wraparound (Part 1 of 2)

LzLabs GmbH. Ex. 1016-35

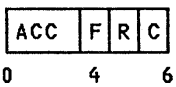
Explanation:

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 model-dependent whether the address wraps to location 0 after location $2^{31} - 1$ or whether an addressing exception is recognized.
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	The choices are model-dependent.

Address Wraparound (Part 2 of 2)

STORAGE KEY

A storage key is associated with each 4K-byte block of storage that is available in the configuration.



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, or 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 both to 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

Three 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, page protection, and low-address protection. The protection facilities are applied independently; access to main storage is only permitted when none of the facilities prohibit the access.

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

LzLabs GmbH. Ex. 1016-36

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 the figure "Summary of Protection Action."

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.

Summary of Protection Action

LzLabs GmbH. Ex. 1016-37

When the access to storage is initiated by the CPU, and key-controlled protection applies, the PSW key is the access key, except that, for the second operand of MOVE WITH KEY and MOVE TO PRIMARY and the first operand of MOVE TO SECONDARY, the access key is specified in a general register. 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 unit of operation is suppressed or the instruction is terminated, and a program interruption for a protection exception takes place. 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 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 is not applied to access to the second operand of TEST BLOCK.

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, an I/O 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 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.

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. 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.

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.

LzLabs GmbH. Ex. 1016-38

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, a program interruption for protection takes place. The contents of the protected location remain unchanged.

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

LOW-ADDRESS PROTECTION

The low-address-protection facility provides protection against the destruction of main-storage information used by the CPU during interruption processing, by prohibiting instructions from storing by using 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.

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, a program interruption for a protection exception takes place, and the unit of operation is suppressed or the instruction terminated.

Any attempt by the program to store by using effective addresses in the range 0 through 511 are 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, 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 Note

Low-address protection and key-controlled protection apply to the same store accesses, except that:

- Low-address protection does not apply to storing performed by the channel subsystem, whereas key-controlled protection does.
- Key-controlled protection does not apply to tracing or the second operand of TEST BLOCK, whereas low-address protection does.

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 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
INSERT VIRTUAL STORAGE KEY
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

LzLabs GmbH. Ex. 1016-39

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 in 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.

Change recording does not take place for the operands of the following instructions since they 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 the section "CPU Retry" in Chapter 11, "Machine-Check Handling"). See the section "Exceptions to Nullification and Suppression" in Chapter 5, "Program Execution," for a description of the handling of the change bit in certain unusual situations.

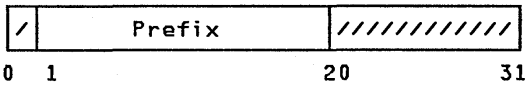
PREFIXING

Prefixing provides the ability to assign the range of real addresses 0-4095 (the prefix area) 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 absolute addresses 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 absolute 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 the figure "Relationship between Real and Absolute Addresses."

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:

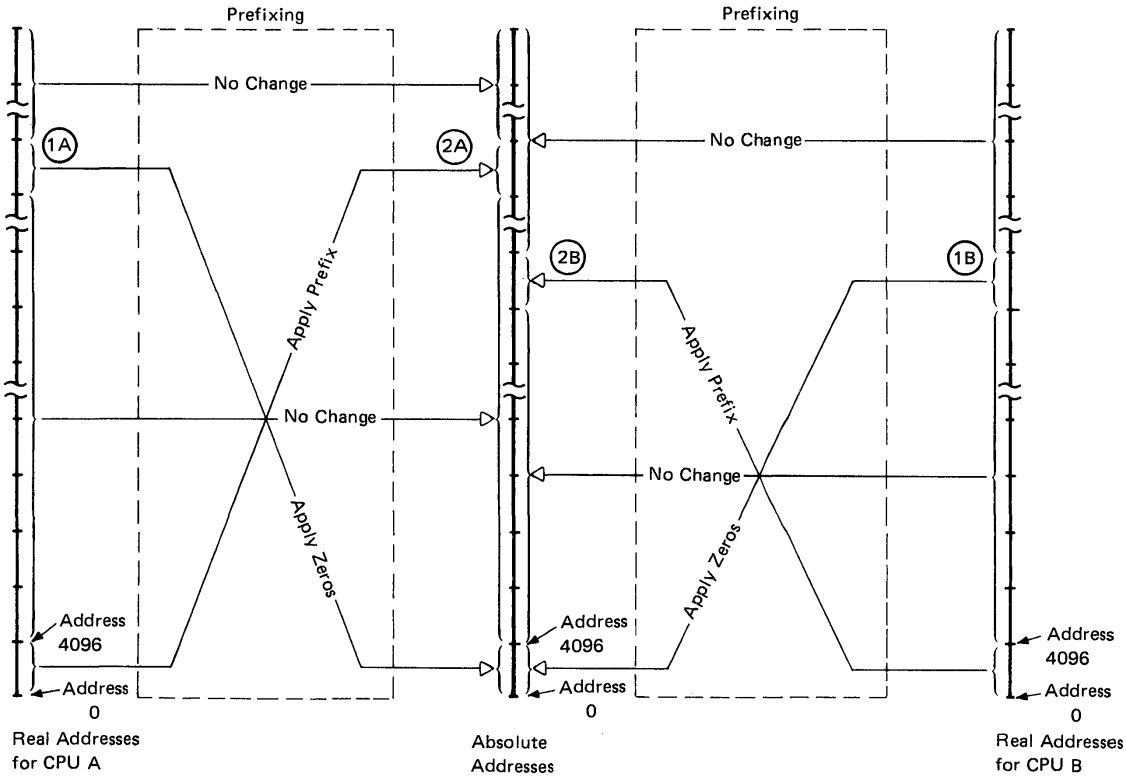
1. Bits 1-19 of the address, if all zeros, are replaced with bits 1-19 of the prefix.

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.

LzLabs GmbH. Ex. 1016-40



(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.

Relationship between Real and Absolute Addresses

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.

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.

At any instant the CPU can translate virtual addresses of two address spaces -- the primary address space, consisting of primary virtual addresses, and the secondary address space, consisting of secondary virtual addresses. The segment table defining the primary address space is specified by control register 1 and that defining the secondary address space by control register 7.

Each address space is assigned an address-space number (ASN). An ASN-translation mechanism is provided which, given an ASN, can locate (by using a two-level table lookup) the designation

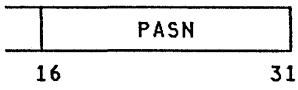
LzLabs GmbH. Ex. 1016-41

of the segment table which defines the address space. Certain instructions use ASN translation and load the resulting segment-table designation into the appropriate control register.

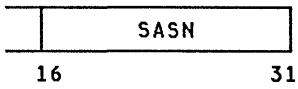
By using the ASN-translation mechanism, any one of up to 64K address spaces can be selected to become the primary or secondary address space.

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.

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.

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

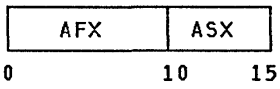
ASN TRANSLATION

ASN translation is the process of translating the 16-bit ASN to locate the address-space-control parameters. ASN translation is performed as part of PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), and SET SECONDARY ASN with space switching (SSAR-ss). ASN translation is also 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.

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.

ASN

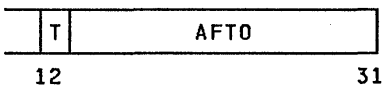


The AFX is used to select an entry from the ASN first table. The origin of the ASN first table is designated by the ASN-first-table origin in control register 14. The ASN-first-table entry contains the origin of the ASN second table. The ASX is used to select an entry from the ASN second table. This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

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
- SET SECONDARY ASN
- PROGRAM CALL with space switching
- PROGRAM TRANSFER with space switching

Otherwise, a special-operation exception is recognized. The ASN-translation-

LzLabs GmbH. Ex. 1016-42

control bit is examined in both the problem and supervisor states.

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.

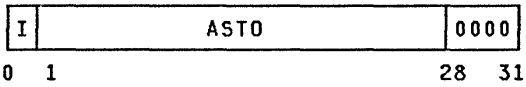
ASN-TRANSLATION TABLES

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

The entry fetched from the ASN first table (AFT) designates the availability and origin of the corresponding ASN second table.

An entry in the ASN first table 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.

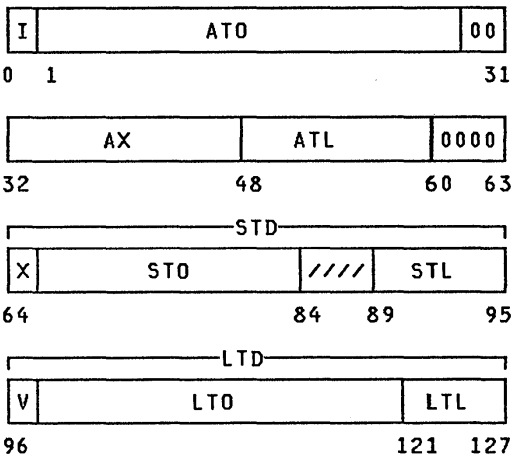
ASN-Second-Table Origin (ASTO): Bits 1-27, with four zeros appended on the right, are used to form a 31-bit real address that designates the beginning of the ASN second table.

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

ASN-Second-Table Entries

The entry fetched from the ASN second table indicates the availability of the address space and contains the address-space-control parameters if the address space is available.

The ASN second-table entry has the following format:



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.

Authorization Index (AX): Bits 32-47 are used as a result of primary ASN translation by PROGRAM CALL and PROGRAM TRANSFER and may be used by 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 is placed in the SSTD, bits 0-31 of control register 7. For PROGRAM CALL, the STD field is placed in 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 is placed in both the PSTD and SSTD, bits 0-31 of control registers 1 and 7, respectively. The contents of

LzLabs GmbH. Ex. 1016-43

the entire STD field are placed in the appropriate control registers without being inspected for validity.

Space-Switch-Event Control (X): Bit 0 of the segment-table designation is the space-switch-event-control bit. When, in PC-ss or PT-ss, this bit is one in control register 1 either before or after the execution of the PC-ss or PT-ss, a program interruption for a space-switch event occurs after the execution of the instruction is completed. 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 are used as a result of primary ASN 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). The contents of the LTD field are placed in control register 5 as a result of primary ASN translation.

Bits 30, 31, and 60-63 of the AST entry must be zeros; otherwise, an ASN-translation-specification exception is recognized as part of the execution of the instruction using that entry for ASN translation.

Programming Note

The unused portion of the STD field, bits 84-88 of the AST entry, which corresponds to bits 20-24 of the PSTD and SSTD, 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 PROGRAM CALL with space switching, PROGRAM TRANSFER with space switching, and SET SECONDARY ASN with space switching. 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.

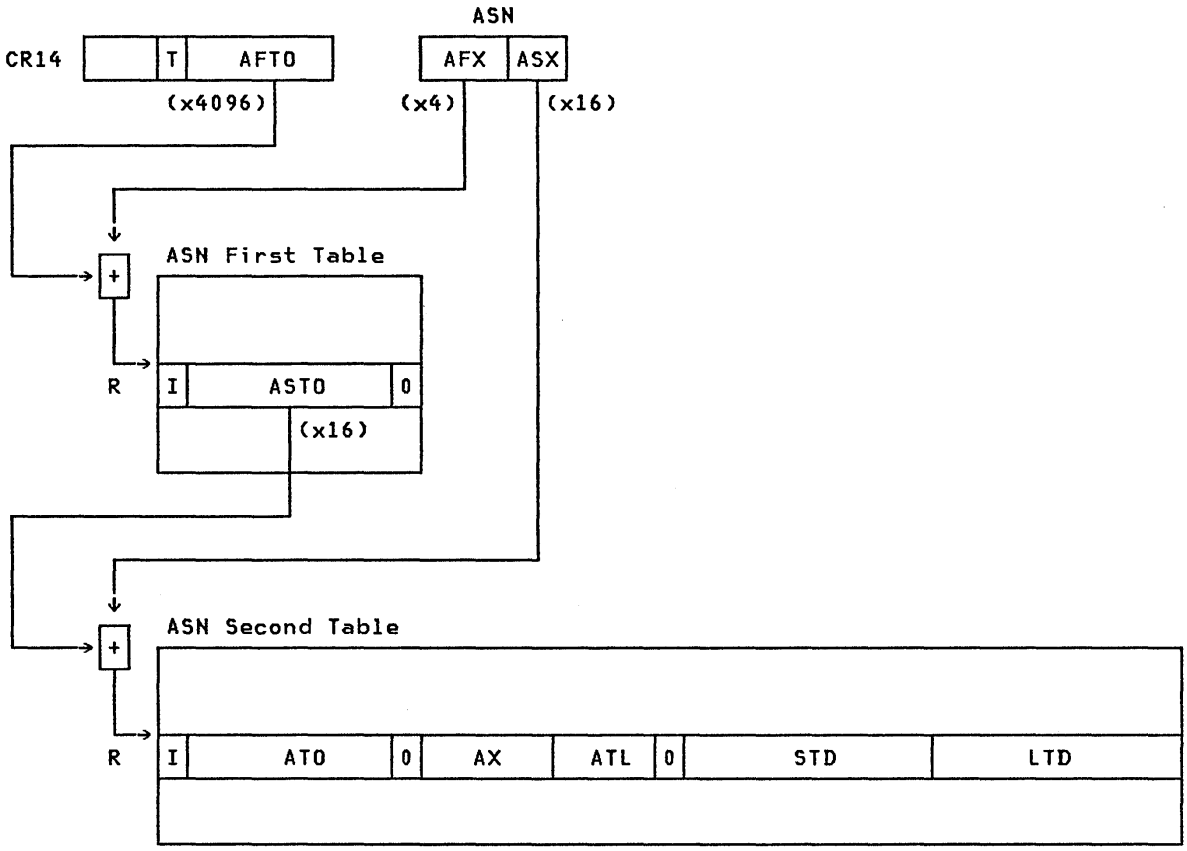
The ASN second index is used to select an entry from the ASN second table. This entry contains the address-space-control parameters.

If the I bit is one in either the ASN-first-table entry or 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 the figure "ASN Translation."

LzLabs GmbH. Ex. 1016-44



R: Address is real

ASN Translation

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 second 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.

All four bytes of the ASN-first-table entry are fetched concurrently. 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. If bit positions 28-31 of the AFT entry do not contain zeros, an ASN-translation-specification exception is 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.

The 31-bit real address of the ASN-second-table entry is obtained by

LzLabs GmbH. Ex. 1016-45

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. A carry, if any, into bit position 0 is ignored. All 31 bits of the address are used, regardless of whether the current PSW specifies the 24-bit or 31-bit addressing mode.

The 16 bytes of the ASN-second-table entry are fetched left to right, a word at a time. 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 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 is recognized.

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.

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 or SSAR-ss, the ASN authorization tests whether the ASN can be established as the secondary ASN and is called secondary-ASN authorization.

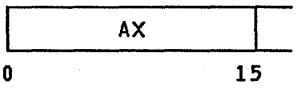
The ASN authorization is performed by means of an authority table in real storage which is designated by the authority-table-origin and authority-table-length fields in the ASN-second-table entry.

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.

Control Register 4

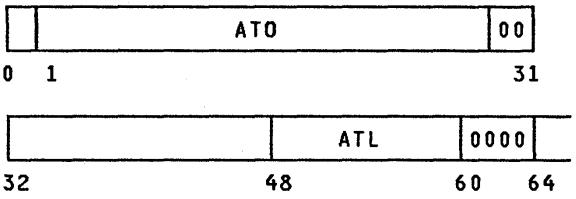
For PT-ss and SSAR-ss, the current contents of control register 4 contain the authorization index. For LOAD ADDRESS SPACE PARAMETERS, the value which will become the new contents of control register 4 is used. The register has the following format:



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.



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.

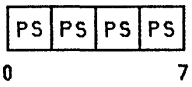
Authority-Table Length (ATL): Bits 48-59 designate the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four

LzLabs GmbH. Ex. 1016-46

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:



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 access is permitted. If the P bit is zero, the access is not permitted.

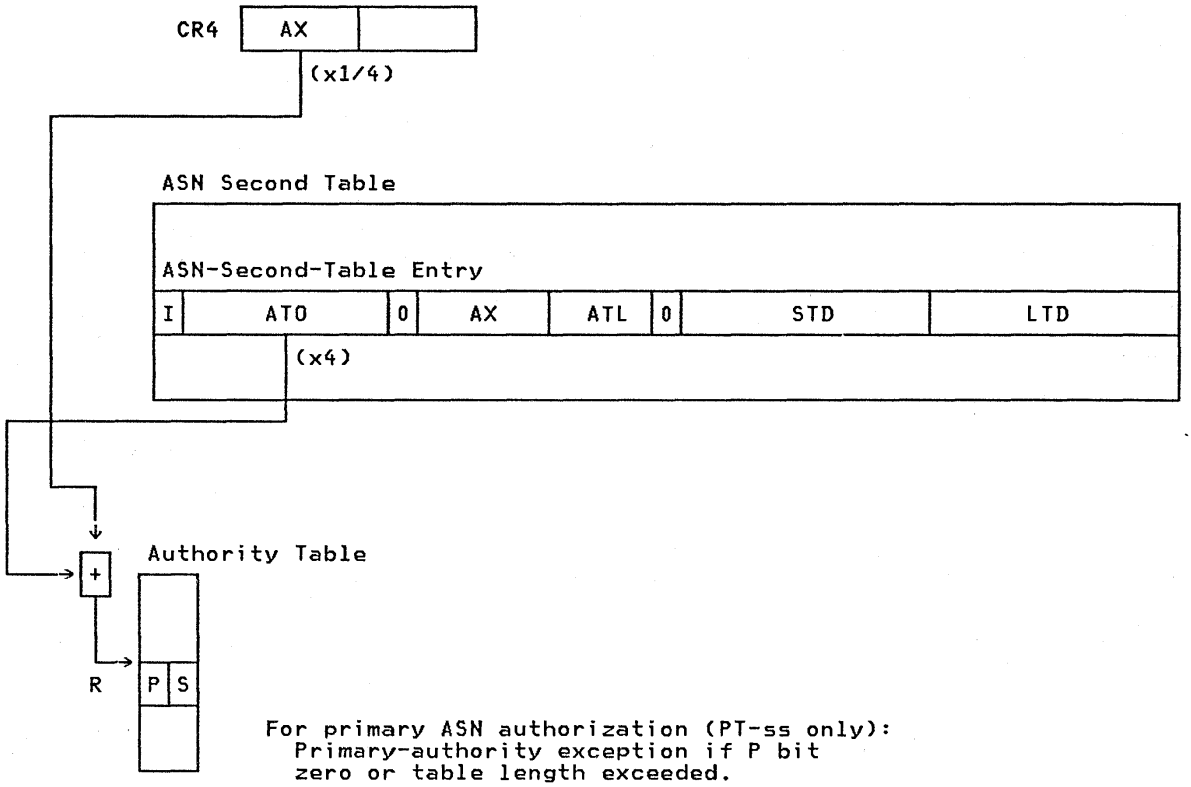
Secondary Authority (S): The right bit of an authority-table entry controls whether the program with the corresponding authorization index is permitted to establish the address space as a secondary address space. If the S bit is one, the access is permitted. If the S bit is zero, the access is not permitted.

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 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, and a secondary-authority exception does not occur. Instead, such a situation 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 secondary-authority bit is examined, depending on whether the primary- or secondary-ASN-authorization process is being performed. The ASN-authorization process is shown in the figure "ASN Authorization."

LzLabs GmbH. Ex. 1016-47



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 1-31 of the AST entry contain 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. A carry, if any, into bit position 0 is ignored. 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 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.

The fetch access to the byte in the authority table is not subject to protection. When the storage address

LzLabs GmbH. Ex. 1016-48

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, 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 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 or secondary-authority exception is recognized for PT-ss or SSAR-ss, respectively. For LASP, 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 occurs in 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 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

LzLabs GmbH. Ex. 1016-49

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 two different address spaces without requiring that the translation parameters in the control registers be changed. These two address spaces are called the primary address space and the secondary address space.

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.

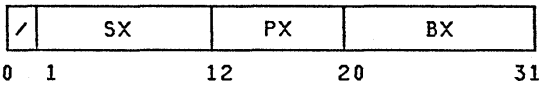
Dynamic address translation may be specified for instruction and data addresses generated by the CPU but is not available for the addressing of data and of CCWs and IDAWs in I/O operations. The CCW-indirect-data-addressing facility is provided to aid I/O operations in a virtual-storage environment.

Address computation can be carried out in either 24- or 31-bit addressing mode. When address computation is performed in 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 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. When DAT is off, the logical address is treated as a real address. When DAT is on, the virtual address may be either a primary virtual address or a secondary virtual address. Primary virtual addresses are translated by means of the primary segment-table designation and secondary virtual addresses by means of the secondary segment-table designation. After selection of the appropriate segment-table designation, the translation process is the same for both types of virtual address.

In the process of translation, two sizes 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 two bits in the PSW and by a set of bits, referred to as the translation parameters, in control registers 0, 1, and 7. Additional controls are located in the translation tables.

PSW

The two bits in the PSW that control dynamic address translation are bit 5, the DAT-mode bit, and bit 16, the address-space-control bit. When the DAT-mode bit is zero, DAT is off, the CPU is said to be in real mode, and instruction and logical addresses are treated as real addresses. When the DAT-mode bit is one (DAT is on) and the address-space-control bit is zero, the CPU is said to be in primary-space mode, and instruction and logical addresses are treated as primary virtual addresses. When DAT is on and the address-space-control bit is one, the CPU is said to be in secondary-space mode, and logical addresses are treated as secondary virtual addresses. The various modes are shown in the figure "Translation Modes."

LzLabs GmbH. Ex. 1016-50

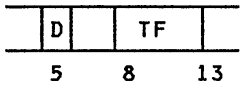
PSW Bit				Handling of Addresses	
5	16	DAT	Mode	Logical Addresses	Instruction Addresses
0	-	Off	Real mode	Real	Real
1	0	On	Primary-space mode	Primary virtual	Primary virtual
1	1	On	Secondary-space mode	Secondary virtual	See note

Translation Modes

Note: When the CPU is in secondary-space mode, it is unpredictable whether instruction addresses are treated as primary virtual or secondary virtual. However, all copies of an instruction used in a single execution are fetched from a single space, and the machine can change the interpretation of instruction addresses as primary virtual or secondary virtual only between instructions and only by issuing a checkpoint-synchronizing function.

Control Register 0

Six bits are provided in control register 0 which are used 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 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.

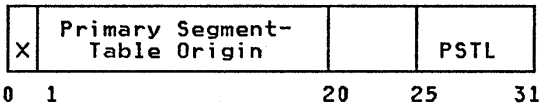
The control bits are encoded as follows:

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:



Space-Switch-Event-Control Bit (X): When bit 0 of control register 1 is one and execution of PROGRAM CALL with space switching (PC-ss) or PROGRAM TRANSFER with space switching (PT-ss) is completed, a space-switch-event program interruption occurs. The space-switch-event-control bit is also examined by LOAD ADDRESS SPACE PARAMETERS, and, if it is one, condition code 3 is set.

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.

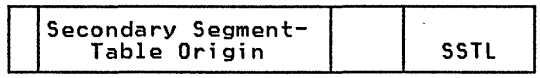
LzLabs GmbH. Ex. 1016-51

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 20-24 of control register 1 are not assigned and are ignored.

Control Register 7

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



Secondary Segment-Table Origin (SSTO): Bits 1-19 of control register 7, with 12 zeros appended on the right, form an address that designates the beginning of the secondary segment table. It is unpredictable whether the address is real or absolute. This table is called the secondary segment table since it is used to translate virtual addresses in the secondary address space.

Secondary Segment-Table Length (SSTL): Bits 25-31 of control register 7 specify the length of the secondary 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 secondary segment table, in units of 64 bytes, is one more than the SSTL value. The contents of the length field are used to establish whether the entry designated by the segment-index portion of a secondary virtual address falls within the secondary segment table.

Bits 0 and 20-24 of control register 7 are not assigned and are ignored.

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, including the case when the PSW specifies the wait state.

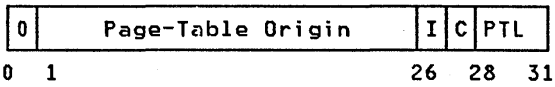
TRANSLATION TABLES

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 designates the origin, availability, and length of the corresponding page table. It also specifies whether the segment is common or private. The segment tables designated by the primary and secondary segment-table designations have the same format.

An entry in 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

LzLabs GmbH. Ex. 1016-52

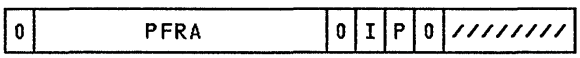
entry and of the page table which it designates. A zero identifies a private segment; in this case, the segment-table entry and the page table it designates may be used only in association with the segment-table origin that designates the segment table in which the segment-table entry resides. A one 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.

Page-Table Length (PTL): Bits 28-31 designate 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 indicates the availability of the page and contains the leftmost bits of the real address. Additionally, the entry contains the page-protection bit. The page-table entry has the following format:



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 unassigned and are not checked for zeros; thus, they are available for programming use.

Summary of Segment-Table and Page-Table Sizes

The sizes of segment tables and page tables are summarized in the figure "Sizes of Segment Tables and Page Tables."

LzLabs GmbH. Ex. 1016-53

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.

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. 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 situations are instead indicated in the condition code. Translation of the operand address of LOAD REAL ADDRESS also differs in that the CPU may be in real mode and 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 bit, both in the PSW. The translation tables are specified by the translation parameters in control registers 1 and 7.

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, control register 1 is used as the effective segment-table designation, and when a secondary virtual address is translated, control register 7 is used as the effective segment-table designation.

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.

LzLabs GmbH. Ex. 1016-54

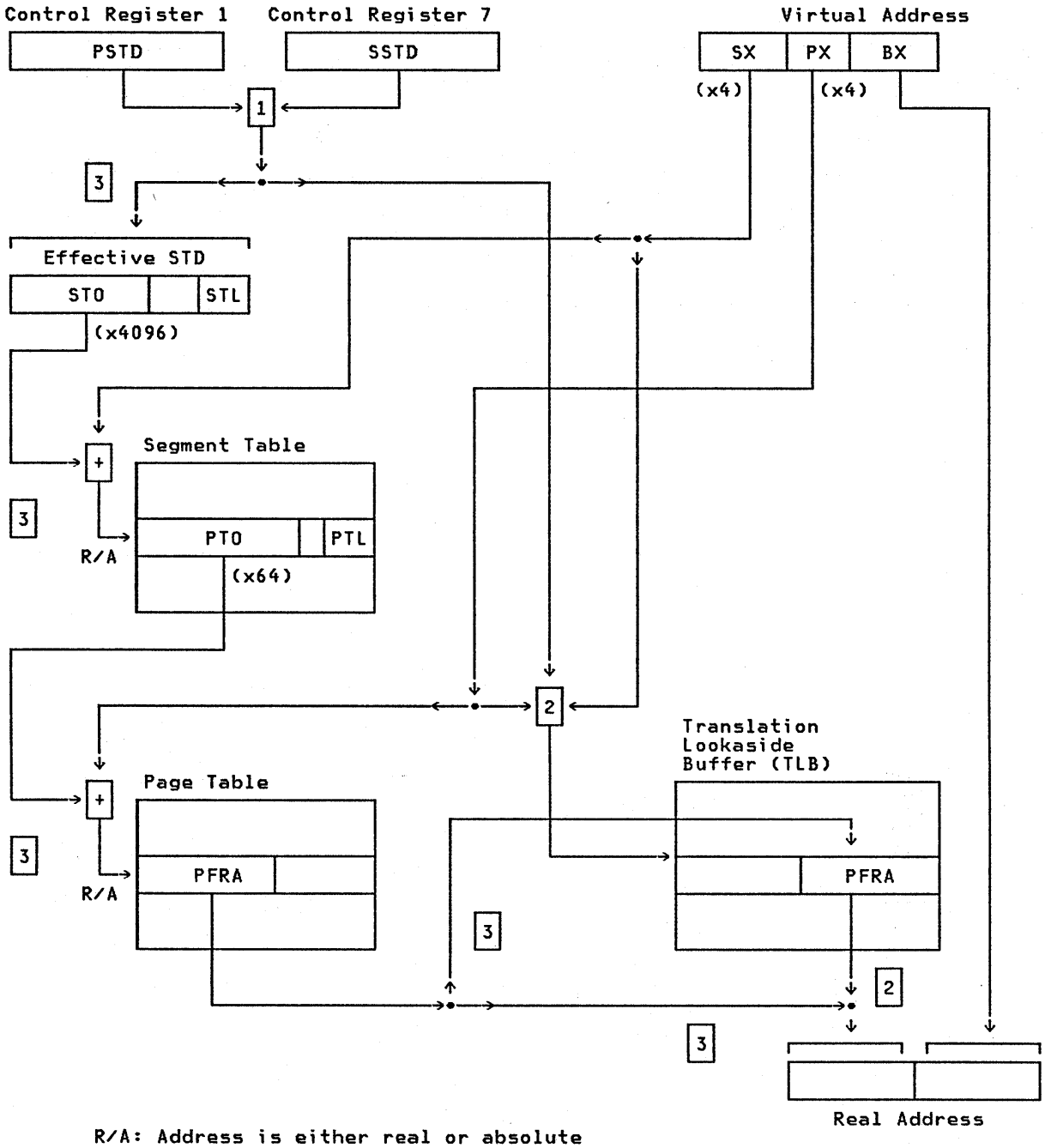
If the I bit is one in either the segment-table entry or the page-table entry, the entry is invalid, and the translation process cannot be completed for this virtual address. A segment-translation or a page-translation exception is recognized.

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 the section "Translation-Lookaside Buffer" in this chapter.

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 the figure "Translation Process."

LzLabs GmbH. Ex. 1016-55



Translation Process (Part 1 of 2)

LzLabs GmbH. Ex. 1016-56

[1] Control register 1 provides the primary segment-table designation for translation of a primary virtual address, and control register 7 provides the secondary segment-table designation for translation of a secondary 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.

Translation Process (Part 2 of 2)

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 real address of the segment-table entry 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, 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.

All four bytes of the segment-table entry are fetched concurrently. 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.

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

The common-segment bit, bit 27 of an entry fetched from the segment table, is used only for the purpose of forming a TLB entry (see the section "Use of the Translation-Lookaside Buffer" later in this chapter).

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 real address of the page-table entry 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

LzLabs GmbH. Ex. 1016-57

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 are fetched concurrently. 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 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, including the case 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 the PSTD or the SSTD are not recognized when the translation of an address uses only the SSTD or only the PSTD, respectively.

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 the section "Recognition of Access Exceptions" in Chapter 6, "Interruptions."

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 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 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.

The size and the structure of the TLB depend on the model. For instance, the TLB may be implemented in such a way as to contain only a few entries pertaining to the currently designated segment table, each entry consisting of the leftmost portion of a virtual address and its corresponding page-frame real address and page-protection bit; or it may contain arrays of values where the page-frame real address and page-protection bit are selected on the basis of the effective segment-table origin and the leftmost bits of the virtual address. Entries within the TLB are not explicitly addressable by the program.

The description of the logical structure of the TLB covers the implementation by all systems operating in the 370-XA

LzLabs GmbH. Ex. 1016-58

mode. 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. Thus, a TLB segment-table entry would contain the following fields:

STO	SX	PTO	PTL	C

STO The segment-table origin in effect when the entry was formed

SX The segment index used to select the entry

PTO The page-table origin fetched from the segment-table entry in real or absolute storage

PTL The page-table length fetched from the segment-table entry in real or absolute storage

C The common-segment bit fetched from the segment-table entry in real or absolute storage

A TLB page-table entry would contain the following fields:

PTO	PX	PFRA	P

PTO The page-table origin in effect when the entry was formed

PX The page index used to select the entry

PFRA The page-frame real address fetched from the page-table entry in real or absolute storage.

P The page-protection bit fetched from the page-table entry in real or absolute storage

Depending on the implementation, not all of the above items are required in the TLB. For example, if the implementation combines into a single TLB entry (1) the information obtained from a page-table entry and (2) the attributes of both the page-table entry and the segment-table entry, then the page-table-origin and page-table-length fields are not required.

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

Use of the Translation-Lookaside Buffer

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 valid, on whether the entry is attached to a particular CPU, on whether a copy of the entry can be placed in the TLB of a particular CPU, and on whether a copy in the TLB of the entry is usable.

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.

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. The table entry may be attached to more than one CPU at a time. When a table entry is described as attached, the term "to a CPU" is implied.

The usable state of a TLB entry denotes that the CPU can attempt to use the TLB entry for implicit address translation.

A segment-table entry or a page-table entry may be placed in the TLB only when the entry is attached and valid and would not cause a translation-specification exception if used for translation. Except for these restrictions, the entry may be placed in the TLB at any time.

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 which would cause an early 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 or b below.
   a. The entry is within the segment table specified by the primary segment-table designation in control register 1.
   b. The entry is within the segment table specified by the second-

LzLabs GmbH. Ex. 1016-59

ary segment-table designation in control register 7 and either of the following requirements is met:

- The CPU is in secondary-space mode.
- The secondary-space control, bit 5 of control register 0, is one.

A page-table entry is attached when it is within the page table designated by either a usable TLB segment-table entry or by an attached and valid segment-table entry which would not cause a translation-specification exception if used for 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 which would cause an early 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:
   - The common-segment bit is one in the TLB entry.
   - The segment-table-origin field in the TLB entry is the same as the current PSTO.
   - The segment-table-origin field in the TLB entry is the same as the current SSTO, and either PSW bit 16 is one or bit 5 of control register 0 is one.

A TLB segment-table entry may be used for implicit address translation only when the entry is in the usable state, the segment index of the entry matches the segment index of the virtual address to be translated, and 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 used to select it.

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

1. The TLB page-table entry is selected by a usable TLB segment-table entry or by an attached and valid segment-table entry which would not cause a translation-specification exception if used for translation.
2. The page-table-origin field in the TLB page-table entry matches the page-table-origin field in the segment-table entry which selects it.
3. 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 which selects it.

A TLB page-table entry may be used for implicit address translation only when the TLB entry is in the usable state as selected by the segment-table entry being used and only when the page index of the TLB page-table entry matches the page index of the virtual address being translated.

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.

Selected page-table entries are cleared from the TLB by means of the INVALIDATE PAGE TABLE ENTRY instruction. All information in the TLB is necessarily cleared only by execution of PURGE TLB, SET PREFIX, or CPU reset.

Programming Notes

1. Although a table entry may be copied into the TLB only when the table entry is both valid and attached, the copy may remain in the TLB even when the table entry itself is no longer valid or attached.
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-table entries that could be used for address translation, given the current translation parameters, the setting of the address-space-control bit, and the setting of the secondary-space-control bit. 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 wait state is specified.
3. More than one copy of a table entry may exist in the TLB. For example,

LzLabs GmbH. Ex. 1016-60

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.

4. The states and use of the DAT entries in both storage and in the TLB are summarized in the figure "Summary of DAT Entries."

State or Function	Conditions to Be Met
STE is attached by means of PSTD (applies only to STE in storage)	• DAT on • No early PSW exception • TF valid • STE in segment table defined by PSTD in CR1
STE is attached by means of SSTD (applies only to STE in storage)	• DAT on • No early PSW exception • TF valid • STE in segment table defined by SSTD in CR7 • PSW bit 16 one or bit 5 of CR0 one
STE in storage is usable for a particular instance of implicit translation	• STE in segment table defined and attached by STD being used for the translation • STE selected by SX
STE can be placed in TLB	• STE attached • STE I bit zero • No TS
STE in TLB is usable	• DAT on • No early PSW exception • TF valid • STE selectable by an STD: - C bit one, or - STO matches PSTO, or - STO matches SSTO, and PSW bit 16 one or bit 5 of CR0 one
STE in TLB is usable for a particular instance of implicit translation	• DAT on • No early PSW exception • TF valid • STE selected by STD being used for the translation: - STO matches, or - C bit one • SX matches
PTE is attached (applies only to PTE in storage)	• PTE in page table defined by usable STE in the TLB, or defined by an STE that can be placed in the TLB
PTE in storage is usable for a particular instance of implicit translation	• PTE attached by means of STE being used for the translation • PTE selected by PX

Summary of DAT Entries (Part 1 of 2)

LzLabs GmbH. Ex. 1016-61

State or Function	Conditions to Be Met
PTE can be placed in TLB	• PTE attached • PTE I bit zero • No TS
PTE in TLB is usable	• PTE selectable by a usable STE in the TLB or by an STE that can be placed in the TLB: - PTO matches and - PX within PTL
PTE in TLB is usable for a particular instance of implicit translation	• PTE selected by STE being used for the translation: - PTO matches and - PX within PTL • PX matches

Explanation:

C bit	Common-segment bit in STE
I bit	Invalid bit in table entry
PSTD	Primary segment-table designation
PSTO	Primary segment-table origin
PTE	Page-table entry
PTL	Page-table length
PTO	Page-table origin
PX	Page index
SSTD	Secondary segment-table designation
SSTO	Secondary segment-table origin
STD	Segment-table designation
STE	Segment-table entry
STO	Segment-table origin
SX	Segment index
TF	Translation format (control register 0, bits 8-12)
TS	Translation-specification exception

Summary of DAT Entries (Part 2 of 2)

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 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, 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 which 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 which 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. 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.

LzLabs GmbH. Ex. 1016-62

3. Some or all TLB entries may be cleared at times other than those required by PURGE TLB, SET PREFIX, CPU reset, and INVALIDATE PAGE TABLE ENTRY.

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 which 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 the 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.

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, then 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 following INVALIDATE PAGE TABLE ENTRY.)

   a. INVALIDATE PAGE TABLE ENTRY should be issued 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

LzLabs GmbH. Ex. 1016-63

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 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 observed, translation is performed as if the table entries from real storage were always used in the translation process.

   a. A valid table entry must not be changed while it is attached to any CPU except either to invalidate the entry by using INVALIDATE PAGE TABLE ENTRY or to alter bits 24-31 of a page-table entry.

   b. When any change is made to a 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 issue PURGE TLB or SET PREFIX or perform CPU reset, after the change occurs and prior to the use of that entry for implicit translation by that CPU, except that the purge is unnecessary if the change was made by using INVALIDATE PAGE TABLE ENTRY.

   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 issue PURGE TLB or SET PREFIX or perform CPU reset, 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 segment-table or page-table length, each CPU to which that table has been attached must issue PTLB after the length has been changed but before that table becomes attached again to the 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 usable TLB copies for 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 usable TLB copies for that segment-table entry and which does not have any usable TLB copies for the page-table entries attached by it.

   The execution of PURGE TLB and SET PREFIX may have an adverse effect on the performance of some models. Use of these instructions should, therefore, be minimized in conformity 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 for an instruction or an operand are 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 logical. 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 designated in the B and D fields of an instruction but that are not used to address storage. This includes operand addresses in LOAD ADDRESS, 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 the section

LzLabs GmbH. Ex. 1016-64

"Address Types" in this chapter. 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 the section "Prefixing" in this chapter.

HANDLING OF ADDRESSES

The handling of addresses is summarized in the figure "Handling of Addresses." This figure lists all addresses that are encountered by the program and specifies the address type.

Virtual Addresses

- Operand address in INSERT VIRTUAL STORAGE KEY
- Operand address in LOAD REAL ADDRESS
- Operand addresses in 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

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, and PROGRAM CALL

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 and COMPARE LOGICAL LONG
- Address for TEST PENDING INTERRUPTION provided the address is nonzero

Real Addresses

- Operand address in INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY EXTENDED, and TEST BLOCK
- Page-table origin in INVALIDATE PAGE TABLE ENTRY
- Segment-table origin[1] in control registers 1 and 7
- Page-table origin[1] in segment-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
- Linkage-table origin in control register 5
- Entry-table origin in linkage-table entry

Handling of Addresses (Part 1 of 2)

LzLabs GmbH. Ex. 1016-65

- The translated address generated by LOAD REAL ADDRESS
- Address[1] of segment-table entry or page-table entry provided by LOAD REAL ADDRESS

Permanently Assigned Real Addresses

- Address of the doubleword at location 184 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

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 in SET CHANNEL MONITOR
- Address limit specified in 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
- Parameter stored in the word at real location 128 for a service-signal external interruption
- I/O interruption parameter stored in the word at real location 188 for an I/O interruption

[1] It is unpredictable whether these addresses are treated as real or absolute.

Handling of Addresses (Part 2 of 2)

ASSIGNED STORAGE LOCATIONS

The figure "Assigned Storage Locations" 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)

LzLabs GmbH. Ex. 1016-66

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 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, 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. 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 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

LzLabs GmbH. Ex. 1016-67

the remaining bit positions of 141.

144-147 (Real Address)

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. The rightmost 12 bits of the address are unpredictable. Bit 0 of location 144 is set to zero if the translation was relative to the primary segment table designated by control register 1, or it is set to one if the translation was relative to the secondary segment table designated by control register 7.

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. Locations 144-145 are set to zeros.

During a program interruption for a space-switch event, the old PASN, which appears in bits 16-31 of control register 4 before the execution of a space-switching PROGRAM CALL or PROGRAM TRANSFER instruction, is stored at locations 146-147. The old space-switch-event-control bit is placed in bit position 0, and zeros are placed in bit positions 1-15 at locations 144-145.

During a program interruption due to an LX-translation or EX-translation exception, the PC number is stored in bit positions 12-31 of the word at location 144. Bits 0-11 are set to zeros.

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, the PER code is stored in bit positions 0-3 of location 150, and zeros are stored in bit positions 4-7 and at location 151.

152-155 (Real Address)

PER Address: During a program interruption due to a program 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 in the word at location 156.

184-187 (Real Address)

Subsystem-Identification Word: During an I/O interruption, the subsystem-identification word is stored in these locations.

188-191 (Real Address)

Interruption Parameter: During an I/O interruption, the interruption parameter from the associated subchannel is stored in these locations.

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 location 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 location 224-231.

232-239 (Real Address)

Machine-Check-Interruption Code: During a machine-check interruption the machine-check-interrup-

LzLabs GmbH. Ex. 1016-68

tion code is stored at locations 232-239.

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.

256-271 (Real Address)

Fixed-Logout Area: Depending on the model, logout information may be placed in this area 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 location 264-267.

352-383 (Absolute Address)

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

384-447 (Absolute Address)

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

LzLabs GmbH. Ex. 1016-69

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	

Assigned Storage Locations (Part 1 of 3)

LzLabs GmbH. Ex. 1016-70

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	Translation-Exception Identification			
94	148	Monitor-Class Number		PER Cde	0 0 0 0 0 0 0 0 0 0 0 0
98	152	PER Address			
9C	156	Monitor Code			
A0	160				
A4	164				
A8	168				
AC	172				
B0	176				
B4	180				
B8	184	Subsystem-Identification Word			
BC	188	I/O-Interruption Parameter			
C0	192				
C4	196				
C8	200				
CC	204				
D0	208				
D4	212				
D8 DC	216 220	Store-Status CPU-Timer Save Area; or Machine-Check CPU-Timer Save Area			
E0 E4	224 228	Store-Status Clock-Comparator Save Area; or Machine-Check Clock-Comparator Save Area			
E8 EC	232 236	Machine-Check Interruption Code			
F0 F4	240 244				
F8	248	Failing-Storage Address			
FC	252				

Assigned Storage Locations (Part 2 of 3)

LzLabs GmbH. Ex. 1016-71

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	
	/	/
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	
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	

Assigned Storage Locations (Part 3 of 3)

LzLabs GmbH. Ex. 1016-72

Stopped, Operating, Load, and Check-Stop States 4-2
Stopped State 4-2
Operating State 4-2
Load State 4-3
Check-Stop State 4-3
Program-Status Word 4-3
Program-Status-Word Format 4-5
Control Registers 4-6
Tracing 4-8
Control-Register Allocation 4-8
Trace Entries 4-9
Operation 4-10
Program-Event Recording 4-11
Control-Register Allocation 4-11
Operation 4-12
Identification of Cause 4-12
Priority of Indication 4-13
Storage-Area Designation 4-14
PER Events 4-14
Successful Branching 4-14
Instruction Fetching 4-14
Storage Alteration 4-15
General-Register Alteration 4-15
Indication of Events Concurrently with Other Interruption Conditions 4-16
Timing 4-18
Time-of-Day Clock 4-18
Format 4-18
States 4-19
Changes in Clock State 4-19
Setting and Inspecting the Clock 4-20
TOD-Clock Synchronization 4-21
Clock Comparator 4-21
CPU Timer 4-22
Externally Initiated Functions 4-23
Service Signal 4-23
Resets 4-24
CPU Reset 4-27
Initial CPU Reset 4-27
Subsystem Reset 4-28
Clear Reset 4-28
Power-On Reset 4-28
Initial Program Loading 4-29
Store Status 4-29
Multiprocessing 4-30
Shared Main Storage 4-30
CPU-Address Identification 4-30
CPU Signaling and Response 4-31
Signal-Processor Orders 4-31
Conditions Determining Response 4-33
Conditions Precluding Interpretation of the Order Code 4-33
Status Bits 4-34

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

LzLabs GmbH. Ex. 1016-73

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.

STOPPED STATE

The state of the CPU is changed from operating to stopped by 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 are taken 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:

- The CPU reset is completed. However, when the reset operation is performed as part of initial program loading, 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 the section "Resets" in this chapter, and address comparison is described in the section "Address-Compare Controls" in Chapter 12, "Operator Facilities."

If the CPU is in the stopped state when an INVALIDATE PAGE TABLE ENTRY instruction is executed on another CPU in the configuration, the invalidation may be performed immediately or may be delayed until the CPU leaves the stopped state.

OPERATING STATE

The state of the CPU is changed from stopped to operating when the start function is performed or when a restart interruption (see Chapter 6) 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 are taken before the CPU returns to the stopped state. When the

LzLabs GmbH. Ex. 1016-74

rate control is set to the instruction-step position and the wait-state bit is one, the start function causes no instruction to be executed, but all pending allowed interruptions are taken 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 the section "Initial Program Loading" in this chapter. See also the section "Initial Program Loading" in Chapter 17, "I/O Support Functions.") If the initial-program-loading operation is completed successfully, the CPU state changes from load to operating, provided the rate control is set to the process position; if the rate control is set to the instruction-step position, the CPU state changes from load to stopped.

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. The figure "Operations on PSW Fields" 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. 1016-75

Instruction	System Mask (PSW Bits 0-7)		PSW Key (PSW Bits 8-11)		Problem State (PSW Bit 15)		Address-Space Control (PSW Bit 16)		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	No	No	No	No	No	No	No	No	AM	No	AM	No
BRANCH AND SAVE	No	No	No	No	No	No	No	No	No	No	Yes	No
BRANCH AND SAVE AND SET MODE	No	No	No	No	No	No	No	No	No	No	Yes	Yes[2]
BRANCH AND SET MODE	No	No	No	No	No	No	No	No	No	No	Yes[1]	Yes[2]
INSERT PROGRAM MASK	No	No	No	No	No	No	No	No	Yes	No	No	No
INSERT PSW KEY	No	No	Yes	No	No	No	No	No	No	No	No	No
INSERT ADDRESS SPACE CONTROL	No	No	No	No	No	No	Yes	No	No	No	No	No
PROGRAM CALL	No	No	No	No	Yes	Yes	No	No	No	No	Yes	Yes
PROGRAM TRANSFER	No	No	No	No	No	Yes[1]	No	No	No	No	No	Yes
SET ADDRESS SPACE CONTROL	No	No	No	No	No	No	No	Yes	No	No	No	No
SET PROGRAM MASK	No	No	No	No	No	No	No	No	No	Yes	No	No
SET PSW KEY FROM ADDRESS	No	No	No	Yes	No	No	No	No	No	No	No	No
SET SYSTEM MASK	No	Yes	No	No	No	No	No	No	No	No	No	No
STORE THEN AND SYSTEM MASK	Yes	ANDs	No	No	No	No	No	No	No	No	No	No
STORE THEN OR SYSTEM MASK	Yes	ORs	No	No	No	No	No	No	No	No	No	No

Explanation:

[1] Cannot be changed from one to zero.

[2] The action takes place only if the associated R field in the instruction is nonzero.

AM The action depends on the addressing mode, bit 32 of the current PSW. In 24-bit mode, the condition code and program mask are saved in the leftmost byte of the general register. In 31-bit addressing mode, the addressing mode, along with bits 1-7 of the 31-bit address, replace the leftmost byte of the register.

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.

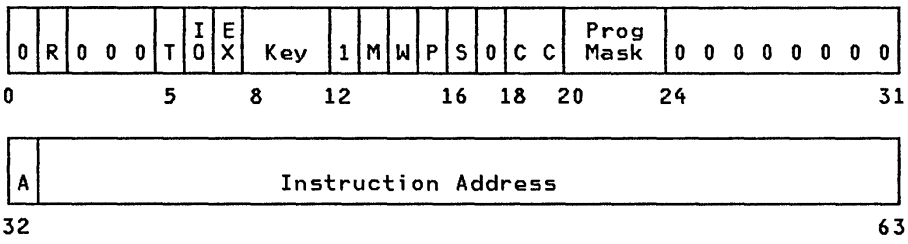
Operations on PSW Fields

Programming Note

A summary of the operations which save or set the problem state, addressing mode, and instruction address is contained in the section "Subroutine Linkage" in Chapter 5, "Program Execution."

LzLabs GmbH. Ex. 1016-76

PROGRAM-STATUS-WORD FORMAT



PSW Format

The following is a summary of the functions of the PSW fields. (See the figure "PSW Format.")

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 is zero, an I/O interruption for that I/O-interruption subclass cannot occur; when the I/O-interruption-subclass mask 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 accesses to the second operand of MOVE TO PRIMARY and MOVE WITH KEY, the third operand is used instead of the PSW key. The third operand is also used instead of the PSW key for accesses to the first operand of MOVE TO SECONDARY.

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.

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. 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, and a program interruption takes place. Another group of instructions, called semiprivileged instructions, are only executed by a CPU

LzLabs GmbH. Ex. 1016-77

in the problem state if specific authority tests are met; otherwise, a privileged-operation exception or a special-operation exception is recognized, and a program interruption takes place.

Address-Space Control (S): Bit 16, in conjunction with PSW bit 5, controls the address-space mode. See the discussion of the PSW under "Translation Control" in Chapter 3, "Storage."

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	Exponent underflow
23	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 exponent-underflow-mask bit or the 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, linkage, entry, and trace tables. See the section "Address Generation" in Chapter 5, "Program Execution," and the section "Address Size and Wraparound" in Chapter 3, "Storage."

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, 17, 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 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 register positions, within those registers designated by the instruction, to be loaded from storage. The instructions LOAD ADDRESS SPACE PARAMETERS, SET SECONDARY ASN, PROGRAM CALL, and PROGRAM TRANSFER offer specialized control over which register positions are loaded and the source of the information to be loaded. Information loaded into the control registers becomes active (that is, assumes control over the system) at the completion of the instruction causing 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 a 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 all register positions, within those registers designated by the instruction, to be placed in storage. The instructions EXTRACT PRIMARY ASN, EXTRACT SECONDARY ASN, and PROGRAM CALL offer specialized control over which register positions are to be loaded into a general register.

Only the general structure of control registers is described here; a definition of the register positions appears with the description of the facility with which the register position is associated. The figure "Assignment of Control-Register Fields" shows the control-register positions which are

LzLabs GmbH. Ex. 1016-78

assigned and the initial value of the field upon execution of initial CPU reset. All control-register positions not listed in the figure are initialized to zero.

Programming Note

To ensure that existing programs run if and when new facilities using additional control-register positions are installed, the program should load zeros in unassigned control-register positions.

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	Dual-address-space control	0
0	5	Secondary-space control	Dual-address-space control	0
0	6	Fetch-protection override	Key-controlled protection	0
0	8-12	Translation format	Dynamic address translation	0
0	16	Malfunction-alert subclass mask	Interruptions	0
0	17	Emergency-signal subclass mask	Interruptions	0
0	18	External-call subclass mask	Interruptions	0
0	19	TOD-clock sync-check subclass mask	Interruptions	0
0	20	Clock-comparator subclass mask	Clock comparator	0
0	21	CPU-timer subclass mask	CPU timer	0
0	22	Service-signal subclass mask	Service signal	0
0	24	Unused[1]		1
0	25	Interrupt-key subclass mask	External interruptions	1
0	26	Unused[1]		1
0	30	IUCV subclass mask[2]	Virtual machines	0
0	31	VMCF subclass mask[2]	Virtual machines	0
1	0	Space-switch-event control	Dual-address-space control	0
1	1-19	Primary segment-table origin	Dynamic address translation	0
1	25-31	Primary segment-table length	Dynamic address translation	0
3	0-15	PSW-key mask	Dual-address-space control	0
3	16-31	Secondary ASN	Dual-address-space control	0
4	0-15	Authorization index	Dual-address-space control	0
4	16-31	Primary ASN	Dual-address-space control	0
5	0	Subsystem-linkage control	Dual-address-space control	0
5	1-24	Linkage-table origin	PC-number translation	0
5	25-31	Linkage-table length	PC-number translation	0
6	0-7	I/O-interruption-subclass mask	I/O	0
7	1-19	Secondary segment-table origin	Dynamic address translation	0
7	25-31	Secondary segment-table length	Dynamic address 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 recording	0
9	16-31	PER general-register masks	Program-event recording	0
10	1-31	PER starting address	Program-event recording	0
11	1-31	PER ending address	Program-event recording	0

Assignment of Control-Register Fields (Part 1 of 2)

LzLabs GmbH. Ex. 1016-79

Ctrl Reg	Bits	Name of Field	Associated with	Initial Value
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
14	0	Unused[1]		1
14	1	Unused[1]		1
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	Timing-facility-damage subclass mask	Machine-check handling	1
14	7	Warning subclass mask	Machine-check handling	0
14	12	ASN-translation control	ASN translation	0
14	13-31	ASN-first-table origin	ASN translation	0

Explanation:

The fields not listed are unassigned.

[1] This bit is not used but is initialized to one for consistency with the System/370 definition.

[2] This bit is used only in a control register of a virtual machine; in a real machine, this bit is reserved.

Assignment of Control-Register Fields (Part 2 of 2)

TRACING

Tracing assists in the determination of system problems by providing an on-going 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

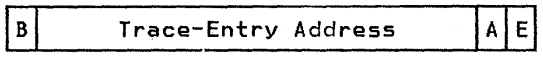
When ASN tracing is on, an entry is made in the trace table for each execution of the following instructions:

- PROGRAM CALL
- PROGRAM TRANSFER
- SET SECONDARY ASN

When explicit tracing is on, execution of TRACE causes 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 this format:



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.

LzLabs GmbH. Ex. 1016-80

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 six types, as shown in the figure "Trace-Entry Formats."

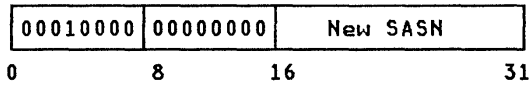
31-Bit Branch



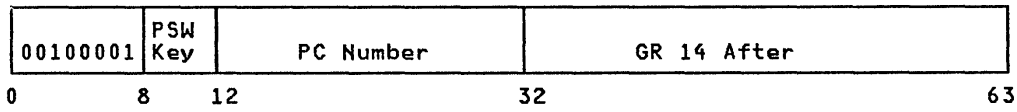
24-Bit Branch



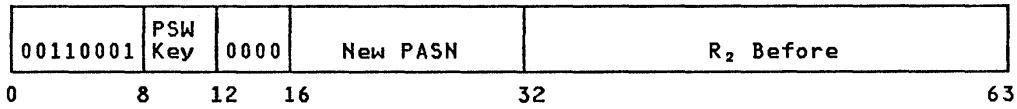
SET SECONDARY ASN



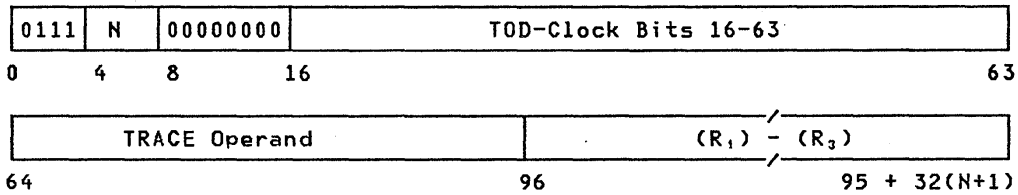
PROGRAM CALL



PROGRAM TRANSFER



TRACE



Trace-Entry Formats

LzLabs GmbH. Ex. 1016-81

Branch Address: The branch address is the address of the next instruction to be executed when the branch is taken. When the 31-bit addressing mode is in effect after branching, bit positions 1-31 of the trace entry for a branch instruction contain the branch address. When the 24-bit addressing mode is in effect after branching, bit positions 8-31 contain the branch address.

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 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.

GR14 After: Bit positions 32-63 of the trace entry made on execution of PROGRAM CALL contain the information which is placed in general register 14: the addressing bit, the return address, and the problem-state bit.

New PASN: Bit positions 16-31 of the trace entry made on execution of PROGRAM TRANSFER contain the new PASN (which may be zero) specified by the instruction.

R2 Before: Bit positions 32-63 of the trace entry made on execution of PROGRAM TRANSFER contain the contents of the general register specified by the R_2 field of the instruction. Bits 0-30 of the general register specified 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 have been provided by a STORE CLOCK instruction executed at the time the TRACE instruction was executed.

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.

(R1)-(R3): The four-byte fields starting with bit 96 of the trace entry for TRACE contain the contents of the general registers whose range is designated 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 the register specified by R_1 and continuing up to and including the register specified by R_3, with register 0 following 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, 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, and instruction execution is nullified. 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 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.

LzLabs GmbH. Ex. 1016-82

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.

The order in which information is placed in a trace entry is undetermined. Furthermore, as observed by other CPU or channel programs, the contents of a byte of a trace entry may 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 at the beginning and also at the completion of the tracing operation.

The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results.

When the tracing operation is completed, a second serialization and checkpoint-synchronization function is performed, as follows. The CPU operation is delayed until all storage accesses due to this instruction have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, for this instruction are canceled, and the results of all stores for this instruction are released, if held exclusive, to permit other CPU and channel programs to access the results.

PROGRAM-EVENT RECORDING

The purpose of the program-event-recording (PER) facility is to assist in debugging programs. It permits the program to be alerted to the following types of PER events:

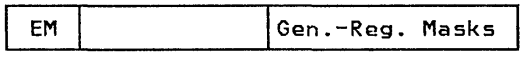
- Execution of a successful branch instruction.
- Fetching of an instruction from the designated storage area.
- Alteration of the contents of the designated storage area.
- Alteration of the contents of designated general registers.

The program can selectively specify that one or more of the above types of events be recognized. 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.

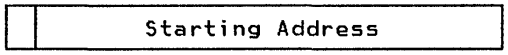
CONTROL-REGISTER ALLOCATION

The information for controlling PER resides in control registers 9, 10, and 11 and consists of the following fields:

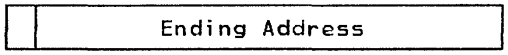
Control Register 9:



Control Register 10:



Control Register 11:



PER-Event Masks (EM): Bits 0-3 of control register 9 specify which types of events are recognized. 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

Bits 0-3, when ones, specify that the corresponding types of events be recognized. When a bit is zero, the corresponding type of event is not recognized.

PER General-Register Masks: 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.

LzLabs GmbH. Ex. 1016-83

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.

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.

Programming Note

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.

However, 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, a particular general-register mask bit are all ones, the CPU is enabled for the corresponding type of event; otherwise, it is disabled.

An interruption due to a PER event is normally taken 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 either completed, partially completed, terminated, suppressed, or nullified.

When the CPU is disabled for a particular PER event at the time it occurs, either by the PER 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. 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.

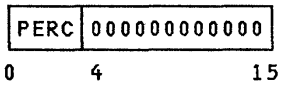
PER events may be recognized in a trial execution of an instruction, and subsequently the instruction, DAT-table entries, and operands may be refetched for the actual execution. If any refetched field was modified by another CPU or by a channel program between the trial and actual executions, it is unpredictable whether the PER events indicated are for the trial execution 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.

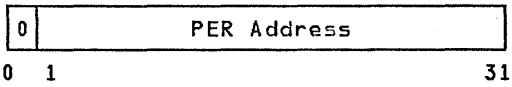
Identification of Cause

A program interruption for PER sets bit 8 of the interruption code to one and places identifying information in real storage locations 150-155. The format of the information stored is as follows:

Locations 150-151:



Locations 152-155:



PER Code (PERC): The occurrence of PER events is indicated by ones in bit positions 0-3 of real location 150, the PER code. 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. 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. Zeros are stored in bit positions 4-7 of location 150.

PER Address: The PER address 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

LzLabs GmbH. Ex. 1016-84

the EXECUTE instruction is placed in the PER-address field. Zeros are stored in bit position 0 of real location 152.

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 or a supervisor-call interruption sets an ILC of 0.

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 the section "Exceptions Associated with the PSW" in Chapter 6, "Interruptions"), both the specification exception and PER are indicated concurrently in the interruption code of the program interruption. However, for a PSW-format error 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 execution of an instruction. However, in the following cases, execution of an interruptible instruction is not completed normally:

- 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 are taken subsequently (subject to the mask bits in the new PSW) in the normal priority order.
- When the stop function is performed, a program interruption indicating the PER event occurs before the stopped state is entered.
- When any program exception is recognized, PER events recognized for that instruction execution are indicated concurrently.
- 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, or without the stopped state being entered.

Programming Notes

1. In the following cases, an instruction can both cause a program interruption for a PER event and change the value of masks controlling an interruption for PER events. The original mask 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 which also changed 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.

2. No instructions can both change the values of general-register-alteration masks 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

LzLabs GmbH. Ex. 1016-85

occurs as a result of LOAD PSW or SUPERVISOR CALL, the ILC indicates the length of these instructions or EXECUTE, as appropriate, unless a concurrent specification exception on LOAD PSW 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 -- involve the designation of an area in storage. 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. A storage-alteration event does not occur for a store access made with an operand address defined to be a real address.

The set of addresses designated for 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 storage-alteration and instruction-fetching events is always performed by using 31-bit addresses. This is accomplished in 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 degradation due to address-range checking is avoided, for those pages not in the designated storage area, by means of an extension to each page-table entry in the TLB. In such an implementation, each page-table entry in the TLB is marked as to whether the corresponding page does or does not contain locations in the designated storage area. Loading the contents of control registers 10 and 11 when the instruction-fetching or storage-alteration event mask is one, or setting either 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.

PER EVENTS

Successful Branching

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 MODE (BSM)
BRANCH ON CONDITION (BC, BCR)
BRANCH ON COUNT (BCT, BCTR)
BRANCH ON INDEX HIGH (BXH)
BRANCH ON INDEX LOW OR EQUAL (BXLE)

A successful-branching event also occurs whenever one of the following instructions is completed:

PROGRAM CALL (PC)
PROGRAM TRANSFER (PT)

A successful-branching event causes a successful-branching PER event to be recognized if bit 0 of the PER-event masks is one and the PER mask in the PSW is one.

A successful-branching PER 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

LzLabs GmbH. Ex. 1016-86

fetched from the storage area designated by control register 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 an instruction-fetching PER event to be recognized if bit 1 of the PER-event masks is one and the PER mask in the PSW is one.

The instruction-fetching PER 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 monitored storage area.

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 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 storing of data by a channel program. (See the section "Recognition of Access Exceptions" in Chapter 6, "Interruptions.") Storing constitutes alteration for PER purposes even if the value stored is the same as the original value.

Implied locations, which are referred to by the CPU in the process of performing an interruption, are not monitored. These locations, however, are monitored when information is stored there explicitly by an instruction. Similarly, monitoring does not apply to storing of data by a channel program.

The I/O instructions are considered to alter the second-operand location only when storing actually occurs.

Storage alteration does not apply to instructions whose operands are specified to be real addresses. Thus, storage alteration does not apply to INVALIDATE PAGE TABLE ENTRY, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY EXTENDED, TEST BLOCK, and TEST PENDING INTERRUPTION (when the effective address is zero).

A storage-alteration event causes a storage-alteration PER event to be recognized if bit 2 of the PER-event mask is one and the PER mask in the PSW is one.

A storage-alteration PER event is indicated by setting bit 2 of the PER code to one.

General-Register Alteration

A general-register-alteration event occurs whenever the contents of a general register are replaced.

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 EDIT AND MARK and TRANSLATE AND TEST are considered to have altered the contents of general register 1 or 2 only when these instructions have caused information to be placed in the register.

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 INSERT CHARACTERS UNDER MASK is not considered to alter the general register when the mask is zero.

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.

A general-register-alteration event causes a general-register-alteration PER 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. It is indicated by setting bit 3 of the PER code to one.

The general-register-alteration PER event is indicated by setting bit 3 of the PER code to one.

LzLabs GmbH. Ex. 1016-87

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.

INDICATION OF 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. The event, however, is not indicated when an access exception prohibits access to the first halfword of the instruction. When the first halfword of the instruction is accessible but an access exception applies to the second or third halfword of the instruction, it is unpredictable whether the instruction-fetching event is indicated.

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, and general-register alteration 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, 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.

5. When LOAD PSW 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. When these instructions introduce a PSW-format error of the type that is recognized as part of the execution of the following instruction, the PSW is stored as the old PSW without the specification exception being recognized.

The indication of PER events concurrently with other program-interruption conditions is summarized in the figure "Indication of PER Events with Other Concurrent Conditions."

LzLabs GmbH. Ex. 1016-88

Concurrent Condition	Type of Ending	PER Event			
		Branch	Instr Fetch	Storage Alter.	GR Alter.
Specification					
Odd instruction address in the PSW	S	No	No	No	No
Instruction access					
First halfword	N or S	No	No	No	No
Second, third halfwords	N or S	No	U	No	No
Specification					
EXECUTE target address odd	S	No	U	No	No
EXECUTE target access	N or S	No	U	No	No
Other nullifying	N	No	Yes	No[1]	No[1]
Other suppressing	S	No	Yes	No[1]	No[1]
All terminating	T	No	Yes	Yes[2]	Yes[2]
All completing	C	Yes	Yes	Yes	Yes

Explanation:

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 contents of the designated storage location or general register were altered, or an attempt was made to execute an instruction whose first byte is located in the designated storage area.

No The PER event is not indicated.

U It is unpredictable whether the PER event is indicated.

[1] 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.

[2] 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.

Indication of PER Events with Other Concurrent Conditions

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.

LzLabs GmbH. Ex. 1016-89

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.

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. However, each CPU has a separate clock comparator and CPU timer.

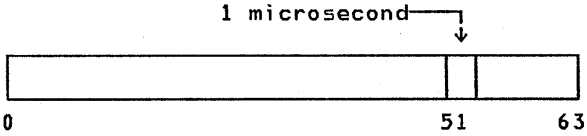
TIME-OF-DAY CLOCK

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 an installation 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 TOD clock is a binary counter with the format shown in the following illustration. The bit positions of the clock are numbered 0 to 63, corresponding to the bit positions of a 64-bit unsigned binary integer.



In the basic form, the TOD clock 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. Thus, it is possible for a single TOD clock to be in more than one 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, the stepping rates are synchronized such that all TOD clocks in the configuration 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.

LzLabs GmbH. Ex. 1016-90

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-microprogram-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.

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. 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 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 clock is set. This occurs when SET CLOCK is executed without encountering any exceptions and any manual TOD-clock control in the configuration is set to the enable-set position. 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 on 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, in 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 32-63. If any clock is set to a value of all zeros in bit positions 32-63 and enters the set state as the result of a signal from another clock, the updating of bits 32-63 of the two clocks is in synchronism.

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 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. 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 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 if 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 causes condition code 3 to be set, and zero is stored.

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.

LzLabs GmbH. Ex. 1016-91

Setting and Inspecting the Clock

The clock can be set to a specific value by execution of SET CLOCK if the manual TOD-clock control of any CPU in the configuration is in the enable-set position. 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.

The TOD clock can be inspected by executing STORE CLOCK, which causes a 64-bit value to be stored. Two executions of STORE CLOCK, possibly on different CPUs in the same configuration, always store different values if the clock is running or, if separate clocks are accessed, both clocks are running and are synchronized.

The values stored for a running clock always correctly imply the sequence of execution of STORE CLOCK on one or more CPUs for all cases where the sequence can be established by means of the program. Zeros are stored in positions to the right of the bit position that is incremented. In a configuration with more than one CPU, however, when the value of a running clock is stored, nonzero values may be stored in positions to the right of the rightmost position that is incremented. This ensures that a unique value is stored.

In a configuration where more than one CPU accesses the same clock, SET CLOCK is interlocked such that the entire contents appear to be updated at once; that is, if SET CLOCK instructions are issued simultaneously by two CPUs, the final result is either one or the other value. If SET CLOCK is issued on one CPU and STORE CLOCK on the other, the result obtained by STORE CLOCK is either the entire old value or the entire new value. When SET CLOCK is issued by one CPU, a STORE CLOCK issued on another CPU may find the clock in the stopped state even when the TOD-clock-sync-control bit is zero in each CPU. The TOD-clock-sync-control bit is bit 2 of control register 0. Since the clock enters the set state before incrementing, the first STORE CLOCK issued after the clock enters the set state may still find the original value introduced by SET CLOCK.

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, or standard epoch, which is the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 a.m. Greenwich Mean Time (GMT) is recommended as the standard epoch for the clock.

3. 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 run. 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 (GMT) May 11, 1971.) Ordinarily, testing bit 0 for a one is sufficient to determine if the clock value is in the standard epoch.

   In converting to or from the current date or time, the programming support assumes each day to be 86,400 seconds. It does not take into account "leap seconds" inserted or deleted because of time-correction standards.

4. 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.

5. 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

LzLabs GmbH. Ex. 1016-92

6. The following chart shows the clock setting at the start of various years. The clock settings, expressed in hexadecimal notation, correspond to 0 AM Greenwich Mean Time on January 1 of each year.

Year	Clock Setting (Hex)
1900	0000 0000 0000 0000
1976	8853 BAF0 B400 0000
1980	8F80 9FD3 2200 0000
1984	96AD 84B5 9000 0000
1988	9DDA 6997 FE00 0000
1992	A507 4E7A 6C00 0000
1996	AC34 335C DA00 0000
2000	B361 183F 4800 0000

7. 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[1]	72C E4E2 6E00 0000

[1] Number of days in four years, including a leap year.

8. 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.

TOD-CLOCK SYNCHRONIZATION

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. 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. Thus, if all clocks are set to the same value, they stay in synchronism.
- Comparing the rightmost 32 bits 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.
- 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 32-63 of any running clock in the configuration are incremented to zero. This permits the program to synchronize all clocks to any particular clock without requiring special operator action to select a "master clock" as the source of the clock-synchronization pulses.

Programming Notes

1. TOD-clock synchronization provices for checking and synchronizing only the rightmost bits of the TOD clock. 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 TOD-clock contents.
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.

LzLabs GmbH. Ex. 1016-93

In a configuration with more than one CPU, each CPU has a separate clock comparator.

The clock comparator has the same format as the TOD clock. In the basic form, the clock comparator 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 specific 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 or 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 has to be replaced, the TOD clock has to be set, the TOD clock has to wrap to zero, or the clock-comparator-subclass mask has to be set to zero. Otherwise, loops of external interruptions are formed.

2. The instruction STORE CLOCK 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 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 TOD clock, except that bit 0 is considered a sign. In the basic form, the CPU timer 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 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 meas-

LzLabs GmbH. Ex. 1016-94

ure 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 specific 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. Hence, 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 has to be replaced, the value in the CPU timer has to wrap to a positive value, or the CPU-timer-subclass mask has to 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 interruption occurs when the execution of STORE CPU TIMER is completed.

EXTERNALLY INITIATED FUNCTIONS

SERVICE SIGNAL

The service-signal facility permits the service processor to communicate with the CPU. Communications to the service processor are model-dependent and are accomplished by means of the DIAGNOSE instruction. When the service processor has completed all or part of a function requested by means of the DIAGNOSE instruction, a service-signal interruption is generated. The service-signal external interruption is a floating interruption condition and can be accepted by any CPU in the configuration. The service-signal request causes an external interruption with the interruption code 2401 hex. A 32-bit parameter is also stored in the word at real location 128. The subclass mask

LzLabs GmbH. Ex. 1016-95

for service signal is bit 22 of control register 0.

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, CPU timer, clock comparator, 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. Clearing does not affect external storage, such as direct-access storage devices used by the control program to hold the contents of unaddressable pages.

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. The figure "Manual Initiation of Resets" 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 the figure "Summary of Reset Actions." For information concerning what resets can be performed by the SIGNAL PROCESSOR instruction, see the section "Signal-Processor Orders" in this chapter.

LzLabs GmbH. Ex. 1016-96

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.

Manual Initiation of Resets

LzLabs GmbH. Ex. 1016-97

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*	C*[1]	C*
Prefix	U	U/V	C	C	C
CPU timer	U	U/V	C	C	C
Clock comparator	U	U/V	C	C	C
Control registers	U	U/V	I	I	I
General registers	U	U/V	U/V	C	C
Floating-point registers	U	U/V	U/V	C	C
Storage keys	U	U	U	C	C[3]
Volatile main storage	U	U	U	C	C[3]
Nonvolatile main storage	U	U	U	C	U
TOD clock	U[2]	U[2]	U[2]	U[2]	T[3]
Floating interruption conditions	C	U	U	C	C[3]
I/O system	R	U	U	R	R[4]

Explanation:

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.

S The CPU is reset; current operations, if any, are terminated; the TLB is 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.

Summary of Reset Actions (Part 1 of 2)

LzLabs GmbH. Ex. 1016-98

Explanation (Continued):

* 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 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.

3 When these units are separately powered, the action is performed only when the power for the unit is turned on.

4 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.

Summary of Reset Actions (Part 2 of 2)

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. A broadcast machine check 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 translation-lookaside buffer is 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.

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:

LzLabs GmbH. Ex. 1016-99

1. The contents of the current PSW, prefix, CPU timer, and clock comparator 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 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 the section "I/O-System Reset" in Chapter 17, "I/O Support Functions").

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 the section "I/O-System Reset" in Chapter 17, "I/O Support Functions"). 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.

Clear Reset

Clear reset combines the initial-CPU-reset function with an initializing function which causes the following actions:

1. The general and floating-point registers are set to zero.

2. The contents of the main storage and the storage keys in the configuration are set to zero with valid checking-block code.

3. 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 quiesced 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 such a case, 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, main 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 initi-

LzLabs GmbH. Ex. 1016-100

ated separately. The following sections describe the power-on resets for the CPU, TOD clock, main 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 and floating-point registers are cleared to zeros with valid checking-block code.

TOD-Clock Power-On Reset: The power-on reset causes the value of the TOD clock to be set to zero and causes the clock to enter the not-set state.

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.

Channel-Subsystem Power-On Reset: The channel-subsystem power-on reset causes I/O-system reset to be performed in the channel subsystem. (See the section "I/O-System Reset" in Chapter 17, "I/O Support Functions.")

INITIAL PROGRAM LOADING

Initial program loading (IPL) provides a manual means for causing a program to be read from a specified 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 designating an input device by means of a four-digit device number with the load-unit-address controls and subsequently activating the load-normal or load-clear key. The load-normal key causes an initial-CPU-reset and a subsystem-reset operation to be performed on this CPU, and the load-clear key causes a clear-reset operation to be performed on this CPU. This CPU then enters the load state. Other CPUs in the configuration perform CPU reset and clear reset, respectively. Subsequently, a channel-program read operation is initiated from the selected input device. The effect of executing the channel program is as if a format-0 CCW in 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. This CPU does not necessarily enter the stopped state during the execution of the reset operation. The load indicator is on while the CPU is in the load state.

The details of the channel-subsystem portion of the IPL operation are defined in the section "Initial Program Loading" in Chapter 17, "I/O Support Functions."

When the IPL I/O operation is completed successfully, the subsystem ID of 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 if 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 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. 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.

The figure "Assigned Storage Locations for Store Status" lists the fields that are stored, their length, and their location in main storage.

Field	Length in Bytes	Absolute Address
CPU timer	8	216
Clock comparator	8	224
Current PSW	8	256
Prefix	4	264
Fl-pt registers 0-6	32	352
General registers 0-15	64	384
Control registers 0-15	64	448

Assigned Storage Locations for Store Status

LzLabs GmbH. Ex. 1016-101

The contents of the registers are not changed. If an error is encountered during the operation, the CPU enters the check-stop state.

The store-status operation can be initiated manually by use of the store-status key (see Chapter 12, "Operator Facilities"). The store-status operation can also be initiated at the addressed CPU by executing SIGNAL PROCESSOR, specifying the stop-and-store-status order. Execution of SIGNAL PROCESSOR specifying the store-status-at-address order permits the same status information to be stored at a designated address (see "Signal Processor Orders" in this chapter).

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 the section "Control Registers" in this chapter.

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.

CPU-ADDRESS IDENTIFICATION

Each CPU has a number 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.

LzLabs GmbH. Ex. 1016-102

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 the figure "Encoding of Orders."

Code	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-FF	Unassigned

Encoding of Orders

The orders are defined as follows:

Sense: The addressed CPU presents its status to the issuing CPU (see the section "Status Bits" in this chapter 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, 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 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 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 operating state.

Start: The addressed CPU performs the start function (see the section "Stopped, Operating, Load, and Check-Stop States" in this chapter). 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 the section "Stopped, Operating, Load, and Check-Stop States" in this chapter). 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 the section "Restart Interruption" in Chapter 6, "Interruptions"). 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 operating state.

Stop and Store Status: The addressed CPU performs the stop function, followed by the store-status function (see the section "Store Status" in this chapter). 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 operating state.

Initial CPU Reset: The addressed CPU performs initial CPU reset (see the section "Resets" in this chapter). The

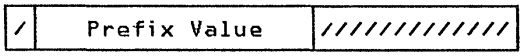
LzLabs GmbH. Ex. 1016-103

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 the section "Resets" in this chapter). 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 format of the parameter register is:



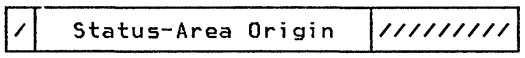
The set-prefix order is completed as follows:

- 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 availability. 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- 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 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 TLB of the addressed CPU is cleared of its 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 area into which the status of the addressed CPU is stored. Bit 0 and bits 23-31 of the parameter register are ignored.

The order is accepted only if the addressed CPU is in the stopped state, the status-area origin 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 format of the parameter register is:



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 address of the area into which status is to be stored is tested for availability. The absolute address of a 512-byte area of storage 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- or 31-bit addressing mode. The 512-byte block of storage at this address is accessed. The

LzLabs GmbH. Ex. 1016-104

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 status of the addressed CPU is placed in the designated area. The information stored, and the format of the area receiving the information, are the same as for the stop-and-store-status order, except that each field, rather than being stored at an offset from the beginning of absolute storage, is stored in the designated area at an offset that is the same as that for the absolute area. Bytes 0-215, 232-255, and 268-351 of the designated area remain unchanged. (See the section "Store Status" in this chapter).

- A serializing and checkpoint-synchronization function is performed on the addressed CPU following storing of the status.

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 issued SIGNAL PROCESSOR is using the CPU-signaling-and-response facility. The concurrently issued instruction may or may not have been issued by or to the addressed CPU and may or may not have been issued to this CPU. The order is rejected. Condition code 2 is set. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

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 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, or store-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 by a CPU addressing itself.

   c. A manual initial-program-load function has been initiated at the addressed CPU, and the reset portion, but not the program-load portion, of the function has been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR 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, or store status at address, 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 the section "Status Bits" in this chapter.

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.

   b. A manual-reset function has been initiated at the addressed CPU, and the function has not yet been completed. The term "manual-reset function" includes the reset portion of IPL. This condition cannot arise as a result of a SIGNAL PROCESSOR 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,

LzLabs GmbH. Ex. 1016-105

or store status at address, 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 the section "Status Bits" in this chapter.

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 response to the designated 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 the figure "Status Conditions."

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

Status Conditions

The status condition assigned to bit position 0 is generated by the CPU executing SIGNAL PROCESSOR. The remaining status conditions are generated by the addressed CPU.

When the equipment-check condition exists, 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 contents of 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 22-27 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 order, each condition is indicated only when the condition precludes the successful execution of the designated 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.

LzLabs GmbH. Ex. 1016-106

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.

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.

Incorrect State: A set-prefix or store-status-at-address order has been rejected because the addressed CPU is not stopped. 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.

Invalid Parameter: The parameter value supplied with a set-prefix or store-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.

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. 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 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 microprogram is not loaded. 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 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, 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 by a CPU addressing itself.

Inoperative: This condition indicates that the performance of the order-code specification 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.

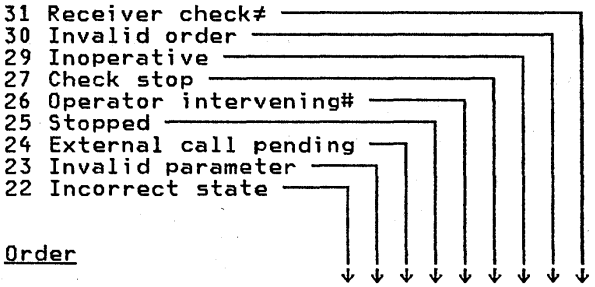
Invalid Order: This condition exists during the communications associated with the execution of SIGNAL PROCESSOR when an unassigned or uninstalled order code is decoded.

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.

LzLabs GmbH. Ex. 1016-107

Status Condition



Order									
	↓	↓	↓	↓	↓	↓	↓	↓	↓
Sense	0	0	X	X	X	X	0	0	X
External call	0	0	X	0	X	X	0	0	X
Emergency signal	0	0	0	0	X	X	0	0	X
Start	0	0	0	0	X	X	X	0	X
Stop	0	0	0	0	X	X	X	0	X
Restart	0	0	0	0	X	X	X	0	X
Stop and store status	0	0	0	0	X	X	X	0	X
Initial CPU reset	0	0	0	0	X	0	X	0	X
CPU reset	0	0	0	0	X	0	X	0	X
Set prefix	X	X	0	0	X	X	X	0	X
Store status at addr.	X	X	0	0	X	X	X	0	X
Unassigned order	0	0	0	0	X	E	X	1	X

Explanation:

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.

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.

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 specified by the R_1 field of the SIGNAL PROCESSOR instruction and sets condition code 1.

Programming Notes

1. A CPU can obtain the following functions by 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 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. 1016-108

Instructions 5-2
Operands 5-2
Instruction Format 5-2
Register Operands 5-4
Immediate Operands 5-4
Storage Operands 5-4
Address Generation 5-4
Bimodal Addressing 5-4
Sequential Instruction-Address Generation 5-5
Operand-Address Generation 5-5
Formation of the Intermediate Value 5-5
Formation of the Address 5-5
Branch-Address Generation 5-6
Formation of the Branch Address 5-6
Instruction Execution and Sequencing 5-6
Decision-Making 5-6
Loop Control 5-7
Subroutine Linkage 5-7
Interruptions 5-11
Types of Instruction Ending 5-11
Completion 5-11
Suppression 5-11
Nullification 5-11
Termination 5-11
Interruptible Instructions 5-11
Point of Interruption 5-11
Execution of Interruptible Instructions 5-12
Exceptions to Nullification and Suppression 5-12
Storage Change and Restoration for DAT-Associated Access Exceptions 5-13
Modification of DAT-Table Entries 5-13
Trial Execution for Editing Instructions and TRANSLATE 5-14
Interlocked Update for Nullification and Suppression 5-14
Authorization Mechanisms 5-14
Mode Requirements 5-15
Extraction-Authority Control 5-15
PSW-Key Mask 5-15
Secondary-Space Control 5-15
Subsystem-Linkage Control 5-15
ASN-Translation Control 5-15
Authorization Index 5-16
PC-Number Translation 5-17
PC-Number Translation Control 5-18
PC-Number Translation Tables 5-18
Linkage-Table Entries 5-18
Entry-Table Entries 5-18
PC-Number-Translation Process 5-19
Linkage-Table Lookup 5-20
Entry-Table Lookup 5-21
Recognition of Exceptions During PC-Number Translation 5-21
Sequence of Storage References 5-21
Interlocks for Virtual-Storage References 5-22
Instruction Fetching 5-23
DAT-Table Fetches 5-24
Storage-Key Accesses 5-24
Storage-Operand References 5-25
Storage-Operand Fetch References 5-25
Storage-Operand Store References 5-25
Storage-Operand Update References 5-26
Storage-Operand Consistency 5-27
Single-Access References 5-27
Multiple-Access Operands 5-27
Block-Concurrent References 5-27
Consistency Specification 5-27
Relation Between Operand Accesses 5-28

LzLabs GmbH. Ex. 1016-109

Other Storage References 5-29
Serialization 5-29
CPU Serialization 5-29
Channel-Program Serialization 5-30

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

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, 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 eight-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, designated by the B, X, and D fields, respectively, in the instruction.

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 FORMAT

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 eight basic formats: RR, RRE, RX, RS, SI, S, SSE, and SS, with two variations of SS. (See the figure "Basic Instruction Formats.")

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

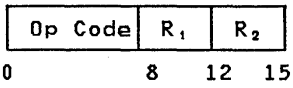
LzLabs GmbH. Ex. 1016-110

these exceptions are explicitly identified in the individual instruction descriptions.

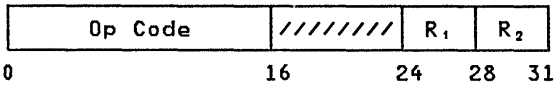
The format names indicate, in general terms, the classes of operands which participate in the operation:

- RR denotes a register-and-register operation.
- RRE denotes a register-and-register operation having an extended op-code field.
- RX denotes a register-and-indexed-storage operation.
- RS denotes a register-and-storage operation.
- 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 having an extended op-code field.

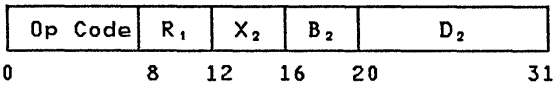
RR Format



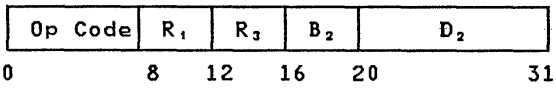
RRE Format



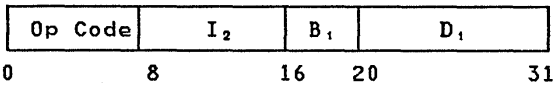
RX Format



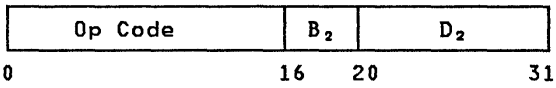
RS Format



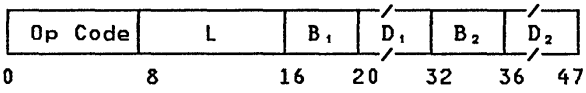
SI Format

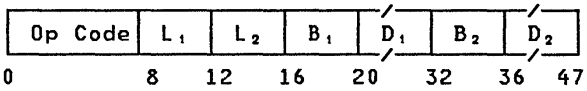


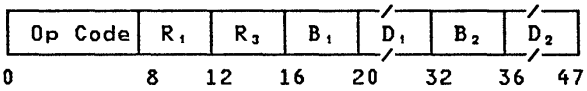
S Format



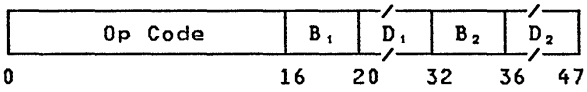
SS Format







SSE Format



Basic Instruction Formats

The first byte or, in the RRE, S, and SSE formats, the first two bytes of an instruction contain the op code. For some instructions in the S format, all or a portion of the second byte is ignored.

The first two bits of the first or only byte of the op code specify the length and format of the instruction, as follows:

Bit Positions 0-1	Instruction Length (in Halfwords)	Instruction Format
00	One	RR
01	Two	RX
10	Two	RRE/RS/RX/S/SI
11	Three	SS/SSE

In the format illustration for each individual instruction description, the op-code field shows 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.

LzLabs GmbH. Ex. 1016-111

Register Operands

In the RR, RRE, RX, and RS 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." In the RR and RRE 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 RS format, the use of the R_3 field depends on the instruction.

The R field designates a general register in the general and control instructions and a floating-point register in the floating-point instructions. In the instructions LOAD CONTROL and STORE CONTROL, the R field designates a control register.

Unless otherwise indicated in the individual instruction description, the register operand is one register in length (32 bits for a general register or a 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 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 designate the first operand, which is one byte in length.

Storage Operands

In the SI, SSE, and SS 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 S, RS, SSE, and SS 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 format, 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.

In the SS format with a single, eight-bit length field, 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, except for the following instructions: EDIT, EDIT AND MARK, TRANSLATE, and TRANSLATE AND TEST.

In the SS format, with two length fields given, 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, 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 of the same length, called the effective length. The effective length is equal to the true length or 256, whichever is less. The instructions using this format, which are MOVE TO PRIMARY, MOVE TO SECONDARY, and MOVE WITH KEY, set the condition code to facilitate programming a loop to move the total number of bytes specified by the true length.

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.

LzLabs GmbH. Ex. 1016-112

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 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 specify 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 zeros 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 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

LzLabs GmbH. Ex. 1016-113

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.

BRANCH-ADDRESS GENERATION

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, RS, or RX.

In the RS and RX formats, the branch address is designated by a base address, a displacement, and, for RX, an index. In the RS and RX formats, the branch address generation follows the normal rules for operand-address generation.

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.

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 1-7 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 executed.

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 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.

DECISION-MAKING

Facilities for decision-making are provided by BRANCH ON CONDITION. This instruction inspects 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,

LzLabs GmbH. Ex. 1016-114

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," 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 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. These branches, being specialized, provide increased performance for these tasks.

SUBROUTINE LINKAGE

Subroutine linkage is provided by the BRANCH AND LINK and BRANCH AND SAVE 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 modes. Linkage between a 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 problem or supervisor state and may be in either the same address space or an address space different from that of the caller. In general, it is used to transfer control to a program of higher authority. PROGRAM TRANSFER permits a change of instruction address, addressing mode, and address space. PROGRAM TRANSFER also permits a reduction in PSW-key-mask authority and a change from supervisor to problem state. In general, it is used to transfer control from one program to another of equal or lower authority. PROGRAM TRANSFER can be used to return from a program called 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 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 an addressing mode and an 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 linkage instructions provided and the functions performed by each are summarized in the figure "Linkage-Instruction Summary."

LzLabs GmbH. Ex. 1016-115

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]
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
SVC[2]	RR	Yes	Yes	Yes	Yes	Yes	Yes	-	-	-	-

Explanation:

- \- No
- AM Saved only if 31-bit addressing mode is specified.
- [1] The action takes place only if the associated R field in the instruction is nonzero.
- [2] MC and SVC, as part of the interruption, save the entire current PSW and load a new PSW.
- * In 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 31-bit mode, the addressing mode and the 31-bit instruction address are saved, and the 31-bit instruction address is set.
- ** A change from supervisor to problem state is allowed; a privileged-operation exception is recognized when a change from problem to supervisor state is specified.
- # Monitor-mask bits provide a means of disallowing linkage, or enabling linkage, for selected classes of events.

Linkage-Instruction Summary

Programming Note

To give the reader a better understanding of the utility and intended usage of the linkage instructions, the following paragraphs in this note describe various program linkages and conventions and the use of the linkage instructions in these situations.

The linkage instructions are provided to permit System/370 programs to operate with no modification or only slight modification on systems operating in the 370-XA mode and also to provide additional function for those programs which are designed to take advantage of operation in the 370-XA mode. 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 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 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.

LzLabs GmbH. Ex. 1016-116

PROGRAM CALL is provided for fast communication to a program operating in 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 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 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. It is assumed that an extension to the current V-type address constant (VCON) will be established by the assembler and linkage editor which consists of a 31-bit entry-point address and a leftmost bit indicating whether the entry is in the 24-bit or 31-bit addressing mode. This extended VCON is shown here as "VCONE." This calling sequence would normally be:

```
L   15,VCONE
BASSM   14,15
```

The return from such a routine would normally be:

```
BSM   0,14
```

The BRANCH AND LINK (BAL, BALR) instructions are provided primarily for compatibility reasons. They are 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 BAL and BALR be avoided since they place nonzero information in the left part of the general register in the 24-bit addressing mode, which may lead to problems. Additionally, BAL and BALR are likely to be slower than BAS and BASR because BAS and BASR always save the right half of the PSW, whereas BAL and BALR 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 BAL or BALR will be:

```
BR   14
```

However, the standard "return instruction":

```
BSM   0,14
```

operates correctly for all cases except for a calling BAL issued in 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 mode also causes the same ILC effect.

The BRANCH AND SAVE (BAS, BASR) instructions are 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. BAS and BASR are the fastest linkage instructions since the linkage information is not mode-sensitive and since the instructions do not change the addressing mode.

The return instruction from a routine called by BAS or BASR may be either

```
BR   14
```

or

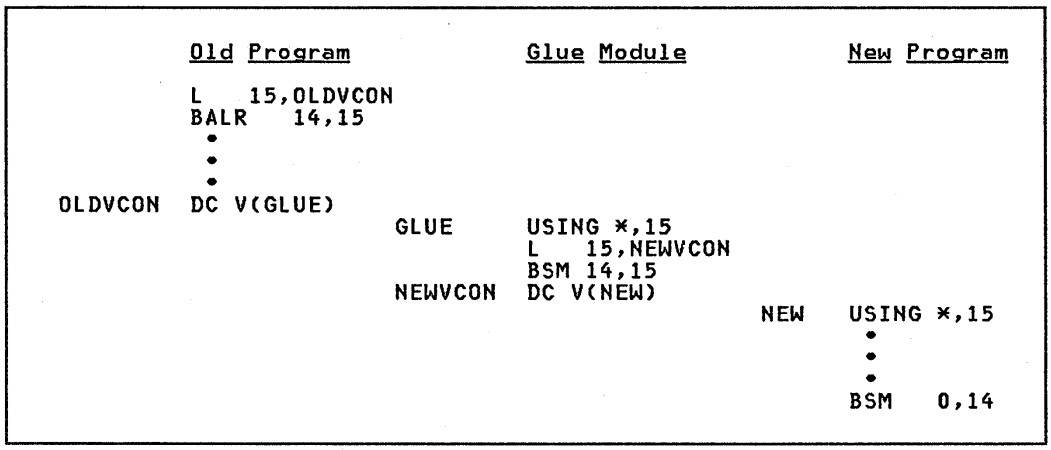
```
BSM   0,14
```

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 a program is sometimes referred to as a glue module. The

LzLabs GmbH. Ex. 1016-117

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 the figure "Glue Module."

Note that the "BSM 14,15" in the glue module causes the addressing mode to be saved in bit 0 of general register 14 and that bits 1-31 of general register 14 are unchanged. Thus, when "BSM 0,14" is issued in the new program, control passes directly back to the old program without passing through the glue module again.



Glue Module

LzLabs GmbH. Ex. 1016-118

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 the section "Interruptible Instructions" later in this chapter.

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.

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 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 the section "Tracing" in Chapter 4, "Control."

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

LzLabs GmbH. Ex. 1016-119

interruption is permitted after partial completion of the instruction:

COMPARE LOGICAL LONG
MOVE LONG
TEST BLOCK

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.

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.

When an instruction 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, suppression, nullification) is associated with the unit of operation. In the case of termination, the entire instruction is terminated, not just the unit of operation.

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, nullification, or suppression of a unit of operation, all preceding units of operation have been completed.

On completion of a unit of operation other than the last one (and on nullification of any unit of operation), the instruction address in the old PSW designates the interrupted instruction, and 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 on the interruption is made the current PSW. It depends on the instruction how the operand parameters are adjusted.

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 from the point of interruption. As in the case of completion and nullification, it depends on the instruction how the operand parameters are adjusted.

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.

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

LzLabs GmbH. Ex. 1016-120

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 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 specified 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, a channel 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 a channel program and are not observable by another CPU 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:

   - Operate on one storage page at a time
   - Perform preliminary testing to ensure that no exceptions occur for any of the required pages
   - 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 unpredict-

LzLabs GmbH. Ex. 1016-121

able whether or not an interruption occurs for an access exception that was not initially applicable.

Trial Execution for Editing Instructions and TRANSLATE

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 a channel program or by another CPU, 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.

Interlocked Update for Nullification and Suppression

When an exception which is defined to cause suppression or nullification is recognized for an instruction with a store-type operand, an interlocked update which does not change the contents of the location may occur for that portion, if any, of the store-type operand for which no access exception exists. The interlocked update can occur only if the priority of the exception is equal to or lower than the priority of an access exception for the store-type operand.

When the exception is a specification exception for a store-type operand which requires alignment on integral boundaries, the interlocked update which may occur is limited to the single byte at the location specified by the operand address.

Programming Note

The interlocked update is only observable by channel programs; it is not observable by other CPUs. Examples of when an interlocked update may occur to the destination-operand location in storage are:

- Decimal-divide exception for DIVIDE DECIMAL
- Specification exception for an odd register number for COMPARE DOUBLE AND SWAP
- Data exception for an invalid decimal sign for ADD DECIMAL

AUTHORIZATION MECHANISMS

The authorization mechanisms which are described in this section permit the control program to establish the degree of function which is provided to a particular semiprivileged program. (A summary of the authorization mechanisms is given in the figure "Summary of Authorization Mechanisms.") The authorization mechanisms are intended for use by programs considered to be semiprivileged, that is, programs which 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.

The 13 instructions which are controlled by the authorization mechanisms are called semiprivileged instructions and are described in Chapter 10, "Control Instructions." The 13 semiprivileged instructions, along with the privileged instruction LOAD ADDRESS SPACE PARAMETERS, are listed in the next figure, "Summary of Authorization Mechanisms."

Programming Note

The authorization mechanisms are defined such that if zeros are placed in the associated control-register positions, a problem program attempting to use the semiprivileged instructions causes a privileged-operation or special-operation exception to be recognized, and the operation is suppressed.

LzLabs GmbH. Ex. 1016-122

Mode Requirements

Most of the semiprivileged instructions can be executed only with DAT on. PROGRAM CALL and PROGRAM TRANSFER are valid only in primary-space mode. 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.

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, and the operation is suppressed. The extraction-authority-control bit is not examined in the supervisor state.

PSW-Key Mask

The PSW-key mask consists of bits 0-15 in control register 3. 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 CALL and PROGRAM TRANSFER and is loaded by LOAD ADDRESS SPACE PARAMETERS. 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 three move instructions that specify a second access key: MOVE TO PRIMARY, MOVE TO SECONDARY, and MOVE WITH 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, and the operation is suppressed. 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 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, and the operation is suppressed. The secondary-space-control bit is examined in both the problem and supervisor states.

Subsystem-Linkage Control

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
PROGRAM TRANSFER

Otherwise, a special-operation exception is recognized, and the operation is suppressed. The subsystem-linkage-control bit 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:

LzLabs GmbH. Ex. 1016-123

LOAD ADDRESS SPACE PARAMETERS
SET SECONDARY ASN
PROGRAM CALL with space switching
PROGRAM TRANSFER with space switching

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

Authorization Index

The authorization index is contained in bits 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 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.

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 uses 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, and the operation is nullified.

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, and the operation is nullified.

The instruction PROGRAM CALL with space switching causes a new authorization index to be loaded from the ASN-second-table entry. This permits the program which is called to be given an authorization index which authorizes it to access more address spaces than those authorized for the calling program.

LzLabs GmbH. Ex. 1016-124

Instr	Mode Requirement		Authorization Mechanism								Space-Switch-Event-Control Bit (CR1.0)
	Priv Op	Trans Mode	Subsystem-Linkage Control (CR5.0)	Secondary-Space Control (CR0.5)	ASN-Translation-Control (CR14.12) Uncd	ASN-Translation-Control (CR14.12) Cond	Extraction-Authority Control (CR0.4)	PSW-Key Mask (CR3.0-15) Bit Test	PSW-Key Mask (CR3.0-15) AND AKM[1]	Authorization Index (CR4.0-15)	
EPAR		SO-PS					Q				
ESAR		SO-PS					Q				
IAC		SO-PS					Q				
IPK							Q				
IVSK		SO-PS					Q				
LASP	P				SO					CC	CC
MVCP		SO-PS		SO				Q			
MVCS		SO-PS		SO				Q			
MVCK								Q			
PC-cp		SO-P	SO						Q		
PC-ss		SO-P	SO			SO			Q		X
PT-cp	Q[2]	SO-P	SO								
PT-ss	Q[2]	SO-P	SO			SO				PA	X
SAC		SO-PS		SO							
SPKA								Q			
SSAR-cp		SO-PS			SO						
SSAR-ss		SO-PS			SO					SA	

Explanation:

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 supervisor state when in problem state.

CC Space-switch-event-control bit and authorization index tests cause a condition code to be set.

CRx.y Control register x, bit position y.

P Privileged-operation exception for privileged instruction.

PA Authority checked in problem and supervisor states; violation causes a primary-authority exception.

Q Privileged-operation exception for semiprivileged instruction. Authority checked only in problem state.

SO Authority checked in problem and supervisor states; violation causes a special-operation exception.

SO-PS CPU must be in primary-space mode or secondary-space mode; if the CPU is in real mode, a special-operation exception is recognized in both problem and supervisor states.

SO-P CPU must be in primary-space mode; if the CPU is in secondary-space mode or in real mode, a special-operation exception is recognized in both problem and supervisor states.

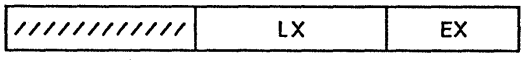
SA Authority checked in problem and supervisor states; violation causes a secondary-authority exception.

X When bit 0 of control register 1 is one, a space-switch event is recognized. The operation is completed. The event is recognized in both the problem and supervisor states.

Summary of Authorization Mechanisms

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 format of the effective address, from which the PC-number is taken, is shown in the following diagram:



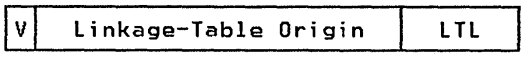
The translation is performed by means of two tables: a linkage table and an entry table. Both of these tables reside in real storage. The linkage-table designation resides in control

LzLabs GmbH. Ex. 1016-125

register 5. The entry table is designated by means of a linkage-table entry.

PC-NUMBER TRANSLATION CONTROL

PC-number translation is controlled by means of the linkage-table designation in control register 5. 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
PROGRAM TRANSFER

Otherwise, a special-operation exception is recognized, and the operation is suppressed. The system-linkage-control bit is examined in both the problem and 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 designate 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.

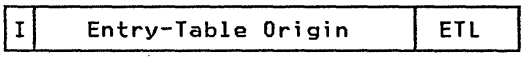
PC-NUMBER TRANSLATION TABLES

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 linkage-index portion of the PC number is used to select a linkage-table entry. The entry fetched from the linkage table designates the availability, origin, and length of the corresponding entry table.

An entry in 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, and the operation is nullified.

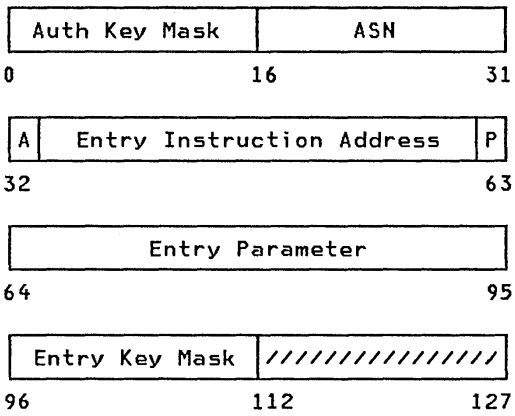
Entry-Table Origin: Bits 1-25, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the entry table.

Entry-Table Length (ETL): Bits 26-31 designate the length of the entry table in units of 64 bytes, thus making the entry table variable in multiples of four 16-byte entries. The length of the entry table, in units of 64 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

The entry fetched from the entry table is 16 bytes in length and has the following format:

LzLabs GmbH. Ex. 1016-126



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, and the operation is suppressed. The test is not performed in the supervisor state.

ASN: Bits 16-31 specify whether a PC-ss or PC-cp is to occur. When bits 16-31 are zeros, a PC-cp is specified. When bits 16-31 are not all zeros, a PC-ss is specified, and the bits contain 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, and the operation is suppressed.

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.

Entry Key Mask: Bits 96-111 are ORed into the PSW key mask in control register 3 as part of the PROGRAM CALL operation.

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.

PC-NUMBER-TRANSLATION PROCESS

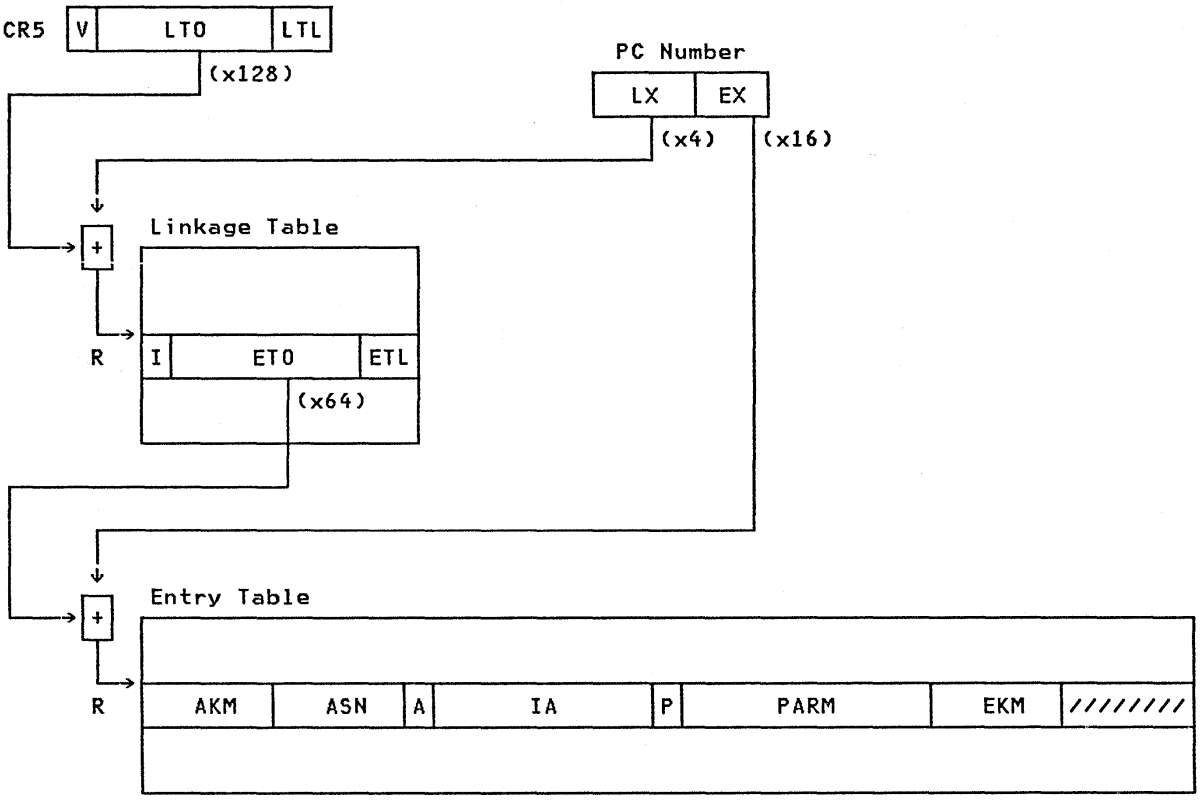
The translation of the PC number is performed by means of a linkage table and entry table both of which reside in real storage.

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 contents of the linkage-table designation in control register 5. 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 linkage table and entry table, key-controlled protection does not apply.

The PC-number-translation process is shown in the figure "PC-Number Translation."

LzLabs GmbH. Ex. 1016-127



R: Address is real

PC-Number Translation

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 control register 5 and adding the linkage index, with two rightmost and 17 leftmost zeros appended. A carry, if any, into bit position 0 is ignored. 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 control register 5, 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 in the seven leftmost bits of the linkage index, an LX-translation exception is recognized.

All four bytes of the linkage-table entry are fetched concurrently. 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.

LzLabs GmbH. Ex. 1016-128

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 the entry index, with four rightmost and 19 leftmost zeros appended. A carry, if any, into bit position 0 is ignored. 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 in the six leftmost bits of the entry index, an EX-translation exception is recognized.

The 16-byte entry-table entry is fetched by using the real address. The entry is fetched left to right, a word at a time. 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.

SEQUENCE OF STORAGE REFERENCES

Conceptually, the CPU processes instructions one at a time, with the execution of one instruction preceding the execution of the following instruction. The execution of the instruction specified 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 appears to the program 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.

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, channel programs, and other CPU programs may, unless otherwise constrained, observe a sequence that differs from the conceptual sequence. Also, in certain situations involving dynamic address translation where different virtual addresses map to the same real address, the effect of its own overlapped operation may be observable by the CPU itself.

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, a channel or another CPU may be able to observe intermediate or partially completed results.

When a program on the CPU interacts with a channel program or another CPU program, the programs may have to take into consideration that a single operation may consist in a series of storage references, that a storage reference may

LzLabs GmbH. Ex. 1016-129

in turn consist in a series of accesses, and that the conceptual and actual sequences of these accesses may differ. Storage references associated with instruction execution are of the following types: instruction fetches, DAT-table fetches, and storage-operand references. For the purposes of the following discussion, storage-key accesses are also considered to be storage references.

Programming Note

The sequence of execution may differ from the simple conceptual definition in the following ways.

- As viewed by a program in the CPU, instructions may appear to be prefetched when different effective addresses are used. (See the section "Interlocks for Virtual-Storage References" in this chapter.)
- As viewed by a channel program or another CPU program, 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 one of the following sections.
- As viewed by a channel program or another CPU program, the storage-operand accesses associated with one instruction are not necessarily performed in the conceptual sequence. (See the section "Relation Between Operand Accesses" in this chapter.)
- As viewed by a channel program, in certain unusual situations, the contents of storage may appear to change and then be restored to the original value. (See the section "Storage Change and Restoration for DAT-Associated Access Exceptions" earlier in this chapter.)

INTERLOCKS FOR VIRTUAL-STORAGE REFERENCES

As described in the previous section, CPU operation appears to that CPU to be performed sequentially; the results stored by one instruction appear to the CPU to be completed before the next instruction is fetched. This appearance is maintained in overlapped machines by means of special circuitry to detect accesses to a common location by comparing effective addresses.

For purposes of this definition, the term "effective address" is used to denote the address before translation, if any, regardless of whether the address is virtual, real, or absolute. If two effective addresses have the same value and map to the same absolute location, the effective addresses are said to be the same even though one may be real or in a different address space.

When all accesses to a main-storage location are made by using the same effective address, then the above rule is strictly maintained, as observed by the CPU itself. When different effective addresses are used to access the common location, the above rule does not hold in two cases:

1. For some instructions, the definition specifies the results which must be obtained for overlapping operands. This definition is specified in terms of the sequence of the storage accesses; that is, the results of some or all of the stores of one operand must be placed in storage before some parts or all parts of the other operand are fetched. When the store and the fetch are performed by means of different effective addresses, then the operand may appear to be fetched before the store.
2. When an instruction changes the contents of a main-storage location from 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. This does not occur if an intervening operation causes the prefetched instructions to be discarded. A definition of when prefetched instructions must be discarded is included in the section "Instruction Fetching" in this chapter.

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 a virtual, real, or absolute address. Analogously, any conceptually prior references to the storage block appear completed when the key for that block is changed or inspected.

Programming Note

A single main-storage location can be accessed in several ways by more than one address.

LzLabs GmbH. Ex. 1016-130

1. The DAT tables may be set up in such a way that multiple addresses in a single address space, or virtual 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 bit in the PSW, or, for logical and instruction addresses, by turning DAT on or off.

3. Certain instructions use real addresses, and the instructions MOVE TO PRIMARY and MOVE TO SECONDARY access two address spaces.

4. 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.

5. The real-to-absolute mapping may be changed by means of the SET PREFIX instruction or a reset.

6. 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.

7. 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 is to describe the effects caused by case 1 above.

For case 2, the effect is not observable since prefetched instructions are discarded and the effect of delayed stores is not observable to the CPU itself.

For case 3, for those instructions which fetch by using real addresses (for example, LOAD REAL ADDRESS), no effect is observable. This is because the only effect across instructions is the prefetching of instructions, and instructions which fetch by using real addresses thus have no special effect. All instructions which store by using a real address or which store into another address space cause prefetched instructions to be discarded, and no effect is observable.

Cases 4 and 5 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 6 and 7 involves accesses as observed by channel programs and other CPUs and is covered in the following sections in this chapter.

INSTRUCTION FETCHING

Instruction fetching consists in fetching the one, two, or three halfwords specified 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 specifies 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. Instructions which are prefetched may be interpreted for execution only for the same virtual address for which the instruction was prefetched.

No limit is established as to 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 channel programs or by other CPU programs does not necessarily change the copy of prefetched instructions. However, if a store that is conceptually earlier occurs on the same CPU using the same effective address as that by which the

LzLabs GmbH. Ex. 1016-131

instruction is subsequently fetched, the updated information is obtained.

All copies of prefetched instructions are discarded when:

- A serializing function is performed
- The CPU enters the operating state
- The CPU changes from one to another of the following modes: primary-space mode, secondary-space mode, and real mode
- A change is made to a translation parameter in control register 1 or 7 when DAT is on

Programming Notes

1. As observed by a CPU itself, its own instruction prefetching is not normally apparent; the only exception occurs when multiple virtual addresses in a single address space, or virtual addresses in different address spaces, map to a single real address. This is described in the section "Interlocks for Virtual-Storage References" in this chapter.

2. The following are some effects of instruction prefetching on the execution of a CPU program as viewed by another CPU or channel program.

   If a channel program or another CPU program changes the contents of a storage location and then sets a flag to indicate that the change has been made, a CPU can test and find the flag set but subsequently can branch to the modified location and execute the original contents. Additionally, when a channel program or another CPU program modifies an instruction, it is possible for the CPU program to recognize the changes to some but not all bit positions of the instruction.

   It is possible for a CPU to prefetch an instruction and subsequently, before the instruction is executed, for another CPU to change the storage key. As a result, a CPU may appear to execute instructions from a protected storage location.

DAT-TABLE FETCHES

The fetching of dynamic-address-translation (DAT) table entries may occur as follows:

1. 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 an INVALIDATE PAGE TABLE ENTRY, PURGE TLB, or SET PREFIX instruction or by CPU reset. 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. All bytes of a DAT-table entry are fetched concurrently, as viewed by all CPUs in the configuration. However, the reference to the entry may appear to access a single byte at a time, as viewed by channel programs.

3. A 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 DAT-table entry.

4. A 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.

5. 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.

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 with the reference to the storage location.

2. When storing is performed, the change bit is set in the associated storage key concurrently with the store operation.

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

LzLabs GmbH. Ex. 1016-132

follows the sequence rules for storage-operand store references and is a single-access reference.

4. The INSERT STORAGE KEY EXTENDED 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 access to the storage key for RESET REFERENCE BIT EXTENDED follows the sequence rules for storage-operand update references. The reference bit is the only bit which is updated.

The record of references provided by the reference bit is not necessarily accurate, and the handling of the reference bit is not subject to the concurrency rules. However, in the majority of situations, reference recording approximately coincides with the storage reference.

The change bit may be set in cases when no storing has occurred. See the section "Exceptions to Nullification and Suppression" in this chapter.

STORAGE-OPERAND REFERENCES

A storage-operand reference is the fetching or storing of the explicit operand or operands in the storage locations specified 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 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 reference 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.

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 reference 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 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 channel programs and other CPU programs. The results of any one instruction are stored in the sequence specified for that instruction.

The CPU does not fetch operands 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.

LzLabs GmbH. Ex. 1016-133

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. In most cases, no special interlock is provided between the fetch and store, and accesses by channel programs and other CPU programs are permitted. An update-type operand is identified in the individual instruction definition by indicating that the access exception is for both fetch and store. The fetch and store accesses associated with an update reference do not necessarily occur one immediately after the other, and it is possible for a channel program or another CPU program to make one or more interleaved accesses to the same location. The interleaved accesses can be either fetches or stores.

The following instructions perform an update which is interlocked against accesses by another CPU to the same location during the execution of the instruction. The instructions INVALIDATE PAGE TABLE ENTRY, TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP cause an interlocked update. 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.

The fetch and store accesses associated with an interlocked-update reference do not necessarily occur one immediately after the other, but all accesses by another CPU are prevented from occurring between the fetch and the store accesses of an interlocked update. Channel-program accesses may occur during the interlock period.

Within the limitations of the above requirements, the fetch and store accesses associated with an update 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 an instruction that causes fetching and subsequently storing of the updated information, 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 but another CPU makes a store operation 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 an update due to another CPU, any change in the contents produced by the interlocked update is lost.

2. 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 and (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 one of the updates can be lost.

3. 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.

4. The instructions TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP facilitate updating of a common storage field by two

LzLabs GmbH. Ex. 1016-134

CPUs. In order for the change by either CPU not to be lost, both CPUs must use an instruction providing an interlocked update. It is possible, however, for a channel to make an access to the same storage location between the fetch and store portions of an interlocked update.

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 operands and the stores associated with storage change and restoration for DAT-associated access exceptions, all storage-operand references are single-access references.

Multiple-Access Operands

For some instructions, 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: CONVERT TO BINARY, CONVERT TO DECIMAL, MOVE WITH OFFSET, PACK, TRANSLATE, TEST BLOCK, and UNPACK.
2. The operands of the decimal instructions.
3. The stores into that portion of the first operand of MOVE LONG which is filled with padding bytes.
4. The stores into a trace entry.

When a storage-operand store reference to a location is not a single-access reference, the contents placed at a byte location are not necessarily the same for each store access; thus, intermediate results in a single-byte location may be observed by channel programs and other CPU programs.

Programming Notes

1. When multiple fetch accesses are made to a single byte that is being changed by a channel program or another CPU program, the result is not necessarily limited to that which could be obtained by fetching the bits individually. For example, the execution of MULTIPLY DECIMAL may consist in repetitive additions and subtractions each of which causes the first operand to be updated in storage.
2. When CPU instructions are used to modify storage locations being accessed by another CPU program or a channel program simultaneously, multiple store accesses to a single byte by the CPU program may result in intermediate values being observed by another CPU program or the channel program. To avoid these intermediate values (especially when modifying a CCW chain), only instructions making single-access references should be used.

Block-Concurrent References

For some references, the accesses to all bytes within a halfword, word, or doubleword are specified to be block-concurrent as observed by other CPU programs. These accesses do not necessarily appear to a channel program 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 be concurrent within a block, no store access to the block by another CPU program is permitted during the time that bytes contained in the block are being fetched. Channel-program accesses to the bytes within the block may occur between the fetches. When a store-type reference is specified to be concurrent within a block, no access to the block, either fetch or store, is permitted by another CPU program during the time that the bytes within the block are being stored. Channel-program accesses to the bytes in the block may occur between the stores.

Consistency Specification

The storage-operand references associated with all S-format and RX-format instructions, with the exception of EXECUTE, CONVERT TO DECIMAL, CONVERT TO BINARY, and the I/O instructions, are

LzLabs GmbH. Ex. 1016-135

block-concurrent, as observed by all CPUs, if the operand is addressed on a boundary which is integral to the size of the operand.

For the instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP, all accesses to the storage operand appear to be block-concurrent as observed by all CPUs.

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 all CPUs to be accessed concurrently.

The instructions LOAD CONTROL and STORE CONTROL access the storage operand in a left-to-right direction, and all bytes accessed within each word appear to all CPUs to be accessed concurrently.

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 all CPUs to be accessed concurrently.

2. The second operand is accessed left to right, and all bytes within a doubleword in the second operand that are moved into a single doubleword in the first operand appear to all CPUs to be fetched concurrently. Thus, if the first and second operands begin on the same byte offset within a doubleword, the second operand appears to be fetched doubleword-concurrent. If the offsets within a doubleword differ by 4, the second operand appears to be fetched word-concurrent.

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 for MOVE LONG appear to all CPUs to be accessed doubleword-concurrent 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 for COMPARE LOGICAL LONG appear to all CPUs to be accessed doubleword-concurrent when both operands start on doubleword boundaries and are an integral number of doublewords in length.

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 all CPUs to be accessed concurrently.

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 all CPUs to be doubleword-concurrent, 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.

RELATION BETWEEN OPERAND ACCESSES

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

LzLabs GmbH. Ex. 1016-136

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.

Programming Note

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 a channel program or another CPU 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 are accessed doubleword-concurrent as observed by other CPUs. These references 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 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.

SERIALIZATION

The sequence of functions performed by a CPU is normally independent of the functions performed by channel programs and other CPU programs. Similarly, the sequence of functions performed by a channel program is normally independent of the functions performed by other channel programs and CPU programs. However, at certain points in its execution, serialization of the CPU program 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 by the CPU, as observed by channel programs and other CPU programs, before the conceptually subsequent storage accesses occur. Serialization affects the sequence of all CPU accesses to storage and to the storage keys, except for those associated with DAT-table-entry

Serialization is performed by CPU reset, all interruptions, and by the execution of the following instructions:

1. 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, SUPERVISOR CALL, and TEST AND SET.
2. LOAD PSW and SET STORAGE KEY EXTENDED.
3. All I/O instructions.
4. PURGE TLB and SET PREFIX, which also cause the translation-lookaside buffer to be cleared of entries.
5. SIGNAL PROCESSOR.
6. INVALIDATE PAGE TABLE ENTRY.
7. TEST BLOCK.
8. MOVE TO PRIMARY, MOVE TO SECONDARY, PROGRAM CALL, PROGRAM TRANSFER, SET ADDRESS SPACE CONTROL, and SET SECONDARY ASN.
9. The three trace functions -- branch tracing, ASN tracing, and explicit

LzLabs GmbH. Ex. 1016-137

tracing -- cause serialization to be performed before the trace action and after completion of the trace action.

The sequence of events associated with a serializing operation is as follows:

- All conceptually previous storage accesses by the CPU are completed, as observed by channel programs and other CPU programs. This includes all conceptually previous stores and changes to the storage keys.
- 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 PSW is fetched, but not necessarily in that sequence.
- 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 program in which the serializing function takes place. It does not affect the sequence of storage accesses under the control of a channel program or another CPU program.

Programming Notes

1. The following are some effects of a serializing operation:
   a. When 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, a channel program and any other CPU 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.

CHANNEL-PROGRAM SERIALIZATION

Serialization of a channel program occurs as follows:

1. For a channel program, 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 CPU programs or another channel program. This includes all accesses for the CCWs, IDAWs, and data.
2. For a channel program, all storage accesses and storage-key accesses are completed, as observed by CPU programs or another channel program, before the subchannel status indicating primary status pending is made available to any CPU.
3. For a channel program, 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 CPU programs or another channel program, before the subchannel status indicating intermediate status pending (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 caused by CPU programs or another channel program. It also does not affect the sequence of storage accesses or storage-key accesses caused by other channel programs.

LzLabs GmbH. Ex. 1016-138

Interruption Action 6-2
  Interruption Code 6-5
  Enabling and Disabling 6-5
  Handling of Floating Interruption Conditions 6-6
  Instruction-Length Code 6-6
    Zero ILC 6-6
    ILC on Instruction-Fetching Exceptions 6-6
  Exceptions Associated with the PSW 6-8
    Early Exception Recognition 6-8
    Late Exception Recognition 6-8
External Interruption 6-9
  Clock Comparator 6-9
  CPU Timer 6-10
  Emergency Signal 6-10
  External Call 6-10
  Interrupt Key 6-10
  Malfunction Alert 6-11
  Service Signal 6-11
  TOD-Clock Sync Check 6-11
Input/Output Interruption 6-11
Machine-Check Interruption 6-12
Program Interruption 6-12
  Program-Interruption Conditions 6-13
    Addressing Exception 6-13
    AFX-Translation Exception 6-15
    ASN-Translation-Specification Exception 6-15
    ASX-Translation Exception 6-15
    Data Exception 6-15
    Decimal-Divide Exception 6-16
    Decimal-Overflow Exception 6-16
    Execute Exception 6-16
    Exponent-Overflow Exception 6-16
    Exponent-Underflow Exception 6-16
    EX-Translation Exception 6-17
    Fixed-Point-Divide Exception 6-17
    Fixed-Point-Overflow Exception 6-17
    Floating-Point-Divide Exception 6-17
    LX-Translation Exception 6-17
    Monitor Event 6-18
    Operand Exception 6-18
    Operation Exception 6-18
    Page-Translation Exception 6-19
    PC-Translation-Specification Exception 6-19
    PER Event 6-19
    Primary-Authority Exception 6-19
    Privileged-Operation Exception 6-20
    Protection Exception 6-20
    Secondary-Authority Exception 6-21
    Segment-Translation Exception 6-21
    Significance Exception 6-21
    Space-Switch Event 6-21
    Special-Operation Exception 6-22
    Specification Exception 6-22
    Programming Note 6-23
    Trace-Table Exception 6-23
    Translation-Specification Exception 6-23
  Collective Program-Interruption Names 6-24
  Recognition of Access Exceptions 6-24
  Multiple Program-Interruption Conditions 6-26
    Access Exceptions 6-29
    ASN-Translation Exceptions 6-30
    Trace Exceptions 6-31
Restart Interruption 6-31
Supervisor-Call Interruption 6-31
Priority of Interruptions 6-32

LzLabs GmbH. Ex. 1016-139

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.

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 the figure "Interruption Action."

LzLabs GmbH. Ex. 1016-140

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		x	terminated or nullified[2]
Repressible cond			13	14, 3-7	x	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			1,2	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			2,3	suppressed or terminated
Fixed-pt overflow	00000000 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
Exponent overflow	00000000 p0001100	000C			1,2	completed
Exponent underflow	00000000 p0001101	000D	22		1,2	completed
Significance	00000000 p0001110	000E	23		1,2	completed
Floating-pt divide	00000000 p0001111	000F			1,2	suppressed
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	2	suppressed
Operand	00000000 p0010101	0015			2	suppressed
Trace table	00000000 p0010110	0016			1,2	nullified
ASN-transl spec	00000000 p0010111	0017			2	suppressed
Space-switch event	00000000 p0011100	001C		1, 0	2	completed
PC-transl spec	00000000 p0011111	001F			2	suppressed
AFX translation	00000000 p0100000	0020			2	nullified
ASX translation	00000000 p0100001	0021			2	nullified
LX translation	00000000 p0100010	0022			2	nullified
EX translation	00000000 p0100011	0023			2	nullified
Primary authority	00000000 p0100100	0024			2	nullified
Secondary auth	00000000 p0100101	0025			2	nullified
Monitor event	00000000 p1000000	0040		8, 16-31	2	completed
PER event	00000000 1nnnnnnn[4]	0080	1	9, 0-3ø	0,1,2,3	completed[5]

Interruption Action (Part 1 of 2)

LzLabs GmbH. Ex. 1016-141

Source Identification	Interruption Code		PSW-Mask Bits	Mask Bits in Ctrl Registers Reg, Bit	ILC Set	Execution of Instruction Identified by Old PSW
EXTERNAL (old PSW 24, new PSW 88)	Locations 134-135					
	Binary	Hex				
Interrupt key	00000000 01000000	0040	7	0, 25	x	unaffected
Malfunction alert	00010010 00000000	1200	7	0, 16	x	unaffected
Emergency signal	00010010 00000001	1201	7	0, 17	x	unaffected
External call	00010010 00000010	1202	7	0, 18	x	unaffected
TOD-clock sync chk	00010000 00000011	1003	7	0, 19	x	unaffected
Clock comparator	00010000 00000100	1004	7	0, 20	x	unaffected
CPU timer	00010000 00000101	1005	7	0, 21	x	unaffected
Service signal	00100100 00000001	2401	7	0, 22	x	unaffected
IUCV[7]	01000000 00000000	4000	7	0, 30	x	unaffected
VMCF[7]	01000000 00000001	4001	7	0, 31	x	unaffected
INPUT/OUTPUT (old PSW 56, new PSW 120)	Locations 184-191					
I/O subclass			6	6, 0-7[6]	x	unaffected
RESTART (old PSW 8, new PSW 0)	None					
Restart key					x	unaffected

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 indicate the extent of the damage and whether or not the unit of operation is nullified 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 or p in the column labeled "Binary."

[4] 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).

[5] The unit of operation is completed, unless a program exception concurrently indicated causes the unit of operation to be nullified, suppressed, or terminated.

[6] Bits 0-7 of control register 6 provide detailed masking of I/O-interruption subclasses 0-7 respectively.

[7] This interruption is used only in a virtual machine; in a real machine, this interruption code is reserved.

@ Additional masks in control register 9, bit positions 16-31, provide detailed control over the source of general-register-alteration PER 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.

x Not stored.

Interruption Action (Part 2 of 2)

LzLabs GmbH. Ex. 1016-142

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 the section "Assigned Storage Locations" in Chapter 3, "Storage.") 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 translation-exception identification, monitor-class number, monitor code, PER code, and PER address, which are stored at real locations 144-159.

For restart interruptions, no interruption code is stored.

For supervisor-call interruptions, the interruption code is stored at real locations 138-139, and the instruction-length code is stored in bit positions 5 and 6 of real location 137.

ENABLING AND DISABLING

By means of mask bits in the current PSW and in 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 these 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. The setting of the significance and exponent-underflow program-mask bits affects the manner in which floating-point operations are completed when the corresponding condition occurs.

The CPU is always enabled for program interruptions for which mask bits are not provided, as well as the supervisor-call and restart interruptions.

The mask bits may allow or disallow all interruptions within the class, or they may selectively allow or disallow interruptions for particular causes. This control may be provided by mask bits in the PSW that are assigned to particular causes, such as the bits assigned to the four maskable program-interruption conditions. Alternatively, there may be a hierarchy of masks, where a mask bit in the PSW controls all interruptions within a type, and mask bits in a control register provide more detailed control over the sources.

When the mask bit is one, the CPU is enabled for the corresponding interruptions. When the mask bit is zero, these interruptions are disallowed. Interruptions that are controlled by a hierarchy of masks are allowed only when all controlling mask bits are ones.

Programming Notes

1. Mask bits in the PSW provide a means of disallowing all 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

LzLabs GmbH. Ex. 1016-143

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.

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 accept the interruption, and then the condition is cleared and not presented to any other CPU in the configuration. A CPU cannot accept 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 accept the interruption when the start key is activated.

Service signal, I/O, and certain machine-check conditions are floating interruption conditions.

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. 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 or an interruption. (See the section "Exceptions Associated with the PSW" later in this chapter.) In the case of LOAD PSW, the instruction address of the LOAD PSW or EXECUTE has been replaced by the instruction address of the new PSW. 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 and the supervisor-call interruption, a PER event may be indicated concurrently with a specification exception having an ILC of 0.

ILC on Instruction-Fetching Exceptions

When a program interruption occurs because of an exception that prohibits access to the instruction, the instruction-length code cannot be set on the basis of the first two bits of the

LzLabs GmbH. Ex. 1016-144

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 originally appearing in the PSW may be obtained.

2. When a segment-translation or page-translation exception is recognized while fetching an instruction, including the target instruction of EXECUTE, 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 other than segment translation or page translation 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 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 access exception encountered in fetching an instruction, and the instruction address has been introduced into the PSW by a means other than sequential operation (by a branch instruction, LOAD PSW, an interruption, or conclusion of an IPL sequence).

   e. The interruption is caused by a specification exception because of an odd instruction address.

   f. The interruption is caused by an early specification exception or by an access exception encountered in fetching an instruction, and changes have been made to a parameter that controls the relation between instruction addresses and real addresses. The relation between instruction addresses and real addresses can be changed without introducing an entire new PSW by switching from real mode, primary-space mode, or secondary-space mode to a different mode, or by changing one or more of the translation parameters in control registers 1 and 7. The early specification exception can be caused by executing STORE THEN OR SYSTEM MASK or SET SYSTEM MASK, which switches to or from real mode while introducing invalid values in bit positions 0-7 of the PSW.

   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.

   For situations c, d, and e, 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 f, the instruction address in the PSW has not been replaced, but the corresponding

LzLabs GmbH. Ex. 1016-145

real address after the change may be different.

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.

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, 17, 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 the execution of LOAD PSW or an interruption 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 the section "ILC on Instruction-Fetching Exceptions" in this chapter.

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.

Programming Notes

1. The execution of LOAD ADDRESS SPACE PARAMETERS, LOAD PSW, PROGRAM CALL, PROGRAM TRANSFER, 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

LzLabs GmbH. Ex. 1016-146

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 the section "Priority of Interruptions" in this chapter.)

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 location 24 and a new PSW to be fetched from real location 88.

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 service-signal interruption, a 32-bit parameter is associated with the interruption and is stored at real locations 128-131. Bit 2 of the external-interruption code indicates that a parameter has been stored. When bit 2 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
Malfunction alert
Emergency signal
External call
TOD-clock sync check
Clock comparator
CPU timer
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

LzLabs GmbH. Ex. 1016-147

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 issued the SIGNAL PROCESSOR instruction is stored at real locations 132-133.

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 issued 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.

LzLabs GmbH. Ex. 1016-148

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 when the service processor signals completion of an operation previously initiated by the program. The program initiates functions at the service processor 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 accept the interruption. The pending request is cleared when it causes an interruption in any one of the CPUs and also by subsystem reset.

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 the rightmost 32 bits of the clocks are not running in synchronism.

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 32-63 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 the rightmost 32 bits 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.

INPUT/OUTPUT 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.

LzLabs GmbH. Ex. 1016-149

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, for example 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 interruption-subclass queues is determined on the basis of the interruption-subclass code (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 location 56 and a new PSW to be fetched from real location 120. 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 register 6 determine whether the CPU is enabled for a particular I/O interruption.

I/O interruptions are grouped into eight I/O-interruption-subclass queues numbered from 0-7. Each interruption-subclass queue has an associated I/O-interruption-subclass mask in bit positions 0-7 of control register 6. Each subchannel has an I/O-interruption-subclass-code 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 in control register 6 is also one. If the corresponding interruption-subclass mask 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 location 48 and a new PSW to be fetched from real location 112.

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 location 40 and a new PSW to be fetched from real location 104.

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 144-159.

Except for the PER-event condition, 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.

The PER-event condition is indicated by setting bit 8 of the interruption code to one, with bits 0-7 set to zeros. When this is the only condition, bits 9-15 are also set to zeros. When a PER-event condition is indicated concurrently with another program interruption condition, bit 8 is one, and the coded

LzLabs GmbH. Ex. 1016-150

value for the other condition appears in bit positions 9-15.

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, 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 the section "Priority of Interruptions" in this chapter for a description of how such strings are terminated.

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.

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 entry. The table entries to which the rule applies are entries for the segment table, page table, linkage table, entry table, and authority 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. 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 the figure "Summary of Action for Addressing and Protection Exceptions."

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, designating 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. 1016-151

Exception	Action on		
	Table-Entry Fetch[1]	Instruction Fetch	Operand Reference
Addressing exception	Suppress	Suppress	Suppress for IPTE, LASP, LPSW, MSCH, SCKC, SPT, SPX, SSCH, SSM, STCRW, STNSM, STOSM, TPI, TPROT, and Tracing[2]. Terminate for all others.[3]
Protection exception for key-controlled protection	--	Suppress	Suppress for IPTE, LASP, LPSW, MSCH, SCKC, SPT, SPX, SSCH, STNSM, STOSM, and TPI[4]. Terminate for all others.[3]
Protection exception for page protection	--	--	Suppress for STCRW, STNSM, STOSM, and TPI[4]. Terminate for all others.[3]
Protection exception for low-address protection	--	--	Suppress for PC, PT, SSAR, STCRW, STNSM, STOSM, TPI[4], and Tracing[2]. Terminate for all others.[3]

Explanation:

-- Not applicable

1 Table entries include segment table, page table, linkage table, entry table, ASN first table, ASN second table, and authority table.

2 The following instructions cause an entry to be made in the trace table when the corresponding tracing function is turned on: BALR, BASR, BASSM, PC, PT, SSAR, and TRACE. The stores into the trace table are subject to addressing and low-address-protection exceptions. The operation is suppressed for these exceptions.

3 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 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 operation is suppressed.

4 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.

Summary of Action for Addressing and Protection Exceptions

LzLabs GmbH. Ex. 1016-152

AFX-Translation Exception

An AFX-translation exception is recognized when, during ASN translation in PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), or SET SECONDARY ASN with space switching (SSAR-ss), bit 0 of the ASN-first-table entry used is not zero.

The ASN being translated 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 AFX-translation exception is indicated by a program-interruption code of 0020 hex (or 00A0 hex if a concurrent PER event is indicated).

ASN-Translation-Specification Exception

An ASN-translation-specification exception is recognized during ASN translation in LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), or SET SECONDARY ASN with space switching (SSAR-ss) when either:

1. Bit positions 28-31 of a valid ASN-first-table entry do not contain zeros.
2. Bit positions 30, 31, and 60-63 of a valid ASN-second-table entry do not contain zeros.

The operation is suppressed.

The instruction-length code is 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).

ASX-Translation Exception

An ASX-translation exception is recognized when, during ASN translation in PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), or SET SECONDARY ASN with space switching (SSAR-ss), bit 0 of the ASN-second-table entry used is not zero.

The ASN being translated 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 ASX-translation exception is indicated by a program-interruption code of 0021 hex (or 00A1 hex if a concurrent PER event is indicated).

Data Exception

A data exception is recognized when any of the following is true:

1. The sign or digit codes of operands in the decimal instructions (described in Chapter 8, "Decimal Instructions") or in CONVERT TO BINARY 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.
3. The multiplicand in MULTIPLY DECIMAL has an insufficient number of high-order zeros.

The action taken for a 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.

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 data exception for an invalid digit code.

The instruction-length code is 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).

LzLabs GmbH. Ex. 1016-153

Programming Notes

1. The definition for 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 data exception occurs, for all fields to appear valid.

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 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 appears in a position specified by the instruction to be checked for a valid sign. (This condition excludes the first operand of ZERO AND ADD, both operands of EDIT, and EDIT AND MARK.)

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).

Exponent-Overflow Exception

An exponent-overflow exception is recognized when the result characteristic of a floating-point 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 or 2.

The exponent-overflow exception is indicated by a program-interruption code of 000C hex (or 008C hex if a concurrent PER event is indicated).

Exponent-Underflow Exception

An exponent-underflow exception is recognized when the result characteristic of a floating-point operation is less than zero and the result fraction is not zero. For an extended-format floating-point result, exponent underflow is indicated only when the high-order characteristic underflows.

LzLabs GmbH. Ex. 1016-154

The interruption may be disallowed by the exponent-underflow mask (PSW bit 22).

The operation is completed. The 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 or 2.

The exponent-underflow exception is indicated by a program-interruption code of 000D hex (or 008D 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 indicated by the entry-table-index part of the PC number is beyond the length of the entry table as designated by the linkage-table entry.

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 EX-translation exception is indicated by a program-interruption code of 0023 hex (or 00A3 hex if a concurrent PER event is indicated).

Fixed-Point-Divide Exception

A fixed-point-divide exception is recognized when in signed binary division the divisor is zero or when the quotient in signed binary division or 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 0008 hex (or 0088 hex if a concurrent PER event is indicated).

Floating-Point-Divide Exception

A floating-point-divide exception is recognized when in floating-point division the divisor has a zero fraction.

The operation is suppressed.

The instruction-length code is 1 or 2.

The floating-point-divide exception is indicated by a program-interruption code of 000F hex (or 008F 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 indicated by the linkage-table-index part of the PC number is beyond the length of the linkage table as designated by control register 5.
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

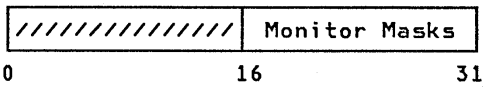
LzLabs GmbH. Ex. 1016-155

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.

Control Register 8:

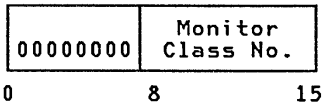


The monitor-mask bits, bits 16-31 of control register 8, correspond to monitor classes 0-15, respectively. Any number of monitor-mask bits may be on at a time; together they specify the classes of monitor events that are monitored at that time. The mask bits are initialized to zero.

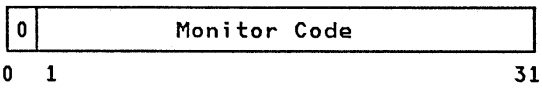
When MONITOR CALL is executed and the corresponding monitor-mask bit is one, a program interruption for monitor event occurs.

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 24-bit mode, bits 0-7 of the address are zeros, while in 31-bit mode, bit 0 is zero.

The operation is completed.

The instruction-length code is 2.

The monitor exception is indicated by a program-interruption code of 0040 hex (or 00C0 hex if a concurrent PER event is indicated).

Operand Exception

An invalid operand has been encountered in a CLEAR SUBCHANNEL, HALT SUBCHANNEL, MODIFY SUBCHANNEL, RESET CHANNEL PATH, RESUME SUBCHANNEL, SET ADDRESS LIMIT, SET CHANNEL MONITOR, START SUBCHANNEL, STORE SUBCHANNEL, or TEST SUBCHANNEL instruction. For details, see the section "Unusual Conditions" in the individual instruction descriptions in Chapter 14, "I/O Instructions."

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.

For the purpose of checking the operation code of an instruction, the operation code is defined as follows:

1. When the first eight bits of an instruction have the value B2 or E5 hex, the first 16 bits form the operation code.
2. In all other cases, the first eight bits alone form the operation code.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

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

LzLabs GmbH. Ex. 1016-156

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 issued 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

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 the instruction that needs the page-table entry in the translation of either 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.

The segment-index and page-index portion of the virtual address causing the exception is stored at real locations 144-147. Bit 0 of real location 144 is set to zero if the virtual address was relative to the primary address space, or it is set to one if the virtual address was relative to the secondary address space. The rightmost 12 bits of the address stored are unpredictable.

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 a reference to 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 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 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 by the contents of control registers 9-11 and one or more of these events occur.

The interruption may be disallowed by PSW bit 1.

The unit of operation is completed, unless another condition has caused the unit of operation to be nullified, suppressed, or terminated.

Additional information identifying the event is stored at real locations 150-155. See the section "Program-Event Recording" in Chapter 4, "Control," for a detailed description of the interruption condition.

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.

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 length of the authority table

LzLabs GmbH. Ex. 1016-157

designated by the ASN-second-table entry.

2. The primary-authority bit indicated by the authorization index is zero.

The ASN being translated 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 problem state, one, to 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 of the instructions MOVE TO PRIMARY and MOVE TO SECONDARY 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.

The operation is suppressed.

The instruction-length code is 1, 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. 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.

3. 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 largest instruction of EXECUTE is protected against fetching.

Except for 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 the figure "Summary of Action for Protection and Addressing Exceptions," which is included in the section "Addressing Exception" in this chapter.

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 designated 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

LzLabs GmbH. Ex. 1016-158

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.

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, designating 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 (SSAR-ss) when either:

1. The authority-table entry indicated by the authorization index in control register 4 is beyond the length of the authority table designated by the ASN-second-table entry.
2. The secondary-authority bit indicated by the authorization index is zero.

The ASN being translated 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 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 the instruction that needs the segment-table entry in the translation of either the 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.

The segment-index and page-index portion of the virtual address causing the exception is stored at real locations 144-147. Bit 0 of real location 144 is set to zero if the virtual address was relative to the primary address space, or it is set to one if the virtual address was relative to the secondary address space. The rightmost 12 bits of the address stored are unpredictable.

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).

Significance Exception

A significance exception is recognized when the result fraction in floating-point addition or subtraction is zero.

The interruption may be disallowed by the significance mask (PSW bit 23).

The operation is completed. The 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 significance exception is indicated by a program-interruption code of 000E hex (or 00AE hex if a concurrent PER event is indicated).

Space-Switch Event

A space-switch event is recognized at the completion of a PROGRAM CALL with

LzLabs GmbH. Ex. 1016-159

space switching (PC-ss) or a PROGRAM TRANSFER with space switching (PT-ss) when any of the following is true:

1. The space-switch-event-control bit, bit 0 of control register 1, is one before the operation.

2. The space-switch-event-control bit is one after the operation.

3. A PER event is reported.

The old PASN, which appears in the right half of control register 4 before the execution of the instruction PC-ss or PT-ss, is stored at real locations 146-147. The old 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.

The operation is completed.

The instruction-length code is 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 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 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 either a PC-ss or a PT-ss.

4. The occurrence of a space-switch event at the completion of a PC-ss or PT-ss 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
   MOVE TO PRIMARY
   MOVE TO SECONDARY
   SET ADDRESS SPACE CONTROL
   SET SECONDARY ASN

3. Execution of PROGRAM CALL or PROGRAM TRANSFER is attempted, and the CPU is not in primary-space mode.

4. 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, and the ASN-translation control, bit 12 of control register 14, is zero.

5. Execution of PROGRAM CALL or PROGRAM TRANSFER is attempted and, the subsystem-linkage control, bit 0 of control register 5, is zero.

6. 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 operation is suppressed.

The instruction-length code is 1, 2, or 3, and indicates the length of the instruction causing the exception.

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. The PSW contains an odd instruction address.

LzLabs GmbH. Ex. 1016-160

2. An operand address does not designate an integral boundary in an instruction requiring such integral-boundary designation.

3. The storage address in INSERT STORAGE KEY or SET STORAGE KEY does not have zeros in the four low-order bit positions.

4. An odd-numbered general register is designated by an R field of an instruction that requires an even-numbered register designation.

5. A floating-point register other than 0, 2, 4, or 6 is specified for a short or long operand, or a floating-point register other than 0 or 4 is specified for an extended operand.

6. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.

7. 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.

8. Bit positions 8-11 of MONITOR CALL do not contain zeros.

9. A one is introduced into an unassigned bit position of the PSW (that is, any of bit positions 0, 2-4, 17, or 24-31). This is handled as an early PSW specification exception.

10. A zero is introduced into bit position 12 of the PSW. This is handled as an early PSW specification exception.

11. 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.

12. Bits 20-22 of the second-operand address of SET ADDRESS SPACE CONTROL are not all zeros.

13. The addressing bit in the general register specified by the R_2 field of PROGRAM TRANSFER is zero, but the high-order seven bits of the instruction address in the same register are not all zeros.

The execution of the instruction identified by the old PSW is suppressed. However, for early PSW specification exceptions (causes 9-11), 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, designating the length of the instruction causing the exception.

When the instruction address is odd (cause 1), it is unpredictable whether the ILC is 1, 2, or 3.

When the exception is recognized because of an early PSW specification exception, and the exception has been introduced by LOAD PSW or an interruption, the ILC is 0. 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 the section "Exceptions Associated with the PSW" in this chapter for a definition of when the exceptions associated with the PSW are recognized.

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.

LzLabs GmbH. Ex. 1016-161

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.

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 and 3 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 the address of the instruction, as the instruction address in the old PSW has been incremented by the amount specified 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.

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. Three collective names are used:

Access exceptions
ASN-translation exceptions
Trace exceptions

The individual exceptions and their priorities are listed in the section "Multiple-Program-Interruption Conditions" in this chapter.

RECOGNITION OF ACCESS EXCEPTIONS

The figure "Handling of Access Exceptions" summarizes the conditions that can cause access exceptions and the action taken when they are encountered.

LzLabs GmbH. Ex. 1016-162

Condition	Translation for Virtual Address of LRA		Translation and Access for Logical Address of TPROT		Translation and Access for Any Other Address	
	Indication	Action	Indication	Action	Indication	Action
Control-register-0 contents[1]						
Invalid encoding of bits 8-12	TS	Suppress	-[4]	-[4]	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[2]	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[2]	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[3]	Complete	P	Term.*
Invalid address	-	-	A	Suppress	A	Term.*

Explanation:

- TS Translation-specification exception.
- ST Segment-translation exception.
- PT Page-translation exception.
- A Addressing exception.
- P Protection exception.
- cc1 Condition code 1 set.
- cc2 Condition code 2 set.
- cc3 Condition code 3 set.
- \- The condition does not apply.
- * Action is to terminate except where otherwise specified in this publication.
- 1 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.
- 2 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.
- 3 The condition code is set as follows:
  - 0 Operand location not protected
  - 1 Fetches permitted, but stores not permitted
  - 2 Neither fetches or stores permitted
- 4 A translation specification exception cannot occur for the logical address of TEST PROTECTION since this exception would have been recognized during the instruction fetch for the instruction.

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 fetch from an unavailable location or detects some other access-exception condition, but a branch instruction or an interruption

LzLabs GmbH. Ex. 1016-163

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.

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 by 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.

The figure "Priority of Program-Interruption Conditions" 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."

The figure "Priority of Access Exceptions" 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 the figures "Priority of ASN-Translation Exceptions" and "Priority of Trace Exceptions," 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

LzLabs GmbH. Ex. 1016-164

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:

Priority Number	Exception
7.B	Access exceptions for third instruction halfword.
8.B	Access exceptions (operand 1)
8.E	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≠E) is between alphabetic characters, and it is unpredictable whether the protection exception or the data exception would be indicated.

LzLabs GmbH. Ex. 1016-165

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	Operation exception.
7.C.2	Privileged-operation exception for privileged instructions.
7.C.3	Execute exception.
7.C.4	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.	Fixed-point divide, floating-point divide, operand, and conditions, other than PER events, which result in completion. Either these conditions are mutually exclusive or their priority is specified in the corresponding definitions.

Priority of Program-Interruption Conditions (Part 1 of 2)

LzLabs GmbH. Ex. 1016-166

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, 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 is 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 and COMPARE LOGICAL LONG, an access exception for a particular operand can be indicated only if the R field for that operand designates an even-numbered register.

6 The 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.

Priority of Program-Interruption Conditions (Part 2 of 2)

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, with DAT on, the exceptions are:

1. Translation specification
2. Segment translation
3. Page translation
4. Addressing (the DAT tables and the operand or instruction)
5. Protection (key-controlled, page, and low-address)

With DAT off, the exceptions are:

1. Addressing (the operand or instruction)
2. Protection (key-controlled and low-address)

LzLabs GmbH. Ex. 1016-167

A.	Protection exception (low-address protection) due to a store-type operand reference with an effective address in the range 0-511.
B.1.	Translation-specification exception due to invalid encoding of bits 8-12 of control register 0.[1]
B.2.	Segment-translation exception due to segment-table entry being outside table.[2]
B.3.	Addressing exception for access to segment-table entry.[3]
B.4.	Segment-translation exception due to I bit in segment-table entry having the value one.[2]
B.5.	Translation-specification exception due to invalid ones in segment-table entry.[3]
B.6.	Page-translation exception due to page-table entry being outside table.[2]
B.7.	Addressing exception for access to page-table entry.[1]
B.8.	Page-translation exception due to I bit in page-table entry having the value one.[2]
B.9.	Translation-specification exception due to invalid ones in page-table entry.[3]
B.10.A	Protection exception (page protection) due to a store-type operand reference to a virtual address which is protected against stores.[4]
B.10.B	Addressing exception for access to instruction or operand.
B.11.	Protection exception (key-controlled protection) due to attempt to access a protected instruction or operand location.

Explanation:

1. Not applicable when DAT is off, except for execution of INVALIDATE PAGE TABLE ENTRY and for translation of operand address of LOAD REAL ADDRESS.
2. Not applicable when DAT is off; not applicable to operand addresses for LOAD REAL ADDRESS and TEST PROTECTION.
3. Not applicable when DAT is off except for translation of operand address for LOAD REAL ADDRESS.
4. Not applicable when DAT is off.

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 CALL, PROGRAM TRANSFER, and SET SECONDARY ASN. The exceptions and the priority in which they are detected are shown in the figure "Priority of ASN-Translation Exceptions."

LzLabs GmbH. Ex. 1016-168

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 having the value one.
3.	ASN-translation-specification exception due to invalid ones (bits 28-31) in ASN-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 having the value one.
6.	ASN-translation-specification exception due to invalid ones (bits 30, 31, 60-63) in ASN-second-table entry.

Priority of ASN-Translation 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 the figure "Priority of Trace Exceptions."

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.

Priority of Trace Exceptions

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, specifying 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. In a multiprocessing configuration, the operation can also be initiated at the addressed CPU by issuing a SIGNAL PROCESSOR instruction which specifies the restart order.

When the rate control is set to instruction step, 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

LzLabs GmbH. Ex. 1016-169

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
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 a one in an unassigned bit position or if the high-order 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.

LzLabs GmbH. Ex. 1016-170

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. 1016-171

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 7-3
  Unsigned Binary Arithmetic 7-3
Signed and Logical Comparison 7-4
Instructions 7-4
  ADD 7-8
  ADD HALFWORD 7-8
  ADD LOGICAL 7-9
  AND 7-9
  BRANCH AND LINK 7-10
  BRANCH AND SAVE 7-10
  BRANCH AND SAVE AND SET MODE 7-11
  BRANCH AND SET MODE 7-11
  BRANCH ON CONDITION 7-12
  BRANCH ON COUNT 7-13
  BRANCH ON INDEX HIGH 7-13
  BRANCH ON INDEX LOW OR EQUAL 7-13
  COMPARE 7-14
  COMPARE AND SWAP 7-14
  COMPARE DOUBLE AND SWAP 7-14
  COMPARE HALFWORD 7-16
  COMPARE LOGICAL 7-16
  COMPARE LOGICAL CHARACTERS UNDER MASK 7-17
  COMPARE LOGICAL LONG 7-17
  CONVERT TO BINARY 7-19
  CONVERT TO DECIMAL 7-20
  DIVIDE 7-20
  EXCLUSIVE OR 7-21
  EXECUTE 7-22
  INSERT CHARACTER 7-23
  INSERT CHARACTERS UNDER MASK 7-23
  INSERT PROGRAM MASK 7-23
  LOAD 7-24
  LOAD ADDRESS 7-24
  LOAD AND TEST 7-24
  LOAD COMPLEMENT 7-24
  LOAD HALFWORD 7-25
  LOAD MULTIPLE 7-25
  LOAD NEGATIVE 7-25
  LOAD POSITIVE 7-26
  MONITOR CALL 7-26
  MOVE 7-27
  MOVE LONG 7-27
  MOVE NUMERICS 7-30
  MOVE WITH OFFSET 7-31
  MOVE ZONES 7-31
  MULTIPLY 7-32
  MULTIPLY HALFWORD 7-32
  OR 7-33
  PACK 7-33
  SET PROGRAM MASK 7-34
  SHIFT LEFT DOUBLE 7-34
  SHIFT LEFT DOUBLE LOGICAL 7-35
  SHIFT LEFT SINGLE 7-35
  SHIFT LEFT SINGLE LOGICAL 7-36
  SHIFT RIGHT DOUBLE 7-36
  SHIFT RIGHT DOUBLE LOGICAL 7-36
  SHIFT RIGHT SINGLE 7-37
  SHIFT RIGHT SINGLE LOGICAL 7-37
  STORE 7-37
  STORE CHARACTER 7-38
  STORE CHARACTERS UNDER MASK 7-38
  STORE CLOCK 7-38
  STORE HALFWORD 7-39

LzLabs GmbH. Ex. 1016-173

STORE MULTIPLE 7-39
SUBTRACT 7-40
SUBTRACT HALFWORD 7-40
SUBTRACT LOGICAL 7-40
SUPERVISOR CALL 7-41
TEST AND SET 7-41
TEST UNDER MASK 7-42
TRANSLATE 7-42
TRANSLATE AND TEST 7-43
UNPACK 7-44

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-to-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.

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.

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

LzLabs GmbH. Ex. 1016-174

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.

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 leftmost 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 the

LzLabs GmbH. Ex. 1016-175

section "Address Generation" in Chapter 5, "Program Execution.") 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 program multiple-precision arithmetic. Thus, for multiple-precision binary-integer addition, ADD LOGICAL can be used to add the corresponding lower-order parts of the operands. If the condition code indicates a carry, a value of one should be added to the sum of the next pair of integers to the left. If the integers are signed, ADD should be used on the leftmost pair. 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 integers 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.

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 the figure "Summary of General Instructions." The figure also indicates when the condition code is set 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 the section "Instructions" in Chapter 5, "Program Execution." 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 Note

The general instructions in the 370-XA mode differ from that provided in the System/370 mode in that

LzLabs GmbH. Ex. 1016-176

(1) conditional-swapping and branch-and-save facilities which are optional in the System/370 mode are part of the standard instruction set in the 370-XA mode, (2) MOVE INVERSE is not offered by any machine in the 370-XA mode, and (3) the following additional general instructions are available in the 370-XA mode:

BRANCH AND SAVE AND SET MODE
BRANCH AND SET MODE
INSERT PROGRAM MASK

In general, bimodal addressing affects the general instructions only in the manner in which logical storage addresses are handled. The instructions BRANCH AND LINK (BAL, BALR), COMPARE LOGICAL LONG, LOAD ADDRESS, MOVE LONG, and TRANSLATE AND TEST are affected in that the leftmost byte of the results in registers is handled differently in the two modes. Otherwise, the general instructions are executed the same way in both the 24-bit and 31-bit addressing modes.

LzLabs GmbH. Ex. 1016-177

Name	Mne-monic	Characteristics					Op Code
ADD	AR	RR C		IF		R	1A
ADD	A	RX C	A	IF		R	5A
ADD HALFWORD	AH	RX C	A	IF		R	4A
ADD LOGICAL	ALR	RR C				R	1E
ADD LOGICAL	AL	RX C	A			R	5E
AND	NR	RR C				R	14
AND	N	RX C	A			R	54
AND (character)	NC	SS C	A			ST	D4
AND (immediate)	NI	SI C	A			ST	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			ϕ[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
COMPARE	CR	RR C					19
COMPARE	C	RX C	A				59
COMPARE AND SWAP	CS	RS C	A SP		$	R ST	BA
COMPARE DOUBLE AND SWAP	CDS	RS C	A SP		$	R ST	BB
COMPARE HALFWORD	CH	RX C	A				49
COMPARE LOGICAL	CLR	RR C					15
COMPARE LOGICAL	CL	RX C	A				55
COMPARE LOGICAL (character)	CLC	SS C	A				D5
COMPARE LOGICAL (immediate)	CLI	SI C	A				95
COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS C	A				BD
COMPARE LOGICAL LONG	CLCL	RR C	A SP	II		R	0F
CONVERT TO BINARY	CVB	RX	A	D	IK	R	4F
CONVERT TO DECIMAL	CVD	RX	A			ST	4E
DIVIDE	DR	RR	SP		IK	R	1D
DIVIDE	D	RX	A SP		IK	R	5D
EXCLUSIVE OR	XR	RR C				R	17
EXCLUSIVE OR	X	RX C	A			R	57
EXCLUSIVE OR (character)	XC	SS C	A			ST	D7
EXCLUSIVE OR (immediate)	XI	SI C	A			ST	97
EXECUTE	EX	RX	AI SP		EX		44
INSERT CHARACTER	IC	RX	A			R	43
INSERT CHARACTERS UNDER MASK	ICM	RS C	A			R	BF
INSERT PROGRAM MASK	IPM	RRE				R	B222
LOAD	LR	RR				R	18
LOAD	L	RX	A			R	58
LOAD ADDRESS	LA	RX				R	41
LOAD AND TEST	LTR	RR C				R	12
LOAD COMPLEMENT	LCR	RR C		IF		R	13
LOAD HALFWORD	LH	RX	A			R	48
LOAD MULTIPLE	LM	RS	A			R	98
LOAD NEGATIVE	LNR	RR C				R	11
LOAD POSITIVE	LPR	RR C		IF		R	10
MONITOR CALL	MC	SI	SP		MO		AF
MOVE (character)	MVC	SS	A			ST	D2

Summary of General Instructions (Part 1 of 3)

LzLabs GmbH. Ex. 1016-178

Name	Mnemonic	Characteristics						Op Code
MOVE (immediate)	MVI	SI		A			ST	92
MOVE LONG	MVCL	RR	C	A	SP	II	R ST	0E
MOVE NUMERICS	MVN	SS		A			ST	D1
MOVE WITH OFFSET	MVO	SS		A			ST	F1
MOVE ZONES	MVZ	SS		A			ST	D3
MULTIPLY	MR	RR			SP		R	1C
MULTIPLY	M	RX		A	SP		R	5C
MULTIPLY HALFWORD	MH	RX		A			R	4C
OR	OR	RR	C				R	16
OR	O	RX	C	A			R	56
OR (character)	OC	SS	C	A			ST	D6
OR (immediate)	OI	SI	C	A			ST	96
PACK	PACK	SS		A			ST	F2
SET PROGRAM MASK	SPM	RR	L					04
SHIFT LEFT DOUBLE	SLDA	RS	C		SP	IF	R	8F
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS			SP		R	8D
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	50
STORE CHARACTER	STC	RX		A			ST	42
STORE CHARACTERS UNDER MASK	STCM	RS		A			ST	BE
STORE CLOCK	STCK	S	C	A		$	ST	B205
STORE HALFWORD	STH	RX		A			ST	40
STORE MULTIPLE	STM	RS		A			ST	90
SUBTRACT	SR	RR	C			IF	R	1B
SUBTRACT	S	RX	C	A		IF	R	5B
SUBTRACT HALFWORD	SH	RX	C	A		IF	R	4B
SUBTRACT LOGICAL	SLR	RR	C				R	1F
SUBTRACT LOGICAL	SL	RX	C	A			R	5F
SUPERVISOR CALL	SVC	RR				¢		0A
TEST AND SET	TS	S	C	A		$	ST	93
TEST UNDER MASK	TM	SI	C	A				91
TRANSLATE	TR	SS		A			ST	DC
TRANSLATE AND TEST	TRT	SS	C	A		GM	R	DD
UNPACK	UNPK	SS		A			ST	F3

Summary of General Instructions (Part 2 of 3)

LzLabs GmbH. Ex. 1016-179

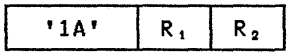
Explanation:

¢	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
AI	Access exceptions for instruction address
B	PER branch event
C	Condition code is set
D	Data exception
EX	Execute exception
GM	Instruction execution includes the implied use of general registers 1 and 2
IF	Fixed-point-overflow exception
II	Interruptible instruction
IK	Fixed-point-divide exception
L	New condition code loaded
MO	Monitor event
R	PER general-register-alteration event
RR	RR instruction format
RRE	RRE instruction format
RS	RS instruction format
RX	RX instruction format
S	S instruction format
SI	SI instruction format
SP	Specification exception
SS	SS instruction format
ST	PER storage-alteration event
T	Trace exceptions (includes trace table, addressing, and low-address protection)

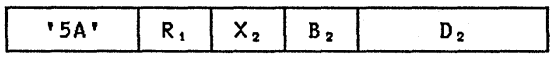
Summary of General Instructions (Part 3 of 3)

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 in 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:

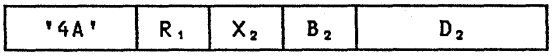
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Exceptions:

Access (fetch, operand 2 of A only)
Fixed-point overflow

ADD HALFWORD

AH $R_1,D_2(X_2,B_2)$ [RX]



The second operand is added to the first operand, and the sum is placed in 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.

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.

LzLabs GmbH. Ex. 1016-180

Resulting Condition Code:

0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Exceptions:

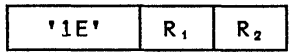
Access (fetch, operand 2)
Fixed-point overflow

Programming Note

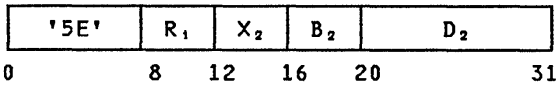
An example of the use of ADD HALFWORD is given in Appendix A.

ADD LOGICAL

ALR R_1,R_2 [RR]



AL $R_1,D_2(X_2,B_2)$ [RX]



The second operand is added to the first operand, and the sum is placed in the first-operand location. The operands and the sum are treated as 32-bit unsigned binary integers.

Resulting Condition Code:

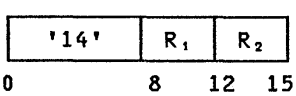
0 Sum is zero, with no carry
1 Sum is not zero, with no carry
2 Sum is zero, with carry
3 Sum is not zero, with carry

Program Exceptions:

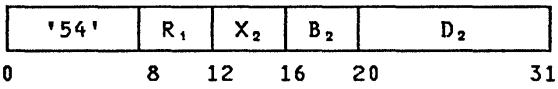
Access (fetch, operand 2 of AL only)

AND

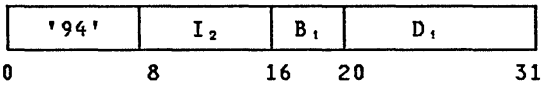
NR R_1,R_2 [RR]



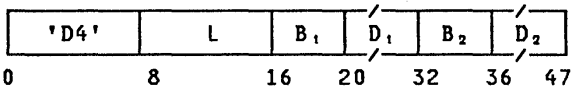
N $R_1,D_2(X_2,B_2)$ [RX]



NI $D_1(B_1),I_2$ [SI]



NC $D_1(L,B_1),D_2(B_2)$ [SS]



The AND of the first and second operands is placed in the first-operand location.

The connective AND is applied to the operands bit by bit. A bit position in the result is 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 is zero
1 Result is not zero
2 --
3 --

Program Exceptions:

Access (fetch, operand 2, N and NC; fetch and store, operand 1, NI and NC)

Programming Notes

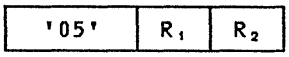
1. An example of the use of AND is given in Appendix A.
2. The instruction AND may be used to set a bit to zero.
3. Accesses to the first operand of AND (NI) and AND (NC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and

LzLabs GmbH. Ex. 1016-181

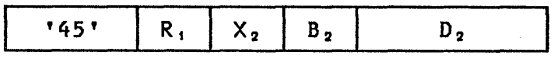
store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, the instruction AND cannot be safely used to update a location in storage if the possibility exists that another CPU or channel program may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

BRANCH AND LINK

BALR R_1,R_2 [RR]



BAL $R_1,D_2(X_2,B_2)$ [RX]



Information from the current PSW, including the updated instruction address, is loaded as link information in the general register designated by R_1. Subsequently, the instruction address is replaced by the branch address.

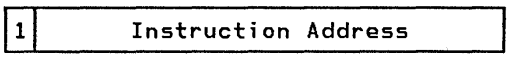
In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of the general register designated by R_2 are used to generate the branch address; however, when the R_2 field is zero, the operation is performed without branching. The branch address is computed before the register designated by R_1 is changed.

The link information in 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.

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:

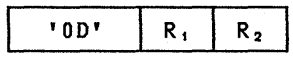
Trace (R_2 field nonzero, BALR only)

Programming Notes

1. An example of the use of the BRANCH AND LINK instruction is given in Appendix A.
2. When the R_2 field in the RR format is zero, the link information is loaded without branching.
3. The BRANCH AND LINK instructions (BAL and BALR) are provided in the 370-XA mode for compatibility purposes. It is recommended that, where possible, the BRANCH AND SAVE instructions (BAS and BASR) be used and BAL and BALR avoided, since the latter place nonzero information in bit positions 0-7 of the link register in 24-bit addressing mode, which may lead to problems. Additionally, BAL and BALR may be slower than BAS and BASR because BAS and BASR always save the right half of the PSW, and BAL and BALR, which do not, may require additional time to test the addressing mode, and even more time, if 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.

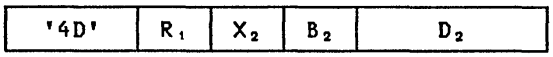
BRANCH AND SAVE

BASR R_1,R_2 [RR]



LzLabs GmbH. Ex. 1016-182

BAS $R_1,D_2(X_2,B_2)$ [RX]



Bits 32-63 of the current PSW, including the updated instruction address, is saved as link information in the general register designated by R_1. 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 the general register designated by 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 the register designated by R_1 is changed.

Condition Code: The code remains unchanged.

Program Exceptions:

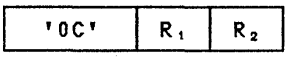
Trace (R_2 field nonzero, BASR only)

Programming Note

The BRANCH AND SAVE instructions (BAS and BASR) are intended as instructions 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 instructions (BAL and BALR). See the programming notes at the end of the section "Subroutine Linkage" in Chapter 5, "Program Execution," 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 AND SAVE instructions.

BRANCH AND SAVE AND SET MODE

BASSM R_1,R_2 [RR]



Bits 32-63 of the current PSW, including the updated instruction address, is saved as link information in the general register designated by R_1. 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.

The contents of the general register designated by the R_2 field specify the new addressing mode and 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 the general register designated by the R_2 field 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 the register designated by R_1 is changed.

Condition Code: The code remains unchanged.

Program Exceptions:

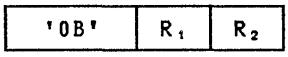
Trace (R_2 field nonzero)

Programming Note

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 at the end of the section "Subroutine Linkage" in Chapter 5, "Program Execution," for a detailed discussion of this and other linkage instructions.

BRANCH AND SET MODE

BSM R_1,R_2 [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 the general register designated by 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 the general register designated by the R_2 field specify the

LzLabs GmbH. Ex. 1016-183

new addressing mode and 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 the general register designated by the R_2 field 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 the register designated by R_1 is changed.

Condition Code: The code remains unchanged.

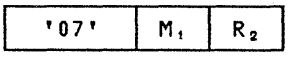
Program Exceptions: None.

Programming Note

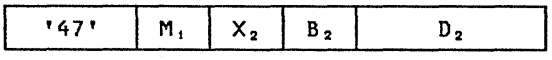
BRANCH AND SET MODE with an R_1 field of zero is intended to be the standard return instruction. BRANCH AND SAVE AND SET MODE with a nonzero R_1 field is intended to be used in a "glue module" to connect old 24-bit programs and new programs which may exploit bimodal addressing. See the programming note at the end of the section "Subroutine Linkage" 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 the general register specified by 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 a checkpoint-synchronization function is performed. CPU operation is delayed until all previous accesses by this CPU to storage have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed.

Condition Code: The code remains unchanged.

Program Exceptions: None.

Programming Notes

1. 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 zero 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.

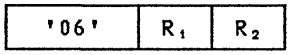
LzLabs GmbH. Ex. 1016-184

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 the checkpoint-synchronization function is actually required.

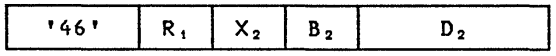
5. Note that the relation between the RR and RX formats in branch-address specification is not the same as in operand-address specification. For branch instructions in the RX format, the branch address is the address specified by X_2, B_2, and D_2; in the RR format, the branch address is contained in the register specified by R_2. For operands, the address specified by X_2, B_2, and D_2 is the operand address, but the register specified by R_2 contains the operand, not the operand address.

BRANCH ON COUNT

BCTR R_1,R_2 [RR]



BCT $R_1,D_2(X_2,B_2)$ [RX]



A one is subtracted from the first operand, and the result is placed in 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 the general register specified by 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 the register designated by R_1 is changed.

Condition Code: The code remains unchanged.

Program Exceptions: None.

Programming Notes

1. An example of the use of BRANCH ON COUNT is given in Appendix A.

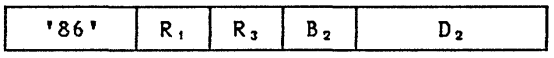
2. The first operand and result can be considered as either signed or unsigned binary integers since the result of a binary subtraction is the same in both cases.

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. Counting is performed without branching when the R_2 field in the RR format contains zero.

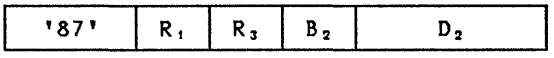
BRANCH ON INDEX HIGH

BXH $R_1,R_3,D_2(B_2)$ [RS]



BRANCH ON INDEX LOW OR EQUAL

BXLE $R_1,R_3,D_2(B_2)$ [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 in 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

LzLabs GmbH. Ex. 1016-185

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 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 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 the register designated by R_1 is changed.

The sum is placed in the first-operand location, regardless of whether the branch is taken.

Condition Code: The code remains unchanged.

Program Exceptions: None.

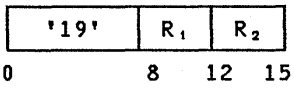
Programming Notes

1. Several examples of the use of BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL are given in Appendix A.

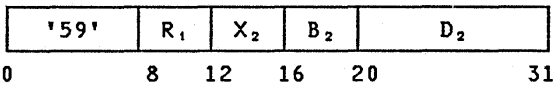
2. The word "index" in the names of these instructions indicates that one of the major purposes is the incrementing and testing of an index value. The increment, being a signed binary integer, may be used to increase or decrease the value in register R_1 by an arbitrary amount.

COMPARE

CR R_1,R_2 [RR]



C $R_1,D_2(X_2,B_2)$ [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:

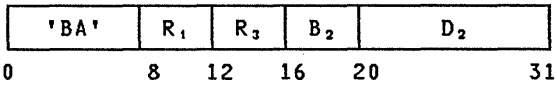
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions:

Access (fetch, operand 2 of C only)

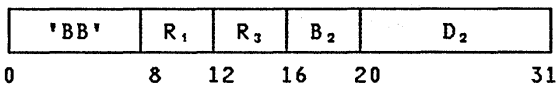
COMPARE AND SWAP

CS $R_1,R_3,D_2(B_2)$ [RS]



COMPARE DOUBLE AND SWAP

CDS $R_1,R_3,D_2(B_2)$ [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 into 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.

LzLabs GmbH. Ex. 1016-186

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 the result of the comparison is unequal, the second-operand location remains unchanged. However, on some models, the value may be fetched and subsequently stored back into the second-operand location. No access by another CPU program to the second-operand location is permitted between the moment that the second operand is fetched for comparison and the moment that it is stored.

When an equal comparison occurs, no access by another CPU program to the second-operand location is permitted between the moment that the second operand is fetched for comparison and the moment that the third operand is stored at the second-operand location.

Serialization is performed before the operand is fetched, and again after the operation is completed. CPU operation is delayed until all previous accesses by this CPU to storage have been completed, as observed by other CPU and channel programs, and then the second operand is fetched. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed, including placing the result value, if any, in storage, as observed by other CPU and channel programs.

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 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 COMPARE AND SWAP and COMPARE DOUBLE AND SWAP are given in Appendix A.

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 program that will update the location or even though the possibility exists 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 condition code 0 is set, the update has been successful. If condition code 1 is set, the storage location no longer contains the original value, the update has not been successful, and the general register designated by the R_1 field of the COMPARE AND SWAP instruction contains the new current value of the storage location. 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

LzLabs GmbH. Ex. 1016-187

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.

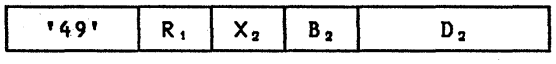
3. COMPARE DOUBLE AND SWAP can be used in a manner similar to that described for COMPARE AND SWAP. In addition, it has another use. Consider a chained list, with a control word used to address the first message in the list, as described in programming note 2b above. If multiple CPU programs are to be permitted to delete messages by using COMPARE AND SWAP (and not just the single 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 program has fetched the address of the most recent message in order to remove the message, another program removes the first two messages and then adds the first message back into the chain. The first 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 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 into which a channel program may store, since the channel-program data may be lost.

5. 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 turned on.

COMPARE HALFWORD

CH $R_1,D_2(X_2,B_2)$ [RX]



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 are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions:

Access (fetch, operand 2)

Programming Note

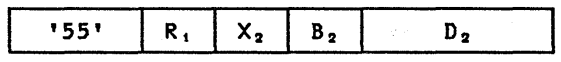
An example of the use of COMPARE HALFWORD is given in Appendix A.

COMPARE LOGICAL

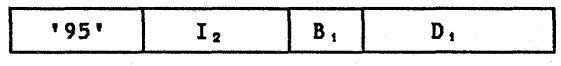
CLR R_1,R_2 [RR]



CL $R_1,D_2(X_2,B_2)$ [RX]

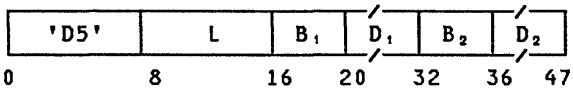


CLI $D_1(B_1),I_2$ [SI]



LzLabs GmbH. Ex. 1016-188

CLC $D_1(L,B_1),D_2(B_2)$ [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 are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions:

Access (fetch, operand 2, CL and CLC; fetch, operand 1, CLI and CLC)

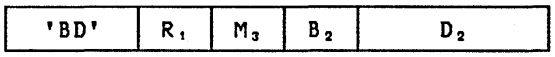
Programming Notes

1. Examples of the use of COMPARE LOGICAL are given in Appendix A.

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.

COMPARE LOGICAL CHARACTERS UNDER MASK

CLM $R_1,M_3,D_2(B_2)$ [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 the general register designated by the R_1 field. 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 Selected bytes are equal, or mask is zero
1 Selected field of first operand is low
2 Selected field of first operand is high
3 --

Program Exceptions:

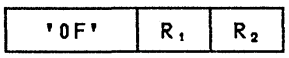
Access (fetch, operand 2)

Programming Note

An example of the use of COMPARE LOGICAL CHARACTERS UNDER MASK is given in Appendix A.

COMPARE LOGICAL LONG

CLCL R_1,R_2 [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 specify 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 the general registers specified by the R_1 and R_2

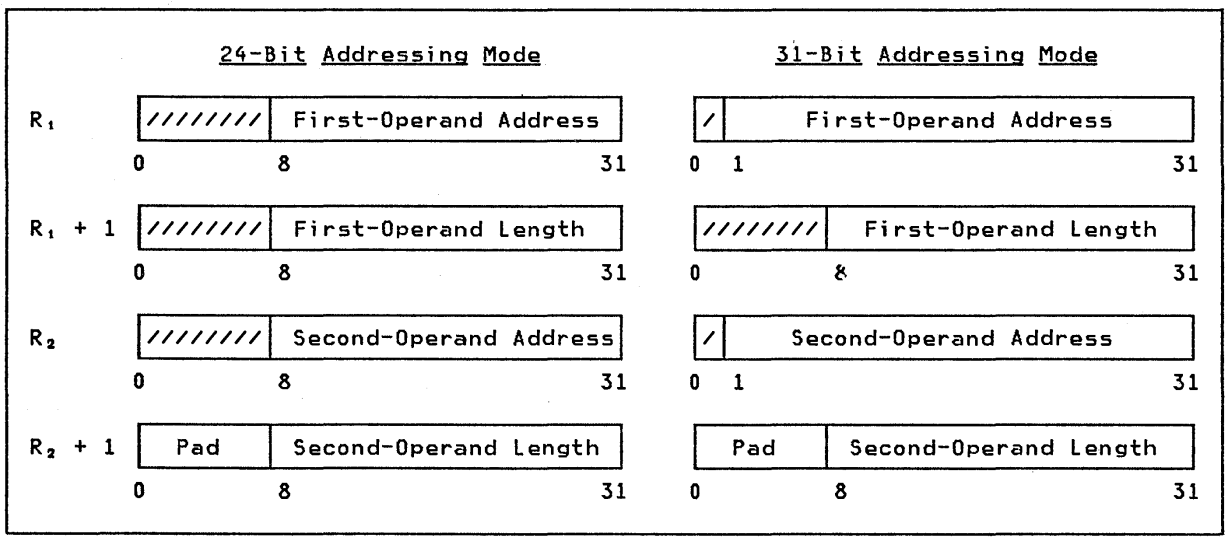
LzLabs GmbH. Ex. 1016-189

fields, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 8-31 of the 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 the general registers specified by the R_1 and R_2 fields 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 the figure "Register Contents for COMPARE LOGICAL LONG."



Register Contents for COMPARE LOGICAL LONG

LzLabs GmbH. Ex. 1016-190

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 causes termination, the contents of registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes compared, and the contents of 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 registers R_1 and R_2 are set to zeros; the contents of bit positions 0-7 of 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 registers R_1 and R_2 at completion identify the first unequal byte in each operand. The lengths in bit positions 8-31 of 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 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 registers R_1 and R_2 are set to zeros, including the case when one or both of the initial length values are zero. The contents of bit positions 0-7 of 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 recognized 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 are equal, or both have zero length
1 First operand is low
2 First operand is high
3 --

Program Exceptions:

Access (fetch, operands 1 and 2)
Specification

Programming Notes

1. An example of the use of COMPARE LOGICAL LONG is given in Appendix A.

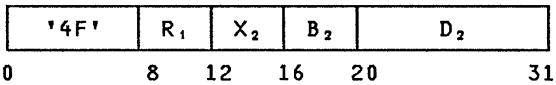
2. When the R_1 and R_2 fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, and, in the absence of dynamic modification of the operand area by another CPU or channel program, condition code 0 is set. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed.

3. Other programming notes concerning interruptible instructions are included in the section "Interruptible Instructions" in Chapter 5, "Program Execution."

4. Special precautions should be taken when COMPARE LOGICAL LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.

CONVERT TO BINARY

CVB $R_1,D_2(X_2,B_2)$ [RX]



The second operand is changed from decimal to binary, and the result is placed in the first-operand location.

The second operand occupies eight bytes in storage and has the format of packed decimal data, as described in Chapter 8,

LzLabs GmbH. Ex. 1016-191

"Decimal Instructions." It is checked for valid sign and digit codes, and a 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 the general register specified by 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:

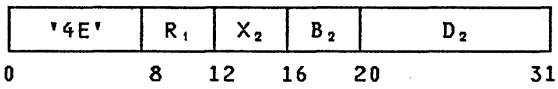
Access (fetch, operand 2)
Data
Fixed-point divide

Programming Notes

1. An example of the use of CONVERT TO BINARY is given in Appendix A.
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 the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

CONVERT TO DECIMAL

CVD $R_1,D_2(X_2,B_2)$ [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 low-order 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:

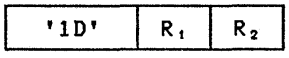
Access (store, operand 2)

Programming Notes

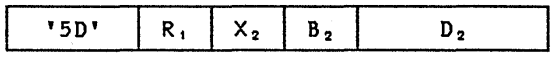
1. An example of the use of CONVERT TO DECIMAL is given in Appendix A.
2. The number to be converted is a 32-bit signed binary integer obtained from a general register. Since 15 decimal digits are available for the result, and the decimal equivalent of 31 bits requires at most 10 decimal digits, an overflow cannot occur.
3. The storage-operand references for CONVERT TO DECIMAL may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

DIVIDE

DR R_1,R_2 [RR]



D $R_1,D_2(X_2,B_2)$ [RX]



The doubleword first operand (the dividend) is divided by the second operand (the divisor), and the remainder and the quotient are placed in the first-operand location.

The R_1 field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R_1 is odd, a specification exception is recognized.

The dividend is treated as a 64-bit signed binary integer. The divisor, the remainder, and the quotient are treated as 32-bit signed binary integers. The remainder and quotient replace the dividend in the pair of registers specified by the R_1 field. The remainder is placed in the even-numbered register,

LzLabs GmbH. Ex. 1016-192

and the quotient is placed in the odd-numbered register.

The sign of the quotient is determined by the rules of algebra. 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.

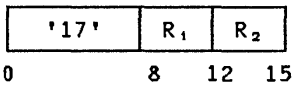
Condition Code: The code remains unchanged.

Program Exceptions:

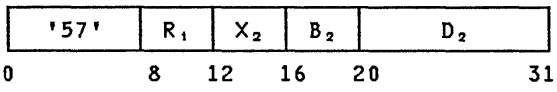
Access (fetch, operand 2 of D only)
Fixed-point divide
Specification

EXCLUSIVE OR

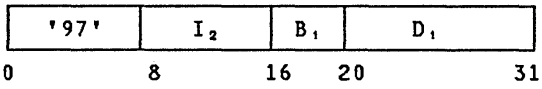
XR R_1,R_2 [RR]



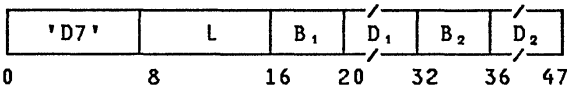
X $R_1,D_2(X_2,B_2)$ [RX]



XI $D_1(B_1),I_2$ [SI]



XC $D_1(L,B_1),D_2(B_2)$ [SS]



The EXCLUSIVE OR of the first and second operands is placed in the first-operand location.

The connective EXCLUSIVE OR is applied to the operands bit by bit. A bit position in the result is set to one if 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 is zero
1 Result is not zero
2 --
3 --

Program Exceptions:

Access (fetch, operand 2, X and XC; fetch and store, operand 1, XI and XC)

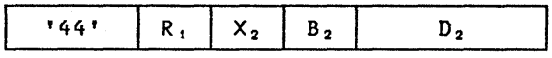
Programming Notes

1. An example of the use of EXCLUSIVE OR is given in Appendix A.
2. EXCLUSIVE OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit 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 channel program may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

LzLabs GmbH. Ex. 1016-193

EXECUTE

EX $R_1,D_2(X_2,B_2)$ [RX]



The single instruction at the second-operand address is modified by the contents of the general register specified by R_1, and the resulting target instruction is executed.

When the R_1 field is not zero, bits 8-15 of the instruction designated by the second-operand address are ORed with bits 24-31 of the register specified by R_1. The ORing does not change either the contents of the register specified by 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 of 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 of 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 as an instruction address rather than a logical address; thus, when the CPU is in secondary-space mode, it is unpredictable whether the target instruction is fetched from the primary space or the secondary space.

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 EXECUTE is given in Appendix A.

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.

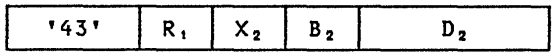
5. When an interruptible instruction is made the target of EXECUTE, the program normally should not designate any register updated by the interruptible instruction as the R_1, X_2, or B_2 register for EXECUTE. Otherwise, on resumption of execution after an interruption, or if the instruction is refetched 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.

6. A program should issue EXECUTE in secondary-space mode only if the virtual address of the target instruction translates to the same real address by means of both the primary segment table and secondary segment table. Otherwise, unpredictable results may occur.

LzLabs GmbH. Ex. 1016-194

INSERT CHARACTER

IC $R_1,D_2(X_2,B_2)$ [RX]



The byte at the second-operand location is inserted into bit positions 24-31 of the general register designated by the R_1 field. The remaining bits in the register remain unchanged.

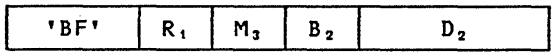
Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)

INSERT CHARACTERS UNDER MASK

ICM $R_1,M_3,D_2(B_2)$ [RS]



Bytes from contiguous locations beginning at the second-operand address are inserted into the first-operand location under control of a mask.

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 the general register designated by the R_1 field. 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 are zeros, or mask is zero
1 Leftmost bit of the inserted field is one
2 Leftmost bit of the inserted field is zero, and not all inserted bits are zeros
3 --

Program Exceptions:

Access (fetch, operand 2)

Programming Notes

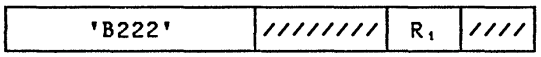
1. Examples of the use of INSERT CHARACTERS UNDER MASK are given in Appendix A.

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 R_1 [RRE]



The condition code and program mask from the current PSW are inserted into bit positions 2-3 and 4-7, respectively, of the general register specified by the R_1 field. Bits 0 and 1 of the register are set to zeros; bits 8-31 are left unchanged.

Bits 16-23 and 28-31 of the instruction are ignored.

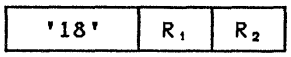
Condition Code: The code remains unchanged.

Program Exceptions: None.

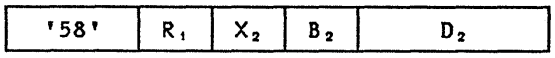
LzLabs GmbH. Ex. 1016-195

LOAD

LR R_1,R_2 [RR]



L $R_1,D_2(X_2,B_2)$ [RX]



The second operand is placed unchanged in the first-operand location.

Condition Code: The code remains unchanged.

Program Exceptions:

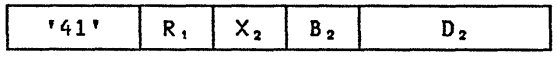
Access (fetch, operand 2 of L only)

Programming Note

An example of the use of LOAD is given in Appendix A.

LOAD ADDRESS

LA $R_1,D_2(X_2,B_2)$ [RX]



The address specified by the X_2, B_2, and D_2 fields is placed in the general register specified by the R_1 field. 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.

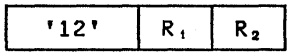
Programming Notes

1. An example of the use of LOAD ADDRESS is given in Appendix A.

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 specified 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 AND TEST

LTR R_1,R_2 [RR]



The second operand is placed unchanged in 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 is zero
1 Result is less than zero
2 Result is 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 R_1,R_2 [RR]



The two's complement of the second operand is placed in the first-operand location. The second operand and result are treated as 32-bit signed binary integers.

LzLabs GmbH. Ex. 1016-196

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 is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:

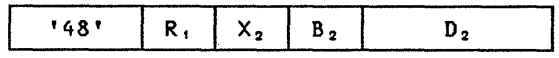
Fixed-point overflow

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 $R_1,D_2(X_2,B_2)$ [RX]



The second operand is considered to be extended to a 32-bit signed binary integer and is placed in the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed binary integer. The second operand is extended to 32 bits by setting each of the 16 leftmost bit positions equal to the sign bit of the storage operand.

Condition Code: The code remains unchanged.

Program Exceptions:

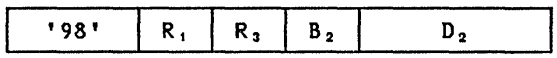
Access (fetch, operand 2)

Programming Note

An example of the use of LOAD HALFWORD is given in Appendix A.

LOAD MULTIPLE

LM $R_1,R_3,D_2(B_2)$ [RS]



The set of general registers starting with the register specified by R_1 and ending with the register specified by 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 the register specified by R_1 and continuing up to and including the register specified by R_3, with register 0 following 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 R_1,R_2 [RR]



The two's complement of the absolute value of the second operand is placed in the first-operand location. The second operand and result are treated as 32-bit signed binary integers.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 --
3 --

Program Exceptions: None.

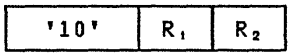
LzLabs GmbH. Ex. 1016-197

Programming Note

The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged.

LOAD POSITIVE

LPR R_1,R_2 [RR]



The absolute value of the second operand is placed in 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 is zero
1 --
2 Result is greater than zero
3 Overflow

Program Exceptions:

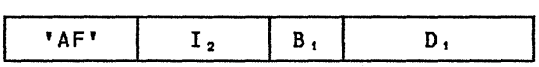
Fixed-point overflow

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.

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 how long a particular program required for execution.

2. The monitor masks provide a means of disallowing all monitor-event program interruptions or allowing

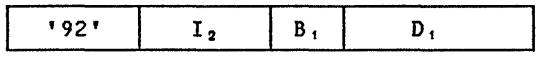
LzLabs GmbH. Ex. 1016-198

monitor-event program interruptions for all or selected classes.

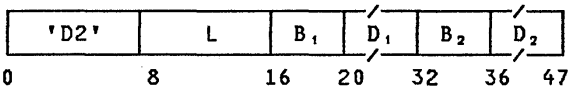
3. The monitor code provides a means of associating descriptive information, in addition to the class number, with each MONITOR CALL. Without the use of a base register, up to 4,096 distinct monitor codes can be associated with a monitoring interruption. With the base register designated by a nonzero 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.

MOVE

MVI $D_1(B_1),I_2$ [SI]



MVC $D_1(L,B_1),D_2(B_2)$ [SS]



The second operand is placed in 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 MOVE are given in Appendix A.

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 LONG

MVCL R_1,R_2 [RR]



The second operand is placed in the first-operand location, provided overlapping of operand 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.

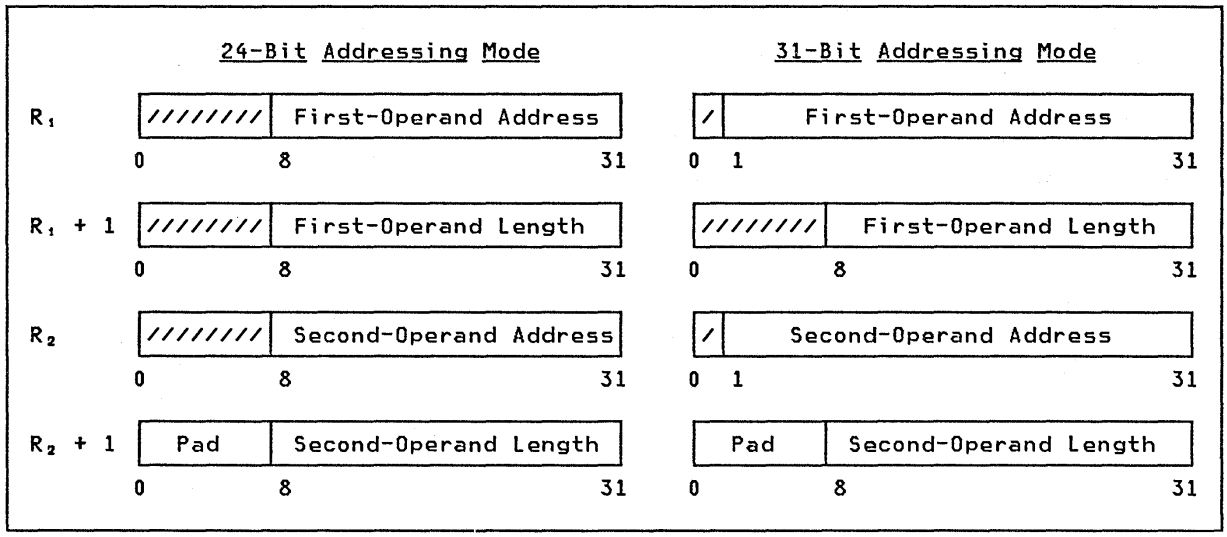
The R_1 and R_2 fields each specify an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of the general registers specified by the R_1 and R_2 fields, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 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 the general registers specified by the R_1 and R_2 fields 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 the figure "Register Contents for MOVE LONG."

LzLabs GmbH. Ex. 1016-199



Register Contents for MOVE LONG

The movement starts at the left end of both fields and proceeds to the right. The operation is ended when the number of bytes specified by bit positions 8-31 of 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.

When the length specified by bit positions 8-31 of 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 causes termination, the contents of registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes moved, and the contents of register 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 registers R_1 and R_2 are set to zeros; the contents of bit positions 0-7 of 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 register $R_2 + 1$ is 0, the address in register R_2 is incremented by the original contents of register $R_2 + 1$, and 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.

As viewed by other CPU and 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.

LzLabs GmbH. Ex. 1016-200

At the completion of the operation, the length in register $R_1 + 1$ is decremented by the number of bytes stored at the first-operand location, and the address in register R_1 is incremented by the same amount. The length in register $R_2 + 1$ is decremented by the number of bytes moved out of the second-operand location, and the address in register R_2 is incremented by the same amount. The leftmost bits which are not part of the address in registers R_1 and R_2 are set to zeros, including the case 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 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 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 is not recognized, and no change bits are set.

Resulting Condition Code:

0 First-operand and second-operand lengths are equal
1 First-operand length is low
2 First-operand length is high
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 MOVE LONG is given in Appendix A.

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 channel program will attempt to access and use the area as soon as it appears to be zero.

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 execution. Thus, if, after a move into several blocks of the first operand, an addressing or protection exception occurs, the registers remain unchanged.

4. When the first-operand length is zero, the operation consists in setting the condition code and setting the high-order bytes of registers R_1 and R_2 to zero.

5. When the contents of the R_1 and R_2 fields are the same, the operation proceeds the same way as when two distinct pairs of registers having the same contents are specified. 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. 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,

LzLabs GmbH. Ex. 1016-201

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.

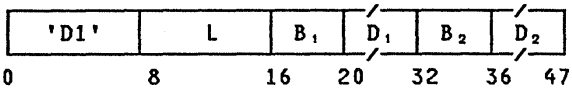
7. Special precautions must be taken if MOVE LONG is made the target of an EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.

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 since 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 the section "Interruptible Instructions" in Chapter 5, "Program Execution."

MOVE NUMERICS

MVN $D_1(L,B_1),D_2(B_2)$ [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 MOVE NUMERICS is given in Appendix A.

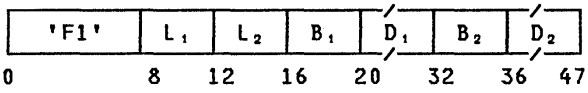
2. MOVE NUMERICS moves the numeric portion of a decimal-data field that is in the zoned format. The zoned-decimal format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.

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 channel program may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

LzLabs GmbH. Ex. 1016-202

MOVE WITH OFFSET

MVO $D_1(L_1,B_1),D_2(L_2,B_2)$ [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 in 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.

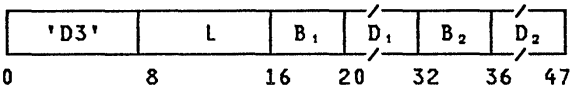
2. Access to the rightmost byte of the first operand of MOVE WITH OFFSET consists in fetching the rightmost four bits and subsequently storing the updated value of this byte. These fetch and store accesses to the rightmost byte of the first operand do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or channel program may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

3. 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.

4. The storage-operand references for MOVE WITH OFFSET may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

MOVE ZONES

MVZ $D_1(L,B_1),D_2(B_2)$ [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)

Programming Notes

1. An example of the use of MOVE ZONES is given in Appendix A.

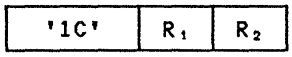
2. MOVE ZONES moves the zoned portion of a decimal field in the zoned format. The zoned format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.

LzLabs GmbH. Ex. 1016-203

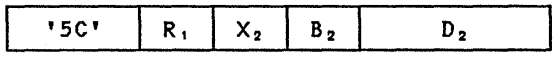
3. Accesses to the first operand of MOVE ZONES consist in fetching the leftmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or channel program may also be updating the location. An example of this effect is shown for the OR (OI) instruction in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

MULTIPLY

MR R_1,R_2 [RR]



M $R_1,D_2(X_2,B_2)$ [RX]



The second word of the first operand (multiplicand) is multiplied by the second operand (multiplier), and the doubleword product is placed at the first-operand location.

The R_1 field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R_1 is odd, a specification exception is recognized.

Both the multiplicand and multiplier are treated as 32-bit signed binary integers. The multiplicand is taken from the odd-numbered register of the pair specified by the R_1 field. The contents of the even-numbered register are ignored. The product is a 64-bit signed binary integer, which replaces the contents of the even-odd pair of general registers specified by the R_1 field. 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:

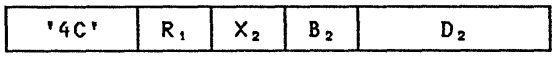
Access (fetch, operand 2 of M only) Specification

Programming Notes

1. An example of the use of MULTIPLY is given in Appendix A.
2. The significant part of the product usually occupies 62 bits or fewer. Only when two maximum negative numbers are multiplied are 63 significant product bits formed.

MULTIPLY HALFWORD

MH $R_1,D_2(X_2,B_2)$ [RX]



The first operand (multiplicand) is multiplied by the second operand (multiplier), and the product is placed at the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed binary integer.

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)

Programming Notes

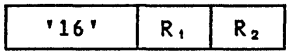
1. An example of the use of MULTIPLY HALFWORD is given in Appendix A.
2. The significant part of the product usually occupies 46 bits or fewer. Only when two maximum negative numbers are multiplied are 47 significant product bits formed. Since the rightmost 32 bits of the product are stored unchanged,

LzLabs GmbH. Ex. 1016-204

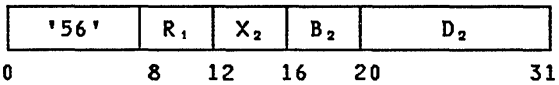
ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow. For a negative product, the 32 bits placed in register R_1 are the rightmost part of the product in two's-complement notation.

OR

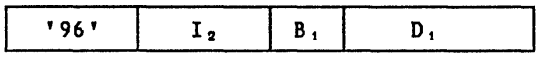
OR R_1,R_2 [RR]



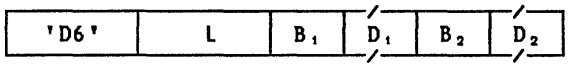
O $R_1,D_2(X_2,B_2)$ [RX]



OI $D_1(B_1),I_2$ [SI]



OC $D_1(L,B_1),D_2(B_2)$ [SS]



The OR of the first and second operands is placed in the first-operand location.

The connective OR is applied to the operands bit by bit. A bit position in the result is 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 only one byte in length, and only one byte is stored.

Resulting Condition Code:

0 Result is zero
1 Result is not zero
2 --
3 --

Program Exceptions:

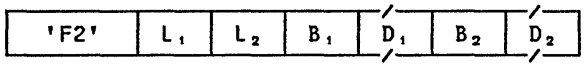
Access (fetch, operand 2, O and OC; fetch and store, operand 1, OI and OC)

Programming Notes

1. Examples of the use of OR are given in Appendix A.
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 channel program may also be updating the location. An example of this effect is shown in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

PACK

PACK $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The format of the second operand is changed from zoned to packed, and the result is placed in the first-operand location. The zoned and packed formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as though it had the zoned format. The numeric bits of each byte are treated as a digit. The zone bits are ignored, 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

LzLabs GmbH. Ex. 1016-205

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:

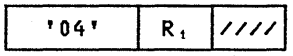
Access (fetch, operand 2; store, operand 1)

Programming Notes

1. An example of the use of PACK is given in Appendix A.
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 must be extended on the right with a dummy byte, which subsequently is ignored in the result field.
4. The storage-operand references for PACK may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

SET PROGRAM MASK

SPM R_1 [RR]



The contents of the general register specified by the R_1 field are used to set the condition code and the program mask of the current PSW.

Bits 12-15 of the instruction are ignored.

Bits 2 and 3 of the register specified by the R_1 field replace the condition code, and bits 4-7 replace the program mask. Bits 0, 1, and 8-31 of the register specified by the R_1 field are ignored.

Resulting Condition Code:

0 Bit 2 is zero, and bit 3 is zero
1 Bit 2 is zero, and bit 3 is one
2 Bit 2 is one, and bit 3 is zero
3 Bit 2 is one, and bit 3 is one

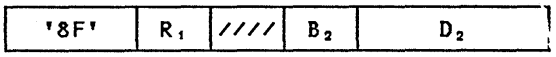
Program Exceptions: None.

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 or 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 $R_1,D_2(B_2)$ [RS]



The 63-bit numeric part of the signed first operand is shifted left the number of bits specified by the second-operand address.

Bits 12-15 of the instruction are ignored.

The R_1 field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R_1 is odd, 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.

LzLabs GmbH. Ex. 1016-206

The first operand is treated as a 64-bit signed binary integer. The sign position of the even register remains unchanged. The leftmost bit position of the odd register contains a numeric bit, which participates in the 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, 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 is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:

Fixed-point overflow
Specification

Programming Notes

1. An example of the use of SHIFT LEFT DOUBLE is given in Appendix A.

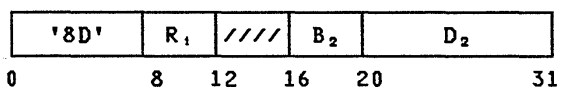
2. The eight shift instructions provide the following three pairs of alternatives: left or right, single or double, and signed or logical. The signed shifts differ from the logical shifts in that, in the signed shifts, overflow is recognized, the condition code is set, and the leftmost bit participates 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. 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.

Bits 12-15 of the instruction are ignored.

The R_1 field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R_1 is odd, 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.

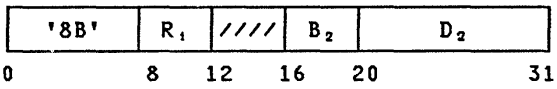
Condition Code: The code remains unchanged.

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.

Bits 12-15 of the instruction are ignored.

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 mask bit is one, a program interruption for fixed-point overflow occurs.

LzLabs GmbH. Ex. 1016-207

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:

Fixed-point overflow

Programming Notes

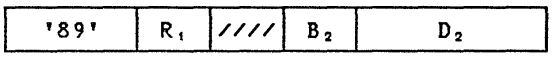
1. An example of the use of SHIFT LEFT SINGLE is given in Appendix A.

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 $R_1,D_2(B_2)$ [RS]



The 32-bit first operand is shifted left the number of bits specified by the second-operand address.

Bits 12-15 of the instruction are ignored.

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.

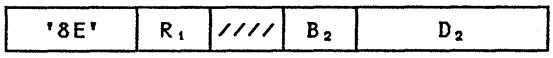
All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 0 are not inspected and are lost. Zeros are supplied to the vacated bit positions on the right.

Condition Code: The code remains unchanged.

Program Exceptions: None.

SHIFT RIGHT DOUBLE

SRDA $R_1,D_2(B_2)$ [RS]



The 63-bit numeric part of the signed first operand is shifted right the number of bits specified by the second-operand address.

Bits 12-15 of the instruction are ignored.

The R_1 field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R_1 is odd, 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 register remains unchanged. The leftmost bit position of the odd 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:

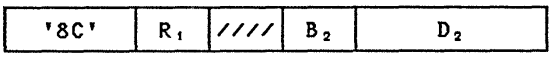
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Exceptions:

Specification

SHIFT RIGHT DOUBLE LOGICAL

SRDL $R_1,D_2(B_2)$ [RS]



The 64-bit first operand is shifted right the number of bits specified by the second-operand address.

Bits 12-15 of the instruction are ignored.

LzLabs GmbH. Ex. 1016-208

The R_1 field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R_1 is odd, 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.

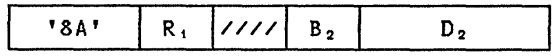
Condition Code: The code remains unchanged.

Program Exceptions:

Specification

SHIFT RIGHT SINGLE

SRA $R_1,D_2(B_2)$ [RS]



The 31-bit numeric part of the signed first operand is shifted right the number of bits specified by the second-operand address.

Bits 12-15 of the instruction are ignored.

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 is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Exceptions: None.

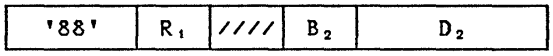
Programming Notes

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 $R_1,D_2(B_2)$ [RS]



The 32-bit first operand is shifted right the number of bits specified by the second-operand address.

Bits 12-15 of the instruction are ignored.

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.

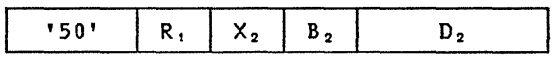
All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 31 are not inspected and are lost. Zeros are supplied to the vacated bit positions on the left.

Condition Code: The code remains unchanged.

Program Exceptions: None.

STORE

ST $R_1,D_2(X_2,B_2)$ [RX]



The first operand is stored at the second-operand location.

LzLabs GmbH. Ex. 1016-209

The 32 bits in the general register are placed unchanged at the second-operand location.

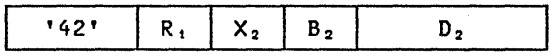
Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

STORE CHARACTER

STC $R_1,D_2(X_2,B_2)$ [RX]



Bits 24-31 of the general register designated by the R_1 field are placed unchanged at the second-operand location. The second operand is one byte in length.

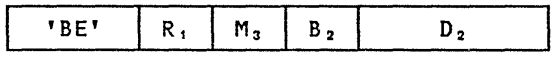
Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

STORE CHARACTERS UNDER MASK

STCM $R_1,M_3,D_2(B_2)$ [RS]



Bytes selected from the first operand under control of a mask are placed in 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 the general register designated by the R_1 field. The bytes corresponding to ones in the mask are placed in the same order in 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.

When the mask is zero, the single byte designated by the second-operand address remains unchanged; however, on some models, the value may be fetched and subsequently stored back at the same storage location. No access by another CPU is permitted to the location designated by the second-operand address between the moment that the value is fetched and the value is stored.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

Programming Notes

1. An example of the use of STORE CHARACTERS UNDER MASK is given in Appendix A.

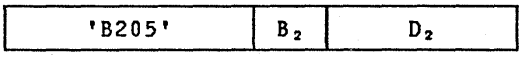
2. STORE CHARACTERS UNDER MASK with a mask of 0111 may be used to store a three-byte address, for example, in modifying the address in a CCW.

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 will be 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 specified by the second-operand address. This access is not interlocked against accesses by channel programs. In addition, it may cause any of the following to occur for the byte specified 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 turned on.

STORE CLOCK

STCK $D_2(B_2)$ [S]



The current value of the TOD clock is stored at the eight-byte field designated by the second-operand address,

LzLabs GmbH. Ex. 1016-210

provided the clock is in the set, stopped, or not-set state.

Zeros are stored for the rightmost bit positions that are not provided by the clock.

Zeros are stored at the operand location when the clock is in the error state or in 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. CPU operation is delayed until all previous accesses by this CPU to storage have been completed, as observed by other CPU and channel programs, and then the value of the clock is fetched. No subsequent instructions or their operands are fetched by this CPU until the clock value has been placed in storage, as observed by other CPU and channel programs.

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.

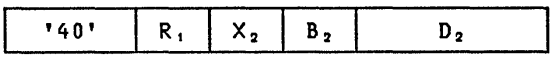
2. Condition code 0 normally indicates that the clock has been set by the control program. Accordingly, the value may be used in elapsed-time measurements and as a valid time-of-day and calendar indication. Condition code 1 indicates that the clock value is the elapsed time since the power for the clock 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. Condition code 3 indicates that the clock is either in the stopped state or not-operational state. These two states can normally be distinguished since an all-zero value is stored when in the not-operational state.

4. If a problem program written for the 370-XA mode is to be run 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 HALFWORD

STH $R_1,D_2(X_2,B_2)$ [RX]



Bits 16-31 of the general register designated by the R_1 field are placed unchanged at the second-operand location. The second operand is two bytes in length.

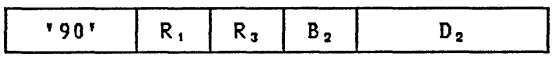
Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

STORE MULTIPLE

STM $R_1,R_3,D_2(B_2)$ [RS]



The contents of the set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 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 register numbers, starting with the register specified by R_1 and continuing up to and including the register specified by R_3, with register 0 following register 15.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

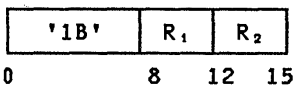
LzLabs GmbH. Ex. 1016-211

Programming Note

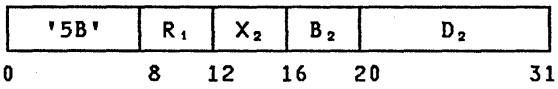
An example of the use of STORE MULTIPLE is given in Appendix A.

SUBTRACT

SR R_1,R_2 [RR]



S $R_1,D_2(X_2,B_2)$ [RX]



The second operand is subtracted from the first operand, and the difference is placed in 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 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Exceptions:

Access (fetch, operand 2 of S only)
Fixed-point overflow

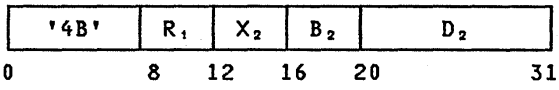
Programming Notes

1. When, in the RR format, the R_1 and R_2 fields 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 $R_1,D_2(X_2,B_2)$ [RX]



The second operand is subtracted from the first operand, and the difference is placed in 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 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Exceptions:

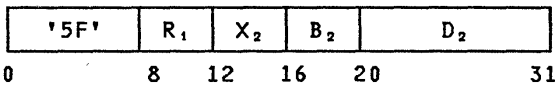
Access (fetch, operand 2)
Fixed-point overflow

SUBTRACT LOGICAL

SLR R_1,R_2 [RR]



SL $R_1,D_2(X_2,B_2)$ [RX]



The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The operands and the difference are treated as 32-bit unsigned binary integers.

Resulting Condition Code:

0 --
1 Difference is not zero, with no carry
2 Difference is zero, with carry

LzLabs GmbH. Ex. 1016-212

3 Difference is not zero, with carry

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.

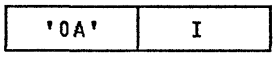
3. A zero difference is always accompanied by a carry out of bit position 0.

4. The condition-code setting for SUBTRACT LOGICAL can also be interpreted as indicating the presence and absence of a borrow, as follows:

   1 Difference is not zero, with borrow
   2 Difference is zero, with no borrow
   3 Difference is not zero, with no borrow

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" in Chapter 6, "Interruptions."

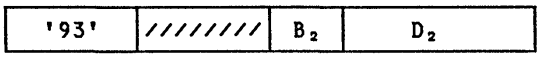
A serialization and a checkpoint-synchronization function is performed. CPU operation is delayed until all previous storage accesses by this CPU to storage have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed.

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 AND SET

TS $D_2(B_2)$ [S]



The leftmost bit (bit position 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.

Bits 8-15 of the instruction are ignored.

The byte in storage is set to all ones as it is fetched for the testing of bit position 0. No access by another CPU to this location is permitted between the moment of fetching and the moment of storing all ones.

A serialization function is performed before the byte is fetched and again after the storing of all ones. CPU operation is delayed until all previous accesses by this CPU to storage have been completed, as observed by other CPU and channel programs, and then the byte is fetched. No subsequent instructions or their operands are accessed by this CPU until the all-ones value has been placed in storage, as observed by other CPU and channel programs.

Resulting Condition Code:

0 Leftmost bit of byte specified was zero
1 Leftmost bit of byte specified was one
2 --
3 --

Program Exceptions:

Access (fetch and store, operand 2)

LzLabs GmbH. Ex. 1016-213

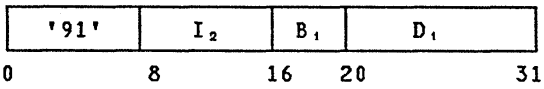
Programming Notes

1. TEST AND SET may be used for controlled sharing of a common storage area by programs operating on different CPUs. This instruction, which is provided primarily for compatibility with programs written for System/360, does not provide suitable functions for sharing between programs on a single CPU or for programs that may be interrupted. COMPARE AND SWAP and COMPARE DOUBLE AND SWAP provide these functions. 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.

TEST UNDER MASK

TM $D_1(B_1),I_2$ [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 the mask is all zeros
1 Selected bits mixed zeros and ones
2 --
3 Selected bits all ones

Program Exceptions:

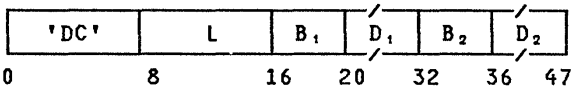
Access (fetch, operand 1)

Programming Note

An example of the use of TEST UNDER MASK is given in Appendix A.

TRANSLATE

TR $D_1(L,B_1),D_2(B_2)$ [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 designates the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding 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)

LzLabs GmbH. Ex. 1016-214

Programming Notes

1. An example of the use of TRANSLATE is given in Appendix A.

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. 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 channel program may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

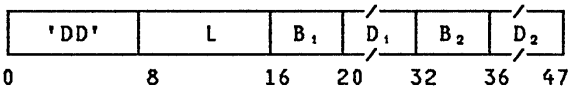
5. Because each eight-bit argument byte is added to the initial second-operand address to obtain the address of a function byte, the list may contain 256 bytes. In cases where it is known that not all eight-bit argument values will occur, it is possible to reduce the size of the list.

6. 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.

7. The storage-operand references of TRANSLATE may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

TRANSLATE AND TEST

TRT $D_1(L,B_1),D_2(B_2)$ [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 designates 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. In the 24-bit addressing mode, the address replaces bits 8-31, and bits 0-7 of general register 1 remain unchanged. In the 31-bit addressing mode, the address replaces bits 1-31, and bit 0 of general register 1 is set to zero. In both modes, bits 0-23 of general register 2 remain unchanged.

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.

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.

LzLabs GmbH. Ex. 1016-215

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)

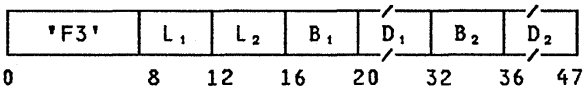
Programming Notes

1. An example of the use of TRANSLATE AND TEST is given in Appendix A.

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.

UNPACK

UNPK $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The format of the second operand is changed from packed to zoned, and the result is placed in the first-operand location. The packed and zoned formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as though it had 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 UNPACK is given in Appendix A.

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 the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

LzLabs GmbH. Ex. 1016-216

CHAPTER 8. DECIMAL INSTRUCTIONS

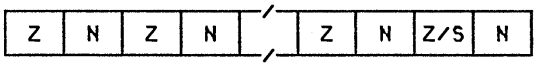
Decimal-Number Formats 8-1
  Zoned Format 8-1
  Packed Format 8-1
  Decimal Codes 8-2
Decimal Operations 8-2
  Decimal-Arithmetic Instructions 8-2
  Editing Instructions 8-3
  Execution of Decimal Instructions 8-3
  Other Instructions for Decimal Operands 8-3
Instructions 8-3
  ADD DECIMAL 8-5
  COMPARE DECIMAL 8-5
  DIVIDE DECIMAL 8-5
  EDIT 8-6
  EDIT AND MARK 8-9
  MULTIPLY DECIMAL 8-10
  SHIFT AND ROUND DECIMAL 8-10
  SUBTRACT DECIMAL 8-11
  ZERO AND ADD 8-11

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 all 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. Decimal operands occupy storage fields that can start on any byte boundary.

ZONED FORMAT

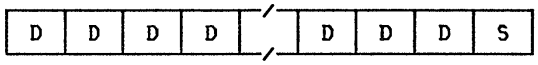


In the zoned format, the rightmost four bits of a byte are called the numeric bits (N) and normally consist of a code representing a decimal digit. The leftmost four bits of a byte are called the zone bits (Z), except for the rightmost byte of a decimal operand, where these bits may be treated either 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

LzLabs GmbH. Ex. 1016-217

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.

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 when appearing 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 operand, but never appear in the completed result of a decimal-arithmetic instruction or CONVERT TO DECIMAL, 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 invalid digit codes are detected but no sign code is invalid, the operation is terminated.

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 appears in the left four bit positions of each byte representing a decimal digit in zoned-format results. Zoned-format results are produced by EDIT, EDIT AND MARK, and UNPACK, except that the left four bit positions of the rightmost byte produced by UNPACK contain whatever code exists in the sign position of the packed operand. The right four bit positions of each result byte in the zoned format for EDIT and EDIT AND MARK contain a decimal-digit code.

The meaning of the decimal codes is summarized in the figure "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.

	Recognized As	
Code	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)

Summary of Digit and Sign Codes

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

LzLabs GmbH. Ex. 1016-218

contains an odd 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.

Programming Note

The same decimal field in storage may be specified for both operands of ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, and SUBTRACT DECIMAL. Thus, a decimal number may be added to itself, compared to itself, etc. SUBTRACT DECIMAL may be used to set a decimal field in storage to zero.

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 the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.") 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 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.

INSTRUCTIONS

The decimal instructions and their mnemonics, formats, and operation codes

LzLabs GmbH. Ex. 1016-219

are listed in the figure "Summary of Decimal Instructions." The figure also indicates when the condition code is set 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.

Programming Note

The decimal instructions for the 370-XA mode and the System/370 mode are identical, with the exception that, in the 31-bit addressing mode, EDIT AND MARK places a 31-bit address in general register 1. In the 24-bit addressing mode, EDIT AND MARK operates as in the System/370 mode; that is, a 24-bit address is placed in general register 1, and the leftmost byte of general register 1 is unchanged.

Name	Mnemonic	Characteristics				Op Code
ADD DECIMAL	AP	SS C	A	D DF	ST	FA
COMPARE DECIMAL	CP	SS C	A	D		F9
DIVIDE DECIMAL	DP	SS	A SP	D DK	ST	FD
EDIT	ED	SS C	A	D	ST	DE
EDIT AND MARK	EDMK	SS C	A	D G1	R ST	DF
MULTIPLY DECIMAL	MP	SS	A SP	D	ST	FC
SHIFT AND ROUND DECIMAL	SRP	SS C	A	D DF	ST	F0
SUBTRACT DECIMAL	SP	SS C	A	D DF	ST	FB
ZERO AND ADD	ZAP	SS C	A	D DF	ST	F8

Explanation:

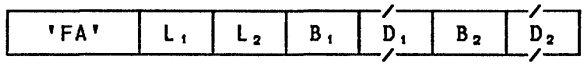
A	Access exceptions for logical addresses
C	Condition code is set
D	Data exception
DF	Decimal-overflow exception
DK	Decimal-divide exception
G1	Instruction execution includes the implied use of general register 1
R	PER general-register-alteration event
SP	Specification exception
SS	SS instruction format
ST	PER storage-alteration event

Summary of Decimal Instructions

LzLabs GmbH. Ex. 1016-220

ADD DECIMAL

AP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The second operand is added to the first operand, and the resulting sum is placed in 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 leftmost 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 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Exceptions:

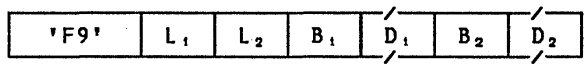
Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal overflow

Programming Note

An example of the use of ADD DECIMAL is given in Appendix A.

COMPARE DECIMAL

CP $D_1(L_1,B_1),D_2(L_2,B_2)$ [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 are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions:

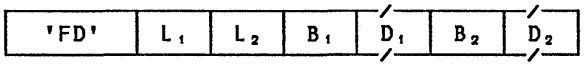
Access (fetch, operands 1 and 2)
Data

Programming Notes

1. An example of the use of COMPARE DECIMAL is given in Appendix A.
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.

DIVIDE DECIMAL

DP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The first operand (the dividend) is divided by the second operand (the divisor). The resulting quotient and remainder are placed in 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

LzLabs GmbH. Ex. 1016-221

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 DIVIDE DECIMAL is given in Appendix A.

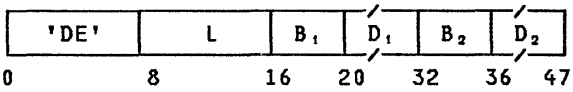
2. The dividend cannot exceed 31 digits and sign. Since the remainder cannot be shorter than one digit and sign, the quotient cannot exceed 29 digits and sign.

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. 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 $D_1(L,B_1),D_2(B_2)$ [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 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 or a decimal digit. 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:

LzLabs GmbH. Ex. 1016-222

Name	Code
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 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

LzLabs GmbH. Ex. 1016-223

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 bytes in the second operand selected by digit selectors or significance starters after the last field separator. When the pattern contains no field separator, there is only one field, which is considered to be the last field. The last field is considered to be of zero length if no digit selectors or significance starters appear in the pattern, if none appear after the last field separator, or if the last byte in the pattern is a field separator.

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.)

The figure "Summary of Editing Functions" 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 is zero or of zero length
1 Last field is less than zero
2 Last field is greater than zero
3 --

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Notes

1. Examples of the use of EDIT are given in Appendix A.

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 appears 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 EDIT designates a location that is less than the length of the first operand 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. 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. 1016-224

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

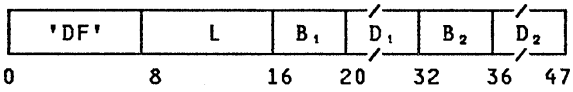
Explanation:

* 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.

Summary of Editing Functions

EDIT AND MARK

EDMK $D_1(L,B_1),D_2(B_2)$ [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 each 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 whenever 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.

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.

See the figure "Summary of Editing Functions" under EDIT for a summary of the EDIT and EDIT AND MARK operations.

Resulting Condition Code:

0 Last field is zero or of zero length
1 Last field is less than zero
2 Last field is greater than zero
3 --

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Notes

1. Examples of the use of EDIT AND MARK are given in Appendix A.

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

LzLabs GmbH. Ex. 1016-225

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.

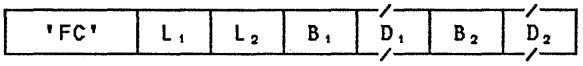
3. No address is inserted in general register 1 when the significance indicator is turned on as a result of encountering a significance starter with the corresponding source digit zero. To ensure that general register 1 contains a proper address when this occurs, the address of the pattern byte that immediately follows the appropriate significance starter could be placed in the register beforehand.

4. When multiple fields are edited with one EDIT AND MARK, the address inserted in general register 1 applies only to the last field edited.

5. See also the programming note under EDIT regarding performance degradation due to a possible trial execution.

MULTIPLY DECIMAL

MP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The product of the first operand (the multiplicand) and the second operand (the multiplier) is placed in 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.

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

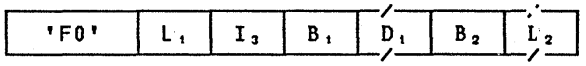
Programming Notes

1. An example of the use of MULTIPLY DECIMAL is given in Appendix A.

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.

LzLabs GmbH. Ex. 1016-226

Shift Value	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, are 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 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 is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:

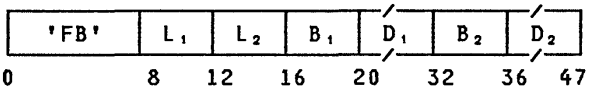
Access (fetch and store, operand 1)
Data
Decimal overflow

Programming Notes

1. Examples of the use of SHIFT AND ROUND are given in Appendix A.
2. SHIFT AND ROUND 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 $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The second operand is subtracted from the first operand, and the resulting difference is placed in 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:

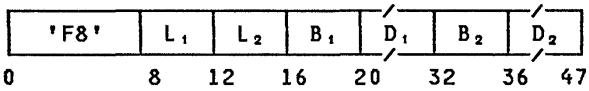
0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal overflow

ZERO AND ADD

ZAP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The second operand is placed in 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. 1016-227

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.

Resulting Condition Code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:

Access (fetch, operand 2; store, operand 1)
Data
Decimal overflow

Programming Note

An example of the use of ZERO AND ADD is given in Appendix A.

LzLabs GmbH. Ex. 1016-228

CHAPTER 9. FLOATING-POINT INSTRUCTIONS

Floating-Point Number Representation 9-1
Normalization 9-2
Floating-Point-Data Format 9-2
Instructions 9-4
  ADD NORMALIZED 9-6
  ADD UNNORMALIZED 9-7
  COMPARE 9-8
  DIVIDE 9-8
  HALVE 9-10
  LOAD 9-10
  LOAD AND TEST 9-11
  LOAD COMPLEMENT 9-11
  LOAD NEGATIVE 9-11
  LOAD POSITIVE 9-12
  LOAD ROUNDED 9-12
  MULTIPLY 9-13
  STORE 9-14
  SUBTRACT NORMALIZED 9-14
  SUBTRACT UNNORMALIZED 9-15

Floating-point instructions are used to perform calculations on operands with a wide range of magnitude and to yield results scaled to preserve precision.

The floating-point instructions provide for loading, rounding, adding, subtracting, comparing, multiplying, dividing, and storing, as well as controlling the sign of short, long, and extended operands. Short operands generally permit faster processing and require less storage than long or extended operands. On the other hand, long and extended operands permit greater precision in computation. Four floating-point registers are provided. Instructions may perform either register-to-register or storage-and-register operations.

Most of the instructions generate normalized results, which preserve the highest precision in the operation. For addition and subtraction, instructions are also provided that generate unnormalized results. Either normalized or unnormalized numbers may be used as operands for any floating-point operation.

FLOATING-POINT NUMBER REPRESENTATION

A floating-point number consists of a signed 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 value of a floating-point number is the product of its fraction and the number 16 raised to the power of the exponent which is represented by its characteristic.

The fraction of a floating-point 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. The fraction is represented by its absolute value and a separate sign bit. The entire number is positive or negative, depending on whether the sign bit of the fraction is zero or one, respectively.

When a floating-point operation would cause the result exponent to exceed 63, the characteristic wraps around from 127 to 0, and an 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 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 a floating-point number with a zero characteristic, zero fraction, and plus sign. A true zero may arise as the normal result of an arithmetic operation because of the particular magnitude of the operands. The result is forced to be a true zero when:

LzLabs GmbH. Ex. 1016-229

1. An exponent underflow occurs and the exponent-underflow mask bit in the PSW is zero,

2. The result fraction of an addition or subtraction operation is zero and the significance mask bit in the PSW is zero, or

3. The operand of the HALVE instruction, one or both operands of the MULTIPLY instruction, or the dividend in the DIVIDE instruction has a zero fraction.

When a program interruption for 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 program interruption for significance occurs, the fraction remains zero, the sign is positive, and the characteristic remains correct.

The sign of a sum, difference, product, or quotient with a zero fraction is positive. The sign of a zero fraction resulting from other operations is established from the operand sign, the same as for nonzero fractions.

NORMALIZATION

A quantity can be represented with the greatest precision by a floating-point number of a given fraction length when that number is normalized. A normalized floating-point 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; its characteristic either remains unchanged, or it 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, and HALVE 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 multiplication and division 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 placed in 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.

Programming Note

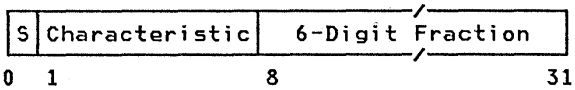
Up to three leftmost bits of the fraction of a normalized number may be zeros, since the nonzero test applies to the entire leftmost hexadecimal digit.

FLOATING-POINT-DATA FORMAT

Floating-point 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.

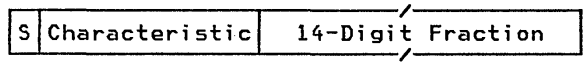
The floating-point registers contain 64 bits each and are numbered 0, 2, 4, and 6. A short or long floating-point number requires a single floating-point register. An extended floating-point number requires a pair of these registers: either registers 0 and 2 or registers 4 and 6; the two register pairs are designated as 0 or 4, respectively. When the R_1 or R_2 field of a floating-point instruction designates any register number other than 0, 2, 4, or 6 for the short or long format, or any register number other than 0 or 4 for the extended format, a program interruption for specification exception occurs.

Short Floating-Point Number



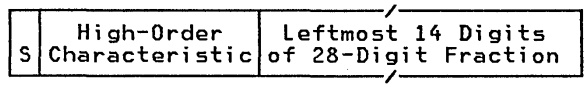
LzLabs GmbH. Ex. 1016-230

Long Floating-Point Number

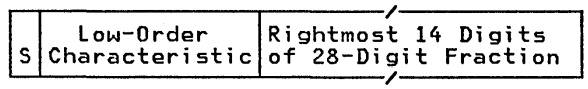


Extended Floating-Point Number

High-Order Part



Low-Order Part



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.

A short floating-point number occupies only the leftmost 32 bit positions of a floating-point register. The rightmost 32 bit positions of the register are ignored when used as an operand in the short format and remain unchanged when a short result is placed in the register.

An extended floating-point number has a 28-digit fraction and consists of two long floating-point numbers which are called the high-order and low-order parts. The high-order part may be any long floating-point 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 floating-point 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 in the extended format is 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. 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 floating-point number depends on the format.

In the short format:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \leq M \leq (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \leq M \leq (1 - 16^{-28}) \times 16^{63}$$

In all formats, approximately:

$$5.4 \times 10^{-79} \leq M \leq 7.2 \times 10^{75}$$

Although the final result of a floating-point 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 floating-point operations is available for both short and long operands. These instructions generate a result that has the same format as the operands, except that for the MULTIPLY instruction, a long product is produced from a short multiplier and multiplicand. Extended floating-point instructions are provided only for normalized addition, subtraction, multiplication, and division. Two additional multiplication instructions generate an extended product from a long multiplier and multiplicand. The rounding instructions provide for rounding from extended to long format and from long to short format.

Programming Notes

1. A long floating-point number can be converted to the extended format by appending any long floating-point number having a zero fraction, including a true zero. Conversion from the extended to the long format can be accomplished by truncation or by means of the LOAD ROUNDED instruction.

LzLabs GmbH. Ex. 1016-231

2. In the absence of an exponent overflow or exponent underflow, the long floating-point 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 floating-point number, does not express the correct characteristic value.

3. 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 floating-point number, depending on whether the 15th digit of the normalized 28-digit fraction is nonzero or zero.

INSTRUCTIONS

The floating-point instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of Floating-Point Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

Mnemonics for the floating-point instructions have an R as the last letter when the instruction is in the RR format. For instructions where all operands are the same length, certain letters are used to represent operand-format length and normalization, as follows:

E	short normalized
U	short unnormalized
D	long normalized
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 LOAD (short), for example, LER is the mnemonic and R_1,R_2 the operand designation.

Programming Note

The only difference between the floating-point instructions for the 370-XA mode and the System/370 mode is that, in the 370-XA mode, DIVIDE with extended operands (DXR) has been added, and the floating-point instructions, including the extended-precision instructions, are part of the standard instruction set.

LzLabs GmbH. Ex. 1016-232

Name	Mnemonic	Characteristics							Op Code
ADD NORMALIZED (extended)	AXR	RR	C		SP	EU	E0	LS		36
ADD NORMALIZED (long)	ADR	RR	C		SP	EU	E0	LS		2A
ADD NORMALIZED (long)	AD	RX	C	A	SP	EU	E0	LS		6A
ADD NORMALIZED (short)	AER	RR	C		SP	EU	E0	LS		3A
ADD NORMALIZED (short)	AE	RX	C	A	SP	EU	E0	LS		7A
ADD UNNORMALIZED (long)	AWR	RR	C		SP		E0	LS		2E
ADD UNNORMALIZED (long)	AW	RX	C	A	SP		E0	LS		6E
ADD UNNORMALIZED (short)	AUR	RR	C		SP		E0	LS		3E
ADD UNNORMALIZED (short)	AU	RX	C	A	SP		E0	LS		7E
COMPARE (long)	CDR	RR	C		SP					29
COMPARE (long)	CD	RX	C	A	SP					69
COMPARE (short)	CER	RR	C		SP					39
COMPARE (short)	CE	RX	C	A	SP					79
DIVIDE (extended)	DXR	RRE			SP	EU	E0	FK		B22D
DIVIDE (long)	DDR	RR			SP	EU	E0	FK		2D
DIVIDE (long)	DD	RX		A	SP	EU	E0	FK		6D
DIVIDE (short)	DER	RR			SP	EU	E0	FK		3D
DIVIDE (short)	DE	RX		A	SP	EU	E0	FK		7D
HALVE (long)	HDR	RR			SP	EU				24
HALVE (short)	HER	RR			SP	EU				34
LOAD (long)	LDR	RR			SP					28
LOAD (long)	LD	RX		A	SP					68
LOAD (short)	LER	RR			SP					38
LOAD (short)	LE	RX		A	SP					78
LOAD AND TEST (long)	LTDR	RR	C		SP					22
LOAD AND TEST (short)	LTER	RR	C		SP					32
LOAD COMPLEMENT (long)	LCDR	RR	C		SP					23
LOAD COMPLEMENT (short)	LCER	RR	C		SP					33
LOAD NEGATIVE (long)	LNDR	RR	C		SP					21
LOAD NEGATIVE (short)	LNER	RR	C		SP					31
LOAD POSITIVE (long)	LPDR	RR	C		SP					20
LOAD POSITIVE (short)	LPER	RR	C		SP					30
LOAD ROUNDED (extended to long)	LRDR	RR			SP		E0			25
LOAD ROUNDED (long to short)	LRER	RR			SP		E0			35
MULTIPLY (extended)	MXR	RR			SP	EU	E0			26
MULTIPLY (long)	MDR	RR			SP	EU	E0			2C
MULTIPLY (long)	MD	RX		A	SP	EU	E0			6C
MULTIPLY (long to extended)	MXDR	RR			SP	EU	E0			27
MULTIPLY (long to extended)	MXD	RX		A	SP	EU	E0			67
MULTIPLY (short to long)	MER	RR			SP	EU	E0			3C
MULTIPLY (short to long)	ME	RX		A	SP	EU	E0			7C
STORE (long)	STD	RX		A	SP				ST	60
STORE (short)	STE	RX		A	SP				ST	70
SUBTRACT NORMALIZED (extended)	SXR	RR	C		SP	EU	E0	LS		37
SUBTRACT NORMALIZED (long)	SDR	RR	C		SP	EU	E0	LS		2B
SUBTRACT NORMALIZED (long)	SD	RX	C	A	SP	EU	E0	LS		6B
SUBTRACT NORMALIZED (short)	SER	RR	C		SP	EU	E0	LS		3B
SUBTRACT NORMALIZED (short)	SE	RX	C	A	SP	EU	E0	LS		7B
SUBTRACT UNNORMALIZED (long)	SWR	RR	C		SP		E0	LS		2F
SUBTRACT UNNORMALIZED (long)	SW	RX	C	A	SP		E0	LS		6F
SUBTRACT UNNORMALIZED (short)	SUR	RR	C		SP		E0	LS		3F
SUBTRACT UNNORMALIZED (short)	SU	RX	C	A	SP		E0	LS		7F

Summary of Floating-Point Instructions (Part 1 of 2)

LzLabs GmbH. Ex. 1016-233

Explanation:

A	Access exceptions for logical addresses
C	Condition code is set
EO	Exponent-overflow exception
EU	Exponent-underflow exception
FK	Floating-point-divide exception
LS	Significance exception
RR	RR instruction format
RRE	RRE instruction format
RX	RX instruction format
SP	Specification exception
ST	PER storage-alteration event

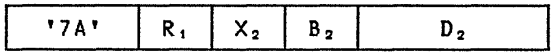
Summary of Floating-Point Instructions (Part 2 of 2)

ADD NORMALIZED

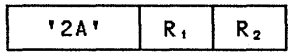
AER R_1,R_2 [RR, Short Operands]



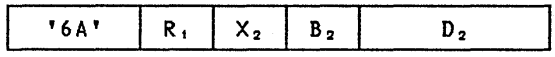
AE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



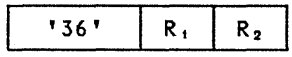
ADR R_1,R_2 [RR, Long Operands]



AD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



AXR R_1,R_2 [RR, Extended Operands]



The second operand is added to the first operand, and the normalized sum is placed in the first-operand location.

Addition of two floating-point 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 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. Vacated rightmost digit positions are filled with zeros, 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 sign is made plus.

An exponent-overflow exception is recognized 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

LzLabs GmbH. Ex. 1016-234

result characteristic 128 less than the correct value, and a program interruption for exponent overflow takes place. The result sign and fraction remain correct, and, for AXR, the characteristic of the low-order part remains correct.

An exponent-underflow exception is recognized when the characteristic of the normalized sum would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the result characteristic 128 greater than the correct value. The result sign and fraction remain correct, and a program interruption for exponent underflow takes place. When exponent underflow occurs and the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the result a true zero. For AXR, no exponent underflow is recognized when the characteristic of the low-order part would be less than zero but the characteristic of the high-order part is zero or greater.

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 significance mask bit. If the significance mask bit is one, no normalization occurs, the intermediate and final result characteristics are the same, and a program interruption for significance takes place. If the significance mask bit is zero, the program interruption does not occur; instead, the result is made a true zero.

The R_1 field for AER, AE, ADR, and AD, and the R_2 field for AER and ADR must designate register 0, 2, 4, or 6. The R_1 and R_2 fields for AXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Resulting Condition Code:

0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Exceptions:

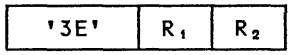
Access (fetch, operand 2 of AE and AD only)
Exponent overflow
Exponent underflow
Significance
Specification

Programming Notes

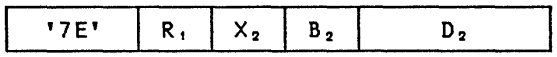
1. An example of the use of ADD NORMALIZED is given in Appendix A.
2. Interchanging the two operands in a floating-point addition does not affect the value of the sum.
3. The ADD NORMALIZED instruction normalizes the sum but not the operands. Thus, if one or both operands are unnormalized, precision may be lost during fraction alignment.

ADD UNNORMALIZED

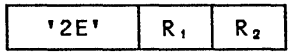
AUR R_1,R_2 [RR, Short Operands]



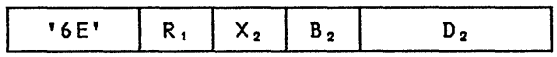
AU $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



AWR R_1,R_2 [RR, Long Operands]



AW $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



The second operand is added to the first operand, and the unnormalized sum is placed in 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. 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.

LzLabs GmbH. Ex. 1016-235

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Exceptions:

Access (fetch, operand 2 of AU and AW only)
Exponent overflow
Significance
Specification

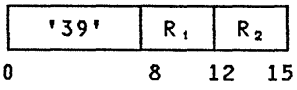
Programming Notes

1. An example of the use of ADD UNNORMALIZED is given in Appendix A.

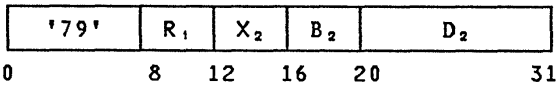
2. Except when the result is made a true zero, the characteristic of the result of ADD UNNORMALIZED is equal to the greater of the two operand characteristics, increased by one if the fraction addition produced a carry, or set to zero if exponent overflow occurred.

COMPARE

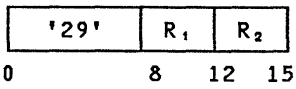
CER R_1,R_2 [RR, Short Operands]



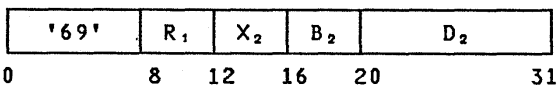
CE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



CDR R_1,R_2 [RR, Long Operands]



CD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



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 floating-point subtraction, except that the 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 exponent-overflow, exponent-underflow, or significance exception cannot occur.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions:

Access (fetch, operand 2 of CE and CD only)
Specification

Programming Notes

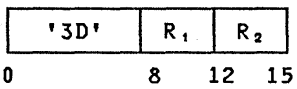
1. Examples of the use of COMPARE are given in Appendix A.

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.

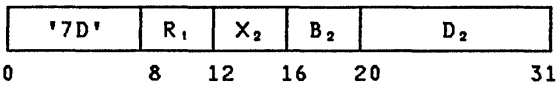
DIVIDE

DER R_1,R_2 [RR, Short Operands]

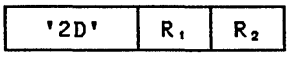


LzLabs GmbH. Ex. 1016-236

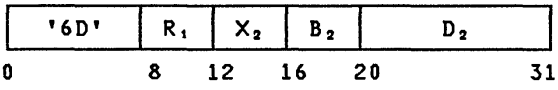
DE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



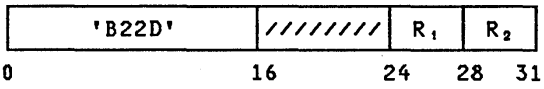
DDR R_1,R_2 [RR, Long Operands]



DD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



DXR R_1,R_2 [RRE, Extended Operands]



The first operand (the dividend) is divided by the second operand (the divisor), and the normalized quotient is placed in the first-operand location. No remainder is preserved.

Floating-point 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.

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 an increase of the characteristic by one. The fraction is then truncated to the proper result-fraction length.

An exponent-overflow exception is recognized when the characteristic of the final quotient would exceed 127 and the fraction is not zero. The operation is completed by making the characteristic 128 less than the correct value. If, for the DIVIDE (DXR) instruction, the low-order characteristic would also exceed 127, it, too, is decreased by 128. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent overflow occurs.

An exponent-underflow exception exists when the characteristic of the final quotient would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the quotient a true zero. For the DXR instruction, 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.

Exponent underflow does not occur when an operand characteristic 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, a floating-point-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 true zero. No exponent overflow or exponent underflow occurs.

The sign of the quotient is determined by the rules of algebra, except that the sign is always plus when the quotient is made a true zero.

The R_1 field for DER, DE, DDR, and DD, and the R_2 field for DER and DDR, must designate register 0, 2, 4, or 6. The R_1 and R_2 fields for DXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of DD and DE only)
- Exponent overflow
- Exponent underflow
- Floating-point divide
- Specification

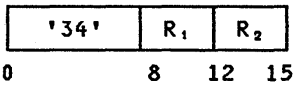
Programming Note

Examples of the use of the DIVIDE instruction are given in Appendix A.

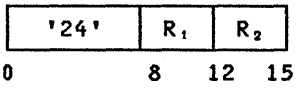
LzLabs GmbH. Ex. 1016-237

HALVE

HER R_1,R_2 [RR, Short Operands]



HDR R_1,R_2 [RR, Long Operands]



The second operand is divided by 2, and the normalized quotient is placed in the first-operand location.

The fraction of the second operand is shifted right one bit position, placing the contents of the rightmost bit position into the leftmost bit position of the guard digit and introducing a zero into 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 exponent-underflow exception exists when the characteristic of the final result would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the result a true zero.

When the fraction of the second operand is zero, the result is made a true zero, and no 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 true zero.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

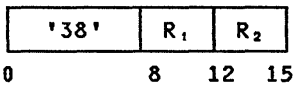
Exponent underflow
Specification

Programming Notes

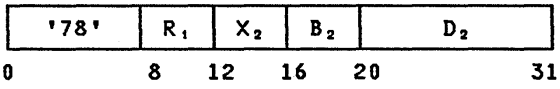
1. An example of the use of HALVE is given in Appendix A.
2. With short and long operands, the halve operation is identical to a divide operation with the number 2 as divisor. Similarly, the result of HDR is identical to that of MD or MDR with one-half as a multiplier. No multiply operation corresponds to HER, since no multiply operation produces short results.
3. The result of HALVE is zero only when the second-operand fraction is zero, or when exponent underflow occurs with the exponent-underflow mask set to zero. A fraction with zeros in every bit position, except for a one in the rightmost bit position, does not become zero after the right shift. This is because the one bit is preserved in the guard-digit position and, when the result is not made a true zero because of exponent underflow, becomes the leftmost bit after normalization of the result.

LOAD

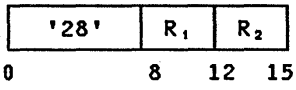
LER R_1,R_2 [RR, Short Operands]



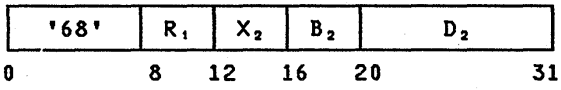
LE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



LDR R_1,R_2 [RR, Long Operands]



LD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



The second operand is placed unchanged in the first-operand location.

LzLabs GmbH. Ex. 1016-238

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

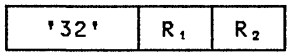
Condition Code: The code remains unchanged.

Program Exceptions:

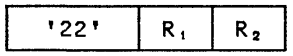
Access (fetch, operand 2 of LE and LD only)
Specification

LOAD AND TEST

LTER R_1,R_2 [RR, Short Operands]



LTDR R_1,R_2 [RR, Long Operands]



The second operand is placed unchanged in the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Exceptions:

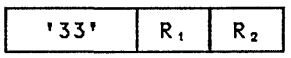
Specification

Programming Note

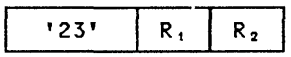
When the same register is specified as the first-operand and second-operand location, the operation is equivalent to a test without data movement.

LOAD COMPLEMENT

LCER R_1,R_2 [RR, Short Operands]



LCDR R_1,R_2 [RR, Long Operands]



The second operand is placed in the first-operand location with the sign bit inverted.

The sign bit is inverted, even if the fraction is zero. The characteristic and fraction are not changed.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

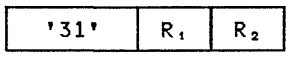
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Exceptions:

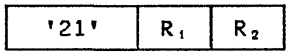
Specification

LOAD NEGATIVE

LNER R_1,R_2 [RR, Short Operands]



LNDR R_1,R_2 [RR, Long Operands]



The second operand is placed in the first-operand location with the sign made minus.

The sign bit is made one, even if the fraction is zero. The characteristic and fraction are not changed.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

LzLabs GmbH. Ex. 1016-239

Resulting Condition Code:

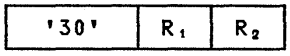
0 Result fraction is zero
1 Result is less than zero
2 --
3 --

Program Exceptions:

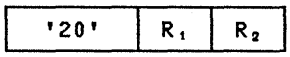
Specification

LOAD POSITIVE

LPER R_1,R_2 [RR, Short Operands]



LPDR R_1,R_2 [RR, Long Operands]



The second operand is placed in the first-operand location with the sign made plus.

The sign bit is made zero. The characteristic and fraction are not changed.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0 Result fraction is zero
1 --
2 Result is greater than zero
3 --

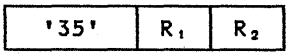
Program Exceptions:

Specification

LOAD ROUNDED

LRER R_1,R_2

[RR, Long Operand 2, Short Operand 1]



LRDR R_1,R_2

[RR, Extended Operand 2, Long Operand 1]



The second operand is rounded to the next shorter format, and the result is placed in the first-operand location.

Rounding consists in adding a one in bit position 32 or 72 of the long or extended second operand, respectively, and propagating any carry to the left. The sign of the fraction is ignored, and addition is performed as if the fractions 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.

The sign of the result is the same as the sign of the second operand. There is no normalization to eliminate leading zeros.

An exponent-overflow exception exists when shifting the fraction right would cause the characteristic to exceed 127. The operation is completed by loading a number whose characteristic is 128 less than the correct value, and a program interruption for exponent overflow occurs. The result is normalized, and the sign and fraction remain correct.

Exponent-underflow and significance exceptions cannot occur.

The R_1 field must designate register 0, 2, 4, or 6; the R_2 field of LRER must designate register 0, 2, 4, or 6; and the R_2 field of LRDR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

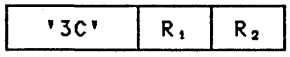
Exponent overflow
Specification

LzLabs GmbH. Ex. 1016-240

MULTIPLY

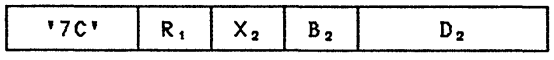
MER R_1,R_2

[RR, Short Multiplier and Multiplicand, Long Product]

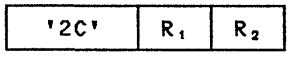


ME $R_1,D_2(X_2,B_2)$

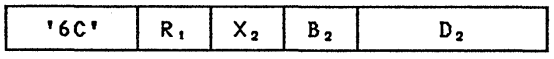
[RX, Short Multiplier and Multiplicand, Long Product]



MDR R_1,R_2 [RR, Long Operands]

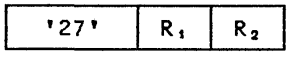


MD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



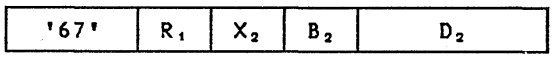
MXDR R_1,R_2

[RR, Long Multiplier and Multiplicand, Extended Product]

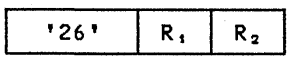


MXD $R_1,D_2(X_2,B_2)$

[RX, Long Multiplier and Multiplicand, Extended Product]



MXR R_1,R_2 [RR, Extended Operands]



The normalized product of the second operand (the multiplier) and the first operand (the multiplicand) is placed in the first-operand location.

Multiplication of two floating-point 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. When 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 MER and ME, 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 MDR and MD, the multiplier and multiplicand fractions have 14 digits, and the final product fraction is truncated to 14 digits. For MXDR and MXD, 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 exponent-overflow exception is recognized when the characteristic of the final product would exceed 127 and the fraction is not zero. The operation is completed by making the characteristic 128 less than the correct value. If, for extended results, the low-order characteristic would also exceed 127, it, too, is decreased by 128. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent overflow occurs.

Exponent overflow is not recognized when the intermediate-product characteristic is initially 128 but is brought back within range by normalization.

An exponent-underflow exception exists when the characteristic of the final product would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-

LzLabs GmbH. Ex. 1016-241

underflow mask bit is zero, program interruption does not take place; instead, the operation is completed by making the product a true zero. For extended results, exponent underflow is not recognized when the low-order characteristic would be less than zero but the high-order characteristic is equal to or greater than zero.

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.

When either or both operand fractions are zero, the result is made a true zero, and no exponent overflow or exponent underflow occurs.

The sign of the product is determined by the rules of algebra, except that the sign is always zero when the result is made a true zero.

The R_1 field for MER, ME, MDR, and MD, and the R_2 field for MER, MDR, and MXDR must designate register 0, 2, 4, or 6. The R_1 field for MXDR, MXD, and MXR, and the R_2 field for MXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

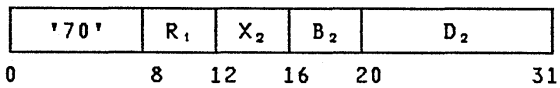
Access (fetch, operand 2 of ME, MD, and MXD only)
Exponent overflow
Exponent underflow
Specification

Programming Notes

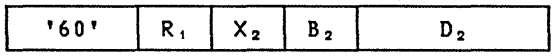
1. An example of the use of MULTIPLY is given in Appendix A.
2. Interchanging the two operands in a floating-point multiplication does not affect the value of the product.

STORE

STE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



STD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



The first operand is placed unchanged in the second-operand location.

The R_1 field must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

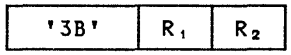
Condition Code: The code remains unchanged.

Program Exceptions:

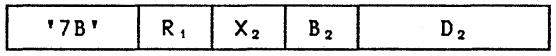
Access (store, operand 2)
Specification

SUBTRACT NORMALIZED

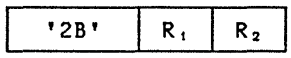
SER R_1,R_2 [RR, Short Operands]



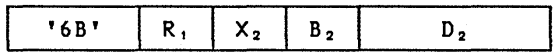
SE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



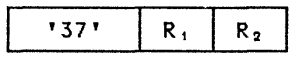
SDR R_1,R_2 [RR, Long Operands]



SD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



SXR R_1,R_2 [RR, Extended Operands]



The second operand is subtracted from the first operand, and the normalized difference is placed in the first-operand location.

The execution of SUBTRACT NORMALIZED is identical to that of ADD NORMALIZED, except that the second operand partic-

LzLabs GmbH. Ex. 1016-242

ipates in the operation with its sign bit inverted.

The R_1 field of SER, SE, SDR, and SD, and the R_2 field of SER and SDR must designate register 0, 2, 4, or 6. The R_1 and R_2 fields of SXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Resulting Condition Code:

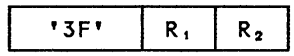
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Exceptions:

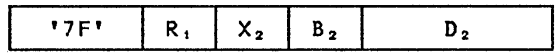
Access (fetch, operand 2 of SE and SD only)
Exponent overflow
Exponent underflow
Significance
Specification

SUBTRACT UNNORMALIZED

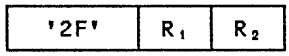
SUR R_1,R_2 [RR, Short Operands]



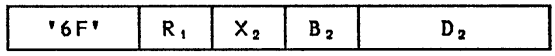
SU $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



SWR R_1,R_2 [RR, Long Operands]



SW $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



The second operand is subtracted from the first operand, and the unnormalized difference is placed in 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_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Exceptions:

Access (fetch, operand 2 of SU and SW only)
Significance
Specification

LzLabs GmbH. Ex. 1016-243

DIAGNOSE 10-4
EXTRACT PRIMARY ASN 10-5
EXTRACT SECONDARY ASN 10-5
INSERT ADDRESS SPACE CONTROL 10-6
INSERT PSW KEY 10-7
INSERT STORAGE KEY EXTENDED 10-7
INSERT VIRTUAL STORAGE KEY 10-7
INVALIDATE PAGE TABLE ENTRY 10-8
LOAD ADDRESS SPACE PARAMETERS 10-10
LOAD CONTROL 10-17
LOAD PSW 10-17
LOAD REAL ADDRESS 10-18
MOVE TO PRIMARY 10-19
MOVE TO SECONDARY 10-19
MOVE WITH KEY 10-20
PROGRAM CALL 10-22
PROGRAM TRANSFER 10-28
PURGE TLB 10-33
RESET REFERENCE BIT EXTENDED 10-33
SET ADDRESS SPACE CONTROL 10-33
SET CLOCK 10-34
SET CLOCK COMPARATOR 10-35
SET CPU TIMER 10-35
SET PREFIX 10-36
SET PSW KEY FROM ADDRESS 10-36
SET SECONDARY ASN 10-37
SET STORAGE KEY EXTENDED 10-40
SET SYSTEM MASK 10-40
SIGNAL PROCESSOR 10-41
STORE CLOCK COMPARATOR 10-42
STORE CONTROL 10-42
STORE CPU ADDRESS 10-43
STORE CPU ID 10-43
STORE CPU TIMER 10-43
STORE PREFIX 10-44
STORE THEN AND SYSTEM MASK 10-44
STORE THEN OR SYSTEM MASK 10-44
TEST BLOCK 10-45
TEST PROTECTION 10-47
TRACE 10-48

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 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 the figure "Summary of Control Instructions." The figure also indicates when the condition code is set and the 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

LzLabs GmbH. Ex. 1016-245

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 Note

The control instructions in the 370-XA mode differ from those offered in the System/370 mode in that the following instructions, which are offered either as standard or as optional in the System/370 mode, are not offered in the 370-XA mode:

CONNECT CHANNEL SET
DISCONNECT CHANNEL SET
INSERT STORAGE KEY
READ DIRECT
RESET REFERENCE BIT
SET STORAGE KEY
WRITE DIRECT

The following additional instruction is available only in the 370-XA mode:

TRACE

LzLabs GmbH. Ex. 1016-246

Name	Mnemonic	Characteristics										Op Code
DIAGNOSE			DM	P	DM							83
EXTRACT PRIMARY ASN	EPAR	RRE		Q		SO				R		B226
EXTRACT SECONDARY ASN	ESAR	RRE		Q		SO				R		B227
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								B229
INSERT VIRTUAL STORAGE KEY	IVSK	RRE		Q	A[1]	SO				R		B223
INVALIDATE PAGE TABLE ENTRY	IPTE	RRE		P	A[1]			$				B221
LOAD ADDRESS SPACE PARAMETERS	LASP	SSE	C	P	AS SP	SO						E500
LOAD CONTROL	LCTL	RS		P	A SP							B7
LOAD PSW	LPSW	S	L	P	A SP			¢				82
LOAD REAL ADDRESS	LRA	RX	C	P	A[1]					R		B1
MOVE TO PRIMARY	MVCP	SS	C	Q	A[1]	SO		¢			ST	DA
MOVE TO SECONDARY	MVCS	SS	C	Q	A[1]	SO		¢			ST	DB
MOVE WITH KEY	MVCK	SS	C	Q	A						ST	D9
PROGRAM CALL	PC	S		Q	AT	Z[1]	T	¢	GM	B	R	B218
PROGRAM TRANSFER	PT	RRE		Q	AT SP	Z[2]	T	¢		B		B228
PURGE TLB	PTLB	S		P				$				B20D
RESET REFERENCE BIT EXTENDED	RRBE	RRE	C	P								B22A
SET ADDRESS SPACE CONTROL	SAC	S			SP	SO		¢				B219
SET CLOCK	SCK	S	C	P	A SP							B204
SET CLOCK COMPARATOR	SCKC	S		P	A SP							B206
SET CPU TIMER	SPT	S		P	A SP							B208
SET PREFIX	SPX	S		P	A SP			$				B210
SET PSW KEY FROM ADDRESS	SPKA	S		Q								B20A
SET SECONDARY ASN	SSAR	RRE			AT	Z[3]	T	¢				B225
SET STORAGE KEY EXTENDED	SSKE	RRE		P				¢				B22B
SET SYSTEM MASK	SSM	S		P	A SP	SO						80
SIGNAL PROCESSOR	SIGP	RS	C	P				$		R		AE
STORE CLOCK COMPARATOR	STCKC	S		P	A SP						ST	B207
STORE CONTROL	STCTL	RS		P	A SP						ST	B6
STORE CPU ADDRESS	STAP	S		P	A SP						ST	B212
STORE CPU ID	STIDP	S		P	A SP						ST	B202
STORE CPU TIMER	STPT	S		P	A SP						ST	B209
STORE PREFIX	STPX	S		P	A SP						ST	B211
STORE THEN AND SYSTEM MASK	STNSM	SI		P	A						ST	AC
STORE THEN OR SYSTEM MASK	STOSM	SI		P	A SP						ST	AD
TEST BLOCK	TB	RRE	C	P	A[1]	II		$	G0	R		B22C
TEST PROTECTION	TPROT	SSE	C	P	A[1]							E501
TRACE	TRACE	RS		P	A SP		T	¢				99

Summary of Control Instructions (Part 1 of 2)

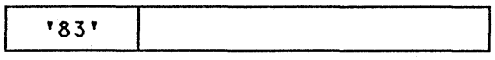
LzLabs GmbH. Ex. 1016-247

Explanation:

¢	Causes serialization and checkpoint synchronization
$	Causes serialization
A	Access exceptions for logical addresses
A[1]	Access exceptions; not all access exceptions may occur; see instruction description for details
AS	Access exceptions and ASN-translation-specification exception; see instruction description for details
AT	ASN-translation exceptions (which include addressing, ASN-translation specification, AFX translation, and ASX translation)
B	PER branch event
C	Condition code is set
DM	Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code
G0	Instruction execution includes the implied use of general register 0
G2	Instruction execution includes the implied use of general register 2
GM	Instruction execution includes the implied use of general registers 2 and 14
II	Interruptible instruction
L	New condition code is loaded
P	Privileged-operation exception
Q	Privileged-operation exception for semiprivileged instructions
R	PER general-register alteration event
RR	RR instruction format
RRE	RRE instruction format
RS	RS instruction format
RX	RX instruction format
S	S instruction format
SI	SI instruction format
SO	Special-operation exception
SP	Specification exception
SS	SS instruction format
SSE	SSE instruction format
ST	PER storage-alteration event
T	Trace exceptions (which include trace table, addressing, and low-address protection)
Z[1]	Additional exceptions and events for PROGRAM CALL (which include addressing, EX-translation, LX-translation, PC-translation-specification, and special-operation exceptions and space-switch event)
Z[2]	Additional exceptions and events for PROGRAM TRANSFER (which include addressing, primary-authority, and special-operation exceptions and space-switch event)
Z[3]	Additional exceptions for SET SECONDARY ASN (which include addressing, secondary authority, and special operation)

Summary of Control Instructions (Part 2 of 2)

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.

Condition Code: The code is unpredictable.

Program Exceptions:

Privileged operation
Depending on the model, other exceptions may be recognized.

LzLabs GmbH. Ex. 1016-248

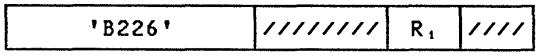
Programming Notes

1. Since the instruction is not intended for problem-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 R_1 [RRE]



The 16-bit PASN, bits 16-31 of control register 4, is placed in bit positions 16-31 of the general register designated by the R_1 field. 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 issued with DAT on; otherwise, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

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 the figure "Priority of Execution: EXTRACT PRIMARY ASN."

Condition Code: The code remains unchanged.

Program Exceptions:

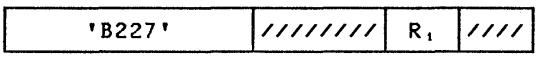
Privileged operation (extraction-authority control is zero in problem state)
Special operation

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.

Priority of Execution: EXTRACT PRIMARY ASN

EXTRACT SECONDARY ASN

ESAR R_1 [RRE]



The 16-bit SASN, bits 16-31 of control register 3, is placed in bit positions 16-31 of the general register designated by the R_1 field. 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 issued with DAT on; otherwise, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

LzLabs GmbH. Ex. 1016-249

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 the figure "Priority of Execution: EXTRACT SECONDARY ASN."

Condition Code: The code remains unchanged.

Program Exceptions:

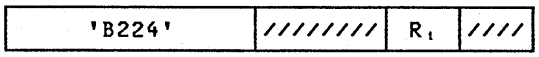
Privileged operation (extraction-authority control is zero in problem state)
Special operation

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.

Priority of Execution: EXTRACT SECONDARY ASN

INSERT ADDRESS SPACE CONTROL

IAC R_1 [RRE]



The address-space-control bit, bit 16 of the current PSW, is placed in bit position 23 of the general register designated by the R_1 field. Bits 16-22 of the register are set to zeros, and bits 0-15 and 24-31 of the register remain unchanged. The address-space-control bit is also used to set the condition code.

Bits 16-23 and 28-31 of the instruction are ignored.

Special Conditions

The instruction must be issued with DAT on; otherwise, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

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 the figure "Priority of Execution: INSERT ADDRESS SPACE CONTROL."

Resulting Condition Code:

0 PSW bit 16 is zero
1 PSW bit 16 is one
2 --
3 --

Program Exceptions:

Privileged operation (extraction-authority control is zero in 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.

Priority of Execution: INSERT ADDRESS SPACE CONTROL

Programming Notes

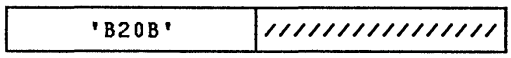
1. Bits 16-22 of the general register designated by the R_1 field are reserved for expansion for use with possible future facilities. The program should not depend on these bits being set to zero. Similarly, condition codes 2 and 3 may be set as a result of future facilities.

LzLabs GmbH. Ex. 1016-250

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 bit 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.

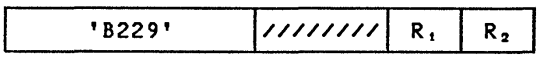
Condition Code: The code remains unchanged.

Program Exceptions:

Privileged operation (extraction-authority control is zero in problem state)

INSERT STORAGE KEY EXTENDED

ISKE R_1,R_2 [RRE]



The storage key for the block that is addressed by the contents of the general register designated by the R_2 field is inserted in the general register designated by the R_1 field.

Bits 16-23 of the instruction are ignored.

In 24-bit addressing mode, bits 8-19 of the register designated by the R_2 field specify a block of 4K bytes in real storage, and bits 0-7 and 20-31 of the register are ignored. In 31 bit addressing mode, bits 1-19 of the register designated by by R_2 field specify a block of 4K bytes 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 the register designated by the R_1 field, and bit 31 is set to zero. The contents of bit positions 0-23 of the register remain unchanged.

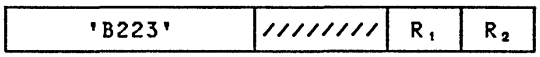
Condition Code: The code remains unchanged.

Program Exceptions:

Addressing (operand 2)
Privileged operation

INSERT VIRTUAL STORAGE KEY

IVSK R_1,R_2 [RRE]



The storage key for the location designated by the virtual address in the general register designated by the R_2 field is inserted in the general register designated by the R_1 field.

Bits 16-23 of the instruction are ignored.

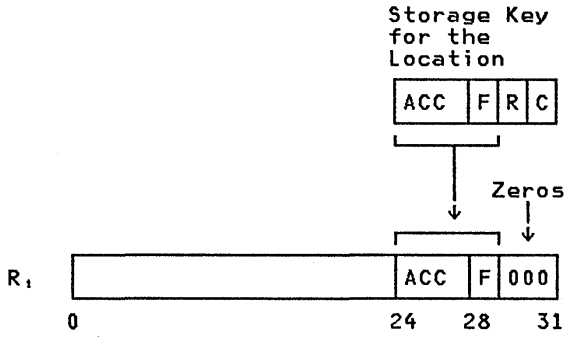
Selected bits of the register designated by the R_2 field are used as a virtual address. In the 24-bit addressing mode, the address is designated by bits 8-31 of the register, and bits 0-7 are ignored. In the 31-bit addressing mode, the address is designated by bits 1-31, and bit 0 is ignored.

The address is a virtual address and is subject to the address-space-control bit, bit 16 of the current PSW. In the primary-space mode, the address is treated as a primary virtual address; in the secondary-space mode, the address is treated as a secondary virtual address. The reference to the storage key is not subject to a protection exception.

LzLabs GmbH. Ex. 1016-251

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 the register designated by the R_1 field, 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 issued 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 the figure "Priority of Execution: INSERT VIRTUAL STORAGE KEY."

Condition Code: Unchanged.

Program Exceptions:

Access (except for protection, address specified by R_2)
Privileged operation (extraction-authority control is zero in 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.

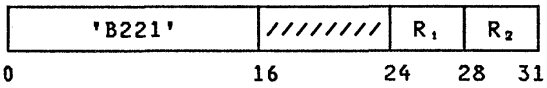
Priority of Execution: INSERT VIRTUAL STORAGE KEY

Programming Note

Since all bytes in a 4K-byte block are associated with the same page and the same storage key, bits 20-31 of the general register specified by R_2 essentially are ignored.

INVALIDATE PAGE TABLE ENTRY

IPTE R_1,R_2 [RRE]



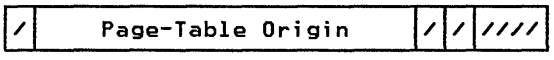
The designated page-table entry is invalidated, and the translation-lookaside-buffers (TLBs) in all CPUs in the configuration are cleared of the associated entries.

The contents of the register designated by the R_1 field have the format of a segment-table entry with only the page-table origin used. The contents of the register designated by the R_2 field 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.

Graphically, the contents of the registers just described are as follows:

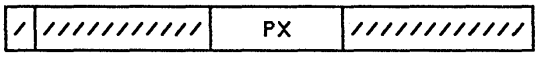
LzLabs GmbH. Ex. 1016-252

R_1



0 1 26 28 31

R_2



0 1 12 20 31

The page-table origin and the page index designate a page-table entry, following 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 using the rules for 31-bit arithmetic, regardless of the setting of the addressing mode, specified by bit 32 of the current PSW. Carries into bit position 0 as a result of the addition of the page index and page-table origin are 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 availability or format errors. Additionally, the page-frame real address contained in the entry is not checked for an addressing exception.

The entire page-table entry is fetched concurrently from storage. 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. No access by another CPU is permitted to the byte location between the moment the byte is fetched and the moment the byte is stored.

A serialization function is performed on the CPU which executes this instruction. The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs, and then the page-table entry is fetched. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed.

If successful in setting the page-invalid bit to one, 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 the first operand
- The page index specified by the second operand
- 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 on 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 specified page-frame real address. Others may clear all entries which contain the specified page index, and some implementations may clear precisely the minimum number of entries required. Therefore, in order for a program to run 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,

LzLabs GmbH. Ex. 1016-253

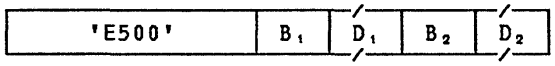
when in the attached state, ever contained a page-frame real address that is different from the current value, copies of the previous values may remain in the TLB.

3. INVALIDATE PAGE TABLE ENTRY cannot be safely used to update a shared location in main storage if the possibility exists that 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.

LOAD ADDRESS SPACE PARAMETERS

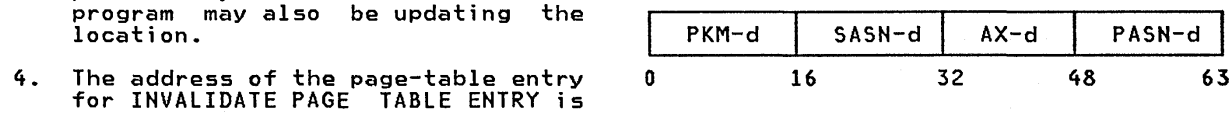
LASP $D_1(B_1),D_2(B_2)$ [SSE]



The contents of 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. These steps, when successful, obtain additional values, which are loaded into control registers 1, 5, and 7. When the steps are not successful, no control registers are changed, and the reason is indicated in the condition code.

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 is translated by means of the address-space tables to obtain a PSTD, LTD, and, optionally, an AX. The secondary ASN is translated by means of the address-space tables to obtain an SSTD, and, optionally, an authority check is 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:



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	
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	Use AX associated with PASN.	Use AX from first operand.
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 loaded from the PSTD, and no authorization is performed.

The operation of LOAD ADDRESS SPACE PARAMETERS is depicted in the figure "Execution of LOAD ADDRESS SPACE PARAMETERS."

PASN Translation

In the PASN translation process, the PASN-d is translated by means of the ASN first table and the ASN second table.

LzLabs GmbH. Ex. 1016-254

The STD and LTD fields and, optionally, the AX field, obtained from the ASN-second-table entry are subsequently used to update the corresponding control registers.

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 LTD-old are left unchanged in the control registers and become the PSTD-new and LTD-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.

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 ASN-second-table entry, when ones, do not result in an ASN-translation exception, and the space-switch-event-control bit in the ASN-second-table entry, when one, does not result in a space-switch event. When either of the invalid bits is one, condition code 1 is set. When the ASN-second-table entry is valid and either the current space-switch-event-control bit in control register 1 is one or the space-switch-event-control bit in the ASN-second-table entry is one, 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 ASN-second-table entry which is accessed as a result of the PASN translation are referred to as AX-p, STD-p, and LTD-p, respectively.

SASN Translation

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 ASN-second-table entry is subsequently used to update the secondary-segment-table designation (SSTD) in control register 7. The ATO and ATL fields obtained are used in the SASN authorization, if it occurs.

SASN translation is performed only when SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d, the SSTD-new is set to the same value as PSTD-new. When SASN-d is equal to SASN-old, bit 29 (force ASN translation) is zero, and bit 31 (skip SASN authorization) is one, then SASN translation is not performed, and SSTD-old becomes SSTD-new.

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 ASN-second-table entry, when ones, do not result in an ASN-translation exception. When either or both of the invalid bits are ones, condition code 2 is set, and the control registers remain unchanged.

The contents of the STD, ATO, and ATL fields in the ASN-second-table entry which is accessed as a result of the SASN translation are referred to as STD-s, ATO-s, and ATL-s, respectively.

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. 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 normal rules for secondary authorization, except that, if the SASN is not authorized, condition code 2 is set, and none of the control registers are updated.

Control-Register Loading

When the PASN-translation, SASN-translation, and SASN-authorization functions, if called for in the operation, are performed without encountering any exceptions, the operation 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 and SASN fields in control register 3 are replaced by the PKM-d and SASN-d fields from the first-operand location.

The PASN, bits 16-31 of control register 4, is replaced by the PASN-d field from the first-operand location.

The authorization index, bits 0-15 of control register 4, is replaced as follows:

When bit 30 of the second-operand address is one, from AX-d.

LzLabs GmbH. Ex. 1016-255

- When bit 30 of the second-operand address is zero and PASN translation is performed, from AX-p.
- When bit 30 of the second-operand address is zero and PASN translation is not performed, the authorization index is not changed.

The primary-segment-table designation in control register 1 and the linkage-table designation in control register 5 are replaced as follows:

- When PASN translation is performed, the primary-segment-table designation in control register 1 and the linkage-table designation in control register 5 are replaced from the STD-p and LTD-p fields, respectively, which are obtained during PASN translation.
- When PASN translation is not performed, the primary-segment-table-designation and linkage-table-designation fields remain unchanged.

The contents of the secondary-segment-table designation in control register 7 are replaced as follows:

- When SASN-d equals PASN-d, by the new contents of control register 1, the primary-segment-table designation.
- When SASN translation is performed, by the contents of the SSTD-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 or the ASN-second-table entries, condition code 1 is set, and the control registers are not changed.

When (1) PASN translation is called for and completed and (2) either the current space-switch-event-control bit, bit 0 of control register 1 is one or the space-switch-event-control bit in the ASN-second-table entry 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 or ASN-second-table entries, or because SASN authorization is called for and the SASN is not authorized, 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.

The figures "Summary of Actions: LOAD ADDRESS SPACE PARAMETERS" and "Priority of Execution: LOAD ADDRESS SPACE PARAMETERS" 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 not available; parameters not loaded
2 Secondary ASN not available or not authorized; 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)
ASN-translation specification
Privileged operation
Special operation
Specification

LzLabs GmbH. Ex. 1016-256

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.	Execution of PASN translation (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 having the value one
10.3	ASN-translation-specification exception due to invalid ones (bits 28-31) in ASN-first-table entry.
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 having the value one.
10.6	ASN-translation-specification exception due to invalid ones (bits 30, 31, 60-63) in ASN-second-table entry.
10.7	Condition code 3 due to either the old or new space-switch-event-control bit having the value one.
11.	Execution of SASN translation (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 having the value one.
11.3	ASN-translation-specification exception due to invalid ones (bits 28-31) in ASN-first-table entry.
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 having the value one.
11.6	ASN-translation-specification exception due to invalid ones (bits 30, 31, 60-63) in ASN-second-table entry.
12.	Execution of secondary authorization (when performed).
12.1	Condition code 2 due to authority-table entry being outside table.
12.2	Addressing exception for access to authority-table entry.
12.3	Condition code 2 due to S bit in authority-table entry being zero.

Priority of Execution: LOAD ADDRESS SPACE PARAMETERS

LzLabs GmbH. Ex. 1016-257

PASN-d Equals PASN-old	Second-Operand-Address Bits*		PASN Translation Performed	Result Field					
	29	30		PSTD-new	AX-new	LTD-new	PKM-new	SASN-new	PASN-new
Yes	0	0	No	PSTD-old	AX-old	LTD-old	PKM-d	SASN-d	PASN-d
Yes	0	1	No	PSTD-old	AX-d	LTD-old	PKM-d	SASN-d	PASN-d
Yes	1	0	Yes	STD-p	AX-p	LTD-p	PKM-d	SASN-d	PASN-d
Yes	1	1	Yes	STD-p	AX-d	LTD-p	PKM-d	SASN-d	PASN-d
No	-	0	Yes	STD-p	AX-p	LTD-p	PKM-d	SASN-d	PASN-d
No	-	1	Yes	STD-p	AX-d	LTD-p	PKM-d	SASN-d	PASN-d

Summary of Actions: LOAD ADDRESS SPACE PARAMETERS (Part 1 of 2)

SASN-d Equals PASN-d	SASN-d Equals SASN-old	Second-Operand-Address Bits*		SASN Translation Performed	SASN Authorization Performed#	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
No	Yes	-	0	Yes	Yes	STD-s
No	No	-	1	Yes	No	STD-s
No	No	-	0	Yes	Yes	STD-s

Explanation:

- Action in this case is the same regardless of the outcome of this comparison or of the setting of this bit.

SASN authorization is performed using ATO-s, ATL-s, and AX-new.

* Second-operand-address bits:
  29 Force ASN translation
  30 Use AX from first operand
  31 Skip secondary authority test

Summary of Actions: LOAD ADDRESS SPACE PARAMETERS (Part 2 of 2)

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, 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. See the figure "Summary of Abbreviations" for a listing of abbreviations used in this instruction description.

LzLabs GmbH. Ex. 1016-258

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
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

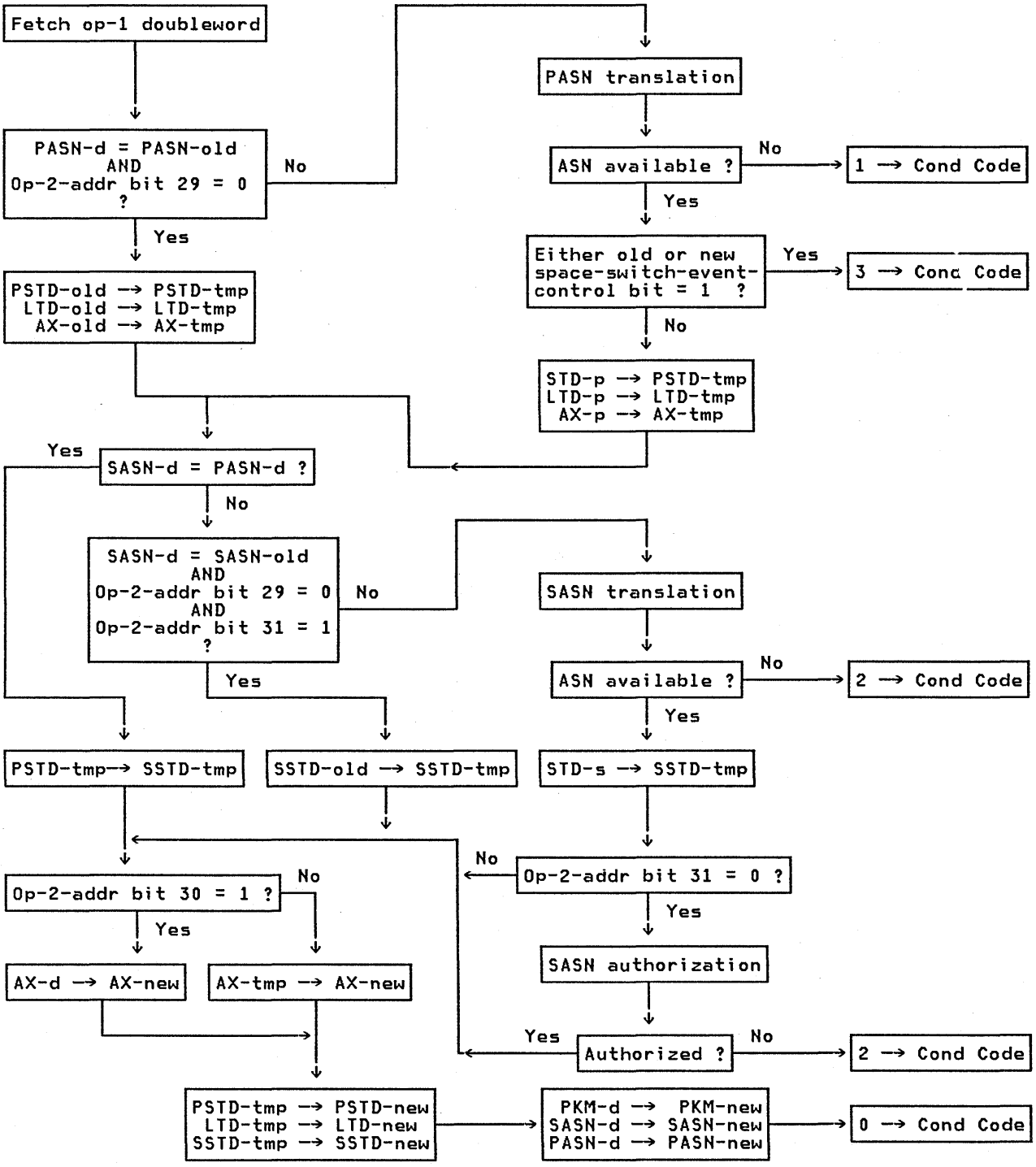
Field in ASN-Second-Table Entry	Abbreviation Used for the Field When Accessed as Part of	
	PASN Translation	SASN Translation
1-29	-	ATO-s
32-47	AX-p	-
48-59	-	ATL-s
64-95	STD-p	STD-s
96-127	LTD-p	-

Explanation:

- The field is not used in this case.

Summary of Abbreviations

LzLabs GmbH. Ex. 1016-259

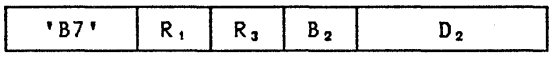


Execution of LOAD ADDRESS SPACE PARAMETERS

LzLabs GmbH. Ex. 1016-260

LOAD CONTROL

LCTL $R_1,R_3,D_2(B_2)$ [RS]



The set of control registers starting with the control register designated by the R_1 field and ending with the control register designated by the R_3 field 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 the control register designated by the R_1 field and continuing up to and including the control register designated by the R_3 field, with control register 0 following control register 15. The second operand remains 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 (fetch, operand 2)
Privileged operation
Specification

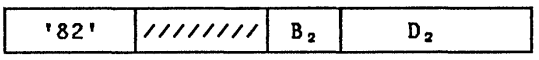
Programming Notes

1. To ensure that existing programs run 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 loading unnecessary control registers. In loading control registers 9-11, the model attempts to optimize for the case when the bits of control register 9 are zeros.

LOAD PSW

LPSW $D_2(B_2)$ [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 at the beginning and also at the completion of the operation. The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit channels and other CPUs to access the results.

When the operation is completed, a second serialization and checkpoint-synchronization function is performed, as follows. The CPU operation is delayed until all storage accesses due to this instruction have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, for this instruction are canceled.

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 the newly loaded PSW contains a zero in bit position 12, or the contents of bit positions 0, 2-4, 17, and 24-31 are not all zeros, or 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 zero.

LzLabs GmbH. Ex. 1016-261

The specification exception, which in this case is listed as a program exception in this instruction, is described in the section "Early Exception Recognition" in Chapter 6, Interruptions." 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.

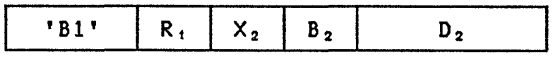
Condition Code: The code is set as specified in the new PSW loaded.

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 the general register designated by the R_1 field.

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.

The translation is performed by using the contents of control register 1 as a segment-table designation when bit 16 of the current PSW is zero or the contents of control register 7 as a segment-table designation when bit 16 of the current PSW is one.

The translation is performed without the use of the translation-lookaside buffer (TLB). Sufficient zeros are appended on the left of the resultant real address to produce a 32-bit result, which is then placed in the general register designated by the R_1 field. 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. A carry into bit position 0 as a result of the addition done either to compute the address of the segment-table entry or page-table entry may be ignored or may result in an addressing exception.

Condition code 0 is set when translation can be completed, that is, when the entry in each table lies within the specified table length and its I bit is zero.

When the I bit in the segment-table entry is one, condition code 1 is set, and the real address of the segment-table entry is placed in the register designated by the R_1 field. When the I bit in the page-table entry is one, condition code 2 is set, and the real address of the page-table entry is placed in the register designated by the R_1 field. When either the segment-table entry or the page-table entry is outside the table, condition code 3 is set, and the register designated by the R_1 field contains the real address of the entry that would have been fetched if the length violation did not occur. In all these cases, zeros are appended on the left of the real address, and the 32-bit result is placed in the register.

Special Conditions

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 the I bit off and has a format error.

A carry into bit position 0 as a result of the addition done either to compute the address of the segment-table entry or 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- or page-table length exceeded

Program Exceptions:

Addressing (segment-table entry or page-table entry)
Privileged operation
Translation specification

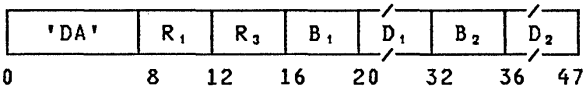
LzLabs GmbH. Ex. 1016-262

Programming Note

Caution must be observed in the execution of LOAD REAL ADDRESS in a multiprocessing configuration. Since INVALIDATE PAGE TABLE ENTRY may turn on the I bit in storage before causing the corresponding entries in 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 accesses 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 occur between the completion of LOAD REAL ADDRESS and the TLB being cleared of the entry.

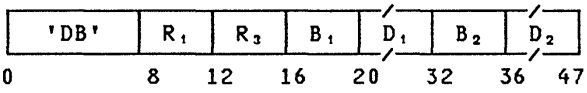
MOVE TO PRIMARY

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 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 by ignoring the state of the address-space-control bit in the current PSW.

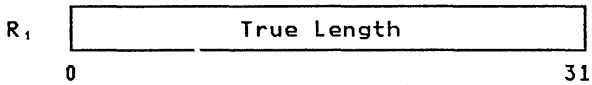
For MOVE TO PRIMARY, movement is to the primary space from the secondary space. The first-operand address is translated using the primary segment table, and the second-operand address is translated using the secondary segment table.

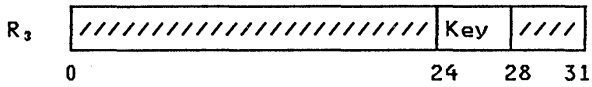
For MOVE TO SECONDARY, movement is to the secondary space from the primary space. The first-operand address is translated using the secondary segment table, and the second-operand address is translated using the primary segment table.

Bit positions 24-27 of the general register specified by the R_3 field are used as the secondary-space access key. Bit positions 0-23 and 28-31 of the register are ignored.

The contents of the general register specified by the R_1 field are a 32-bit unsigned value called the true length.

Graphically, 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.

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 virtual or 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 at the beginning and also at the completion of the operation.

LzLabs GmbH. Ex. 1016-263

The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results.

When the operation is completed, a second serialization and checkpoint-synchronization function is performed, as follows. The CPU operation is delayed until all storage accesses due to this instruction have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, for this instruction are canceled, and the results of all stores for this instruction are released, if held exclusive, to permit other CPU and channel programs to access the results.

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. The special-operation exception is recognized in both the problem and supervisor states.

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 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 the figure "Priority of Execution: MOVE TO PRIMARY and MOVE TO SECONDARY."

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 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 or to DAT being off.
8.	Privileged-operation exception due to selected PSW-key-mask bit being zero in problem state.
9.	Completion due to length zero.
10.	Access exceptions for operands.

Priority of Execution: MOVE TO PRIMARY and MOVE TO SECONDARY

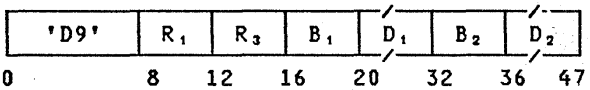
Programming Notes

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 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, or MOVE LONG.

MOVE WITH KEY

MVCK $D_1(R_1,B_1),D_2(B_2),R_3$ [SS]



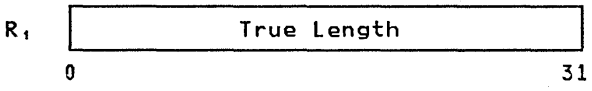
LzLabs GmbH. Ex. 1016-264

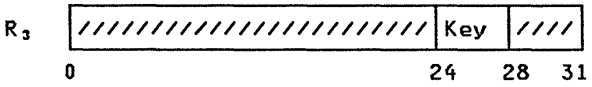
The first operand is replaced by the second operand. The fetch accesses to the second-operand location are performed using the key specified in the third operand, and the store accesses to the first-operand location are performed using the PSW key.

Bit positions 24-27 of the general register specified by the R_3 field are used as the source access key. Bit positions 0-23 and 28-31 of the register are ignored.

The contents of the general register specified by the R_1 field are a 32-bit unsigned value called the true length.

Graphically, 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.

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 supervisor state, any value for the source access key is valid.

The priority of the recognition of exceptions and condition codes is shown in the figure "Priority of Execution: MOVE WITH KEY Instruction."

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 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 being zero in problem state.
9.	Completion due to length zero.
10.	Access exceptions for operands.

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:

```
        LA      RW,256
LOOP    MVCK    D1(R1,B1),D2(B2),R3
        BC      8,END
        AR      B1,RW
        AR      B2,RW
        SR      R1,RW
        B       LOOP
END
```

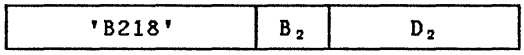
2. The performance of MOVE WITH KEY on most models may be significantly slower than that of the MOVE (MVC) and MOVE LONG instructions. Therefore, MOVE WITH KEY should not be

LzLabs GmbH. Ex. 1016-265

used if the key of the source and the target are the same.

PROGRAM CALL

PC $D_2(B_2)$ [S]



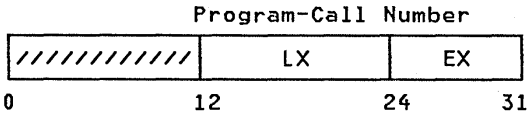
A two-level lookup is performed to locate an entry-table entry (ETE). The ETE contains an authorization key mask; an ASN; an entry parameter, which is loaded into general register 4; and information to update the PSW-key mask in control register 3 and to replace the problem-state bit, addressing bit, and instruction address in the PSW. The original contents of the control-register and the PSW fields are saved in general registers 3 and 14.

The ETE also causes a space-switching operation to occur if it specifies a nonzero ASN. When the ETE specifies a zero ASN, the operation is called PROGRAM CALL to current primary (PC-cp); when the ETE specifies 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). From this ASTE, a new PSTD, AX, and LTD are loaded into control registers 1, 4, and 5, respectively. Whether or not space switching is specified, the previous PASN and PSTD are placed in the SASN and SSTD, respectively, and the previous PASN is saved in general register 3.

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 program-call number and have the following format:

Second-Operand Address:

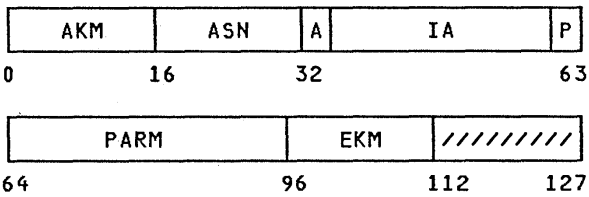


Linkage Index (LX): Bits 12-23 of the second-operand address are the linkage index and are used to select an entry from the linkage table designated by the linkage-table designation in control register 5.

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 the figure "Execution of PROGRAM CALL." The detailed definition for this table-lookup process is in the section "PC-Number Translation" in Chapter 5, "Program Execution." The entry-table entry has the following format:



When bit 32 of the ETE is zero (24-bit addressing mode), then bits 33-39 of the ETE must be zeros; otherwise, a PC-translation exception is recognized.

After the entry-table entry 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 entry-table entry 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. When PROGRAM CALL is executed in 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.

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.

The current PSTD, bits 0-31 of control register 1, is placed in control register 7 to become the current SSTD.

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.

Bits 32-62 of the current PSW (the addressing bit and the updated instruction address) are placed in bit posi-

LzLabs GmbH. Ex. 1016-266

tions 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, with a zero appended on the right, are placed in PSW bit positions 32-63 (the addressing bit and the instruction address). Bit 63 of the ETE is placed in PSW bit position 15 (the problem-state bit).

Bits 64-95 of the ETE (the entry parameter) are loaded into general register 4.

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.

PROGRAM CALL to Current Primary (PC-cp)

If bits 16-31 of the ETE (the ASN) are zeros, a PROGRAM CALL to current primary (PC-cp) is specified, and the operation is completed after performing those actions as described above.

The PC-cp operation is depicted in parts 1 and 2 of the figure "Execution of PROGRAM CALL."

PROGRAM CALL with Space Switching (PC-ss)

If the ASN in the ETE is nonzero, a PROGRAM CALL with space switching (PC-ss) instruction is specified, and the ASN is translated by means of a two-level table lookup.

The PC-ss operation is depicted in parts 1, 2 and 3 of the figure "Execution of PROGRAM CALL." The PC-ss operation is completed as follows:

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 six-bit ASX to index into the ASN second table specified by the AFX. The ASN table-lookup process is described in the section "ASN Translation" in Chapter 3, "Storage." The exceptions associated with ASN translation are collectively called ASN-translation exceptions. These exceptions and their priority are described in Chapter 6, "Interruptions."

Bits 16-31 of the entry-table entry are placed in bit positions 16-31 of control register 4 as the new PASN.

Bits 64-95 of the ASN-second-table entry (the STD) are loaded into control register 1 as the new PSTD.

Bits 32-47 of the ASN-second-table entry (the AX) are loaded into bit positions 0-15 of control register 4 as the new authorization index.

Bits 96-127 of the ASN-second-table entry (the LTD) are loaded into control register 5 as the new linkage-table designation.

PROGRAM CALL Serialization

For both the PC-cp and PC-ss operations, a serialization and checkpoint-synchronization function is performed at the beginning and also at the completion of the operation.

The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results.

When the operation is completed, a second serialization and checkpoint-synchronization function is performed, as follows. The CPU operation is delayed until all storage accesses due to this instruction have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, for this instruction are canceled.

Special Conditions

The instruction can be executed only when the CPU is in primary-space mode and the subsystem-linkage control, bit 0 of control register 5, is one. If the CPU is in real mode or secondary-space mode, or if the subsystem-linkage control is zero, a special-operation exception is recognized. In addition, the PC-ss instruction can be executed only when the ASN-translation control, bit 12 of control register 14, is one. If PC-ss is attempted with the ASN-translation control zero, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

When, for PC-ss, the 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.

LzLabs GmbH. Ex. 1016-267

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: PROGRAM CALL."

Condition Code: The code remains unchanged.

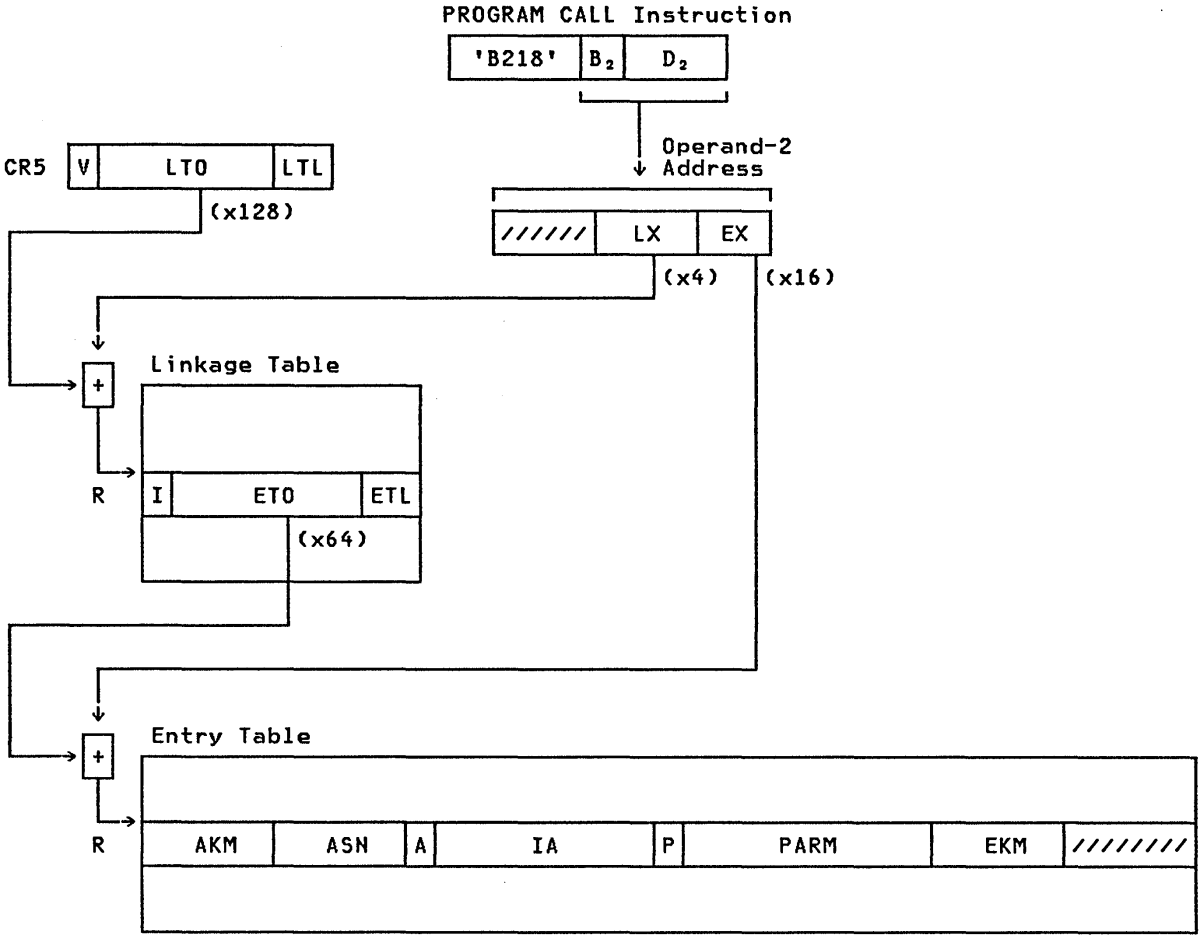
Program Exceptions:

Addressing (linkage-table entry or entry-table entry)
ASN translation (PC-ss only)
EX translation
LX translation
PC-translation specification
Privileged operation (AND of AKM and PSW-key mask is zero in problem state)
Space-switch event (PC-ss only)
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	Special-operation exception due to DAT being off, the CPU being in secondary-space mode, or the subsystem-linkage-control bit in control register 5 being zero.
8.A	Trace exceptions.
8.B.1	LX-translation exception due to linkage-table entry being outside table.
8.B.2	Addressing exception for access to linkage-table entry.
8.B.3	LX-translation exception due to I bit (bit 0) in linkage-table entry having the value one.
8.B.5	EX-translation exception due to entry-table entry being outside table.
8.B.6	Addressing exception for access to entry-table entry.
8.B.7	PC-translation-specification exception due to invalid combination (bit 32 is zero and bits 33-39 not zeros) in entry-table entry.
8.B.8	Privileged-operation exception due to a zero result from ANDing PSW-key mask and AKM in problem state.
8.B.9	Special-operation exception due to the ASN-translation control, bit 12 of control register 14, being zero. (PC-ss only)
8.B.10	ASN-translation exceptions. (PC-ss only)
9.	Space-switch event. (PC-ss only)

Priority of Execution: PROGRAM CALL

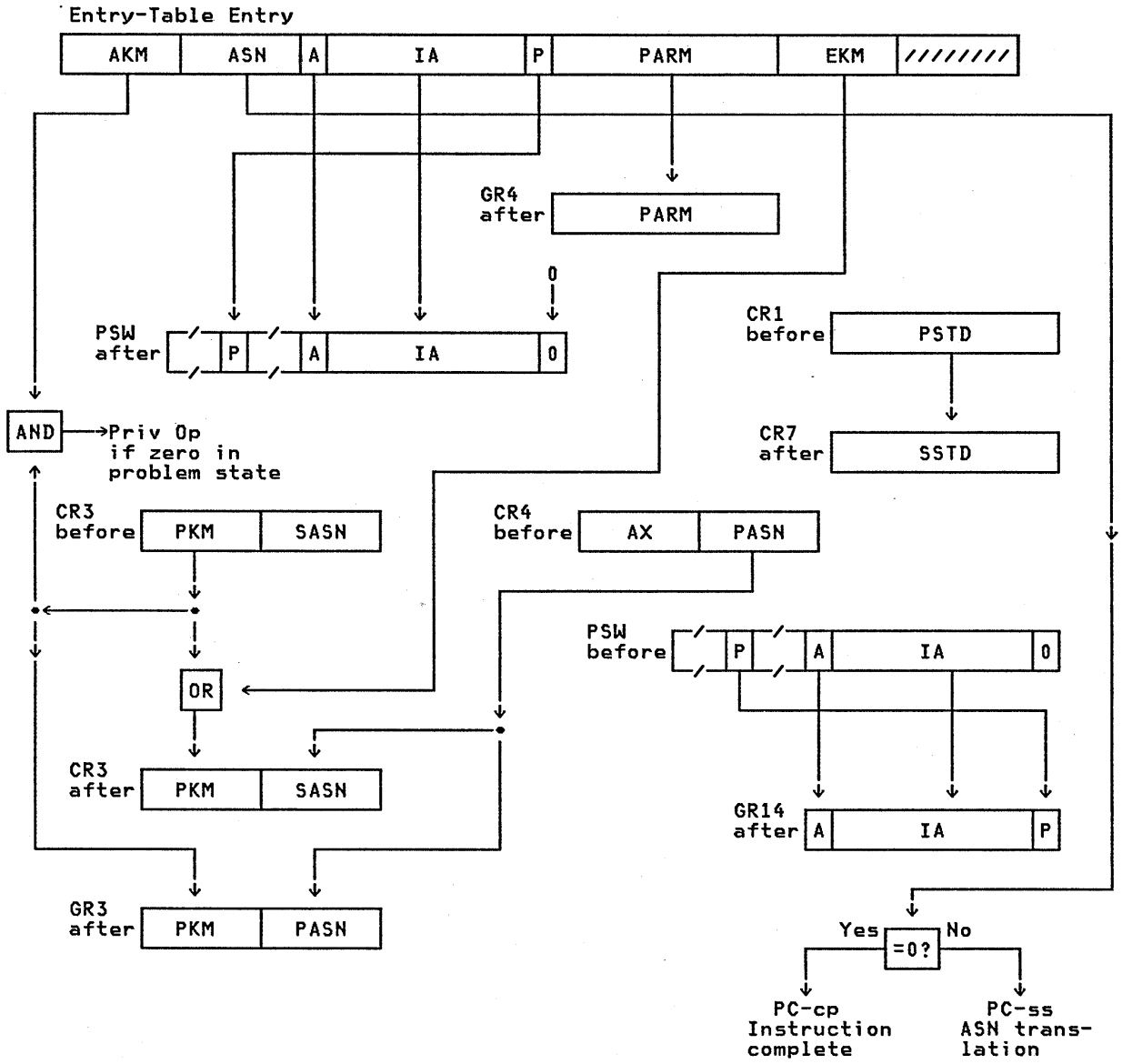
LzLabs GmbH. Ex. 1016-268



R: Address is real

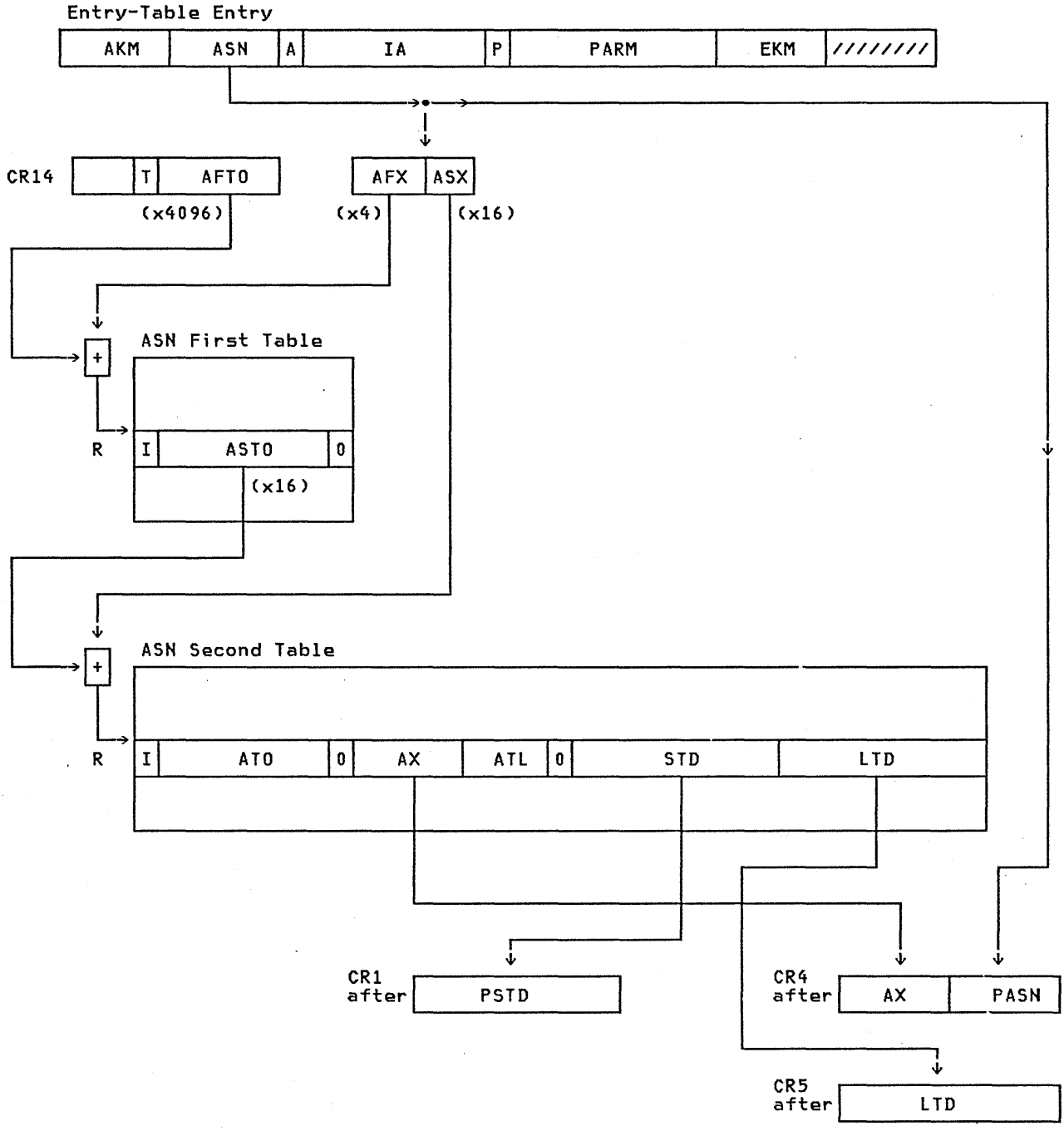
Execution of PROGRAM CALL (Part 1 of 3): PC-Number Translation

LzLabs GmbH. Ex. 1016-269



Execution of PROGRAM CALL (Part 2 of 3): PC-cp and PC-ss

LzLabs GmbH. Ex. 1016-270



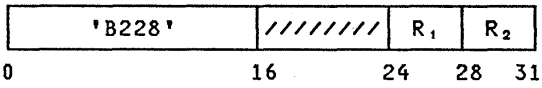
R: Address is real

Execution of PROGRAM CALL (Part 3 of 3): ASN Translation for PC-ss

LzLabs GmbH. Ex. 1016-271

PROGRAM TRANSFER

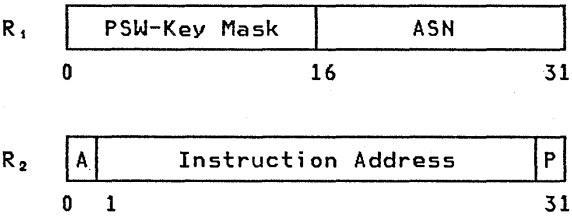
PT R_1,R_2 [RRE]



The contents of the general register specified by the R_1 field are used to update the PSW-key mask and the PASN. The contents of the register specified by the R_2 field are used to update the problem-state bit, addressing-mode bit, and instruction address in the current PSW.

Bits 16-23 of the instruction are ignored.

The format of the two registers specified by the R_1 and R_2 fields is as follows:



When the contents of bit positions 16-31 of the general register specified by the R_1 field 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 the general register specified by the R_2 field are used to update the problem-state bit, the addressing-mode bit, and the instruction address of the current PSW. Bit 31 of the general register specified by the R_2 field 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 the general register specified by the R_2 field replace the addressing-mode bit and the instruction address, bits 32-62 of the current PSW. Bit 63 of the PSW is set to zero.

Bits 0-15 of the general register specified by the R_1 field are ANDed with the PSW-key mask, bits 0-15 of control register 3, and the result replaces the contents of 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 the figure "Execution of PROGRAM TRANSFER." On a PT-cp operation, the operation is completed when the common portion of the PROGRAM TRANSFER operation, described above, is completed. The authorization index, PASN, primary STD, and linkage-table designation are not changed by PT-cp.

PROGRAM TRANSFER with Space Switching (PT-ss)

If the ASN in bits 16-31 of the register specified by the R_1 field is not equal to the current PASN, a PROGRAM TRANSFER with space switching (PT-ss) 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 the figure "Execution of PROGRAM TRANSFER." The PT-ss operation is completed as follows:

For a PT-ss, the contents of bit positions 16-31 of the general register specified by the R_1 field are used as an ASN, which is translated by means of a two-level table lookup.

Bits 16-25 of the general register are used as a 10-bit AFX to index into the ASN first table. Bits 26-31 are used as a six-bit ASX to index into the ASN second table. The ASN table-lookup process is described in the section "ASN Translation" in Chapter 3, "Storage." 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 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 the section "ASN Authorization" in Chapter 3, "Storage."

The PT-ss operation is completed by placing bits 64-95 of the ASN-second-table entry in both the PSTD and SSTD,

LzLabs GmbH. Ex. 1016-272

bit positions 0-31 of control registers 1 and 7, respectively. The contents of bit positions 32-47 of the ASN-second-table entry are placed in the authorization index, bit positions 0-15 of control register 4. The contents of bit positions 96-127 of the ASN-second-table entry are placed in the LTD, bit positions 0-31 of control register 5. The ASN, bits 16-31 of the general register specified by the R_1 field, is placed in the SASN and PASN, bit positions 16-31 of control registers 3 and 4.

PROGRAM TRANSFER Serialization

For both PT-cp and PT-ss, a serialization and checkpoint-synchronization function is performed at the beginning and also at the completion of the operation.

The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results.

When the operation is completed, a second serialization and checkpoint-synchronization function is performed, as follows. The CPU operation is delayed until all storage accesses due to this instruction have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, for this instruction are canceled.

Special Conditions

The instruction can be executed only when the CPU is in primary-space mode and the subsystem-linkage control, bit 0 of control register 5, is one. If the CPU is in real mode or secondary-space mode, or the subsystem-linkage control is zero, a special-operation exception is recognized.

Bit 31 of the general register specified by the R_2 field is placed in the problem-state-bit position, PSW bit position 15, unless the operation could 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 the register specify a valid combination for PSW bits 32-39. If the contents of bit position 0 of the general register specified by the R_2 field is zero and bits 1-7 are not 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 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 the figure "Priority of Execution: PROGRAM TRANSFER."

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (authority-table entry, PT-ss only)
- ASN translation (PT-ss only)
- Primary authority (PT-ss only)
- Privileged operation (attempt to set supervisor state in problem state)
- Space-switch event (PT-ss only)
- Special operation
- Specification
- Trace

LzLabs GmbH. Ex. 1016-273

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 the subsystem-linkage-control bit in control register 5 being zero.
8.A	Trace exceptions.
8.B.1	Privileged-operation exception due to attempt to set supervisor state when in problem state.
8.B.2	Specification exception due to a value of zero in bit 0 and a nonzero value in bits 1-7 of R_2.
8.B.3	Special-operation exception due to the ASN-translation control, bit 12 of control register 14, being zero. (PT-ss only)
8.B.4	ASN-translation exceptions. (PT-ss only)
8.B.5	Primary-authority exception due to authority-table entry being outside table. (PT-ss only)
8.B.6	Addressing exception for access to authority-table entry. (PT-ss only)
8.B.7	Primary-authority exception due to P bit in authority-table entry being zero. (PT-ss only)
9.	Space-switch event. (PT-ss only)

Priority of Execution: PROGRAM TRANSFER

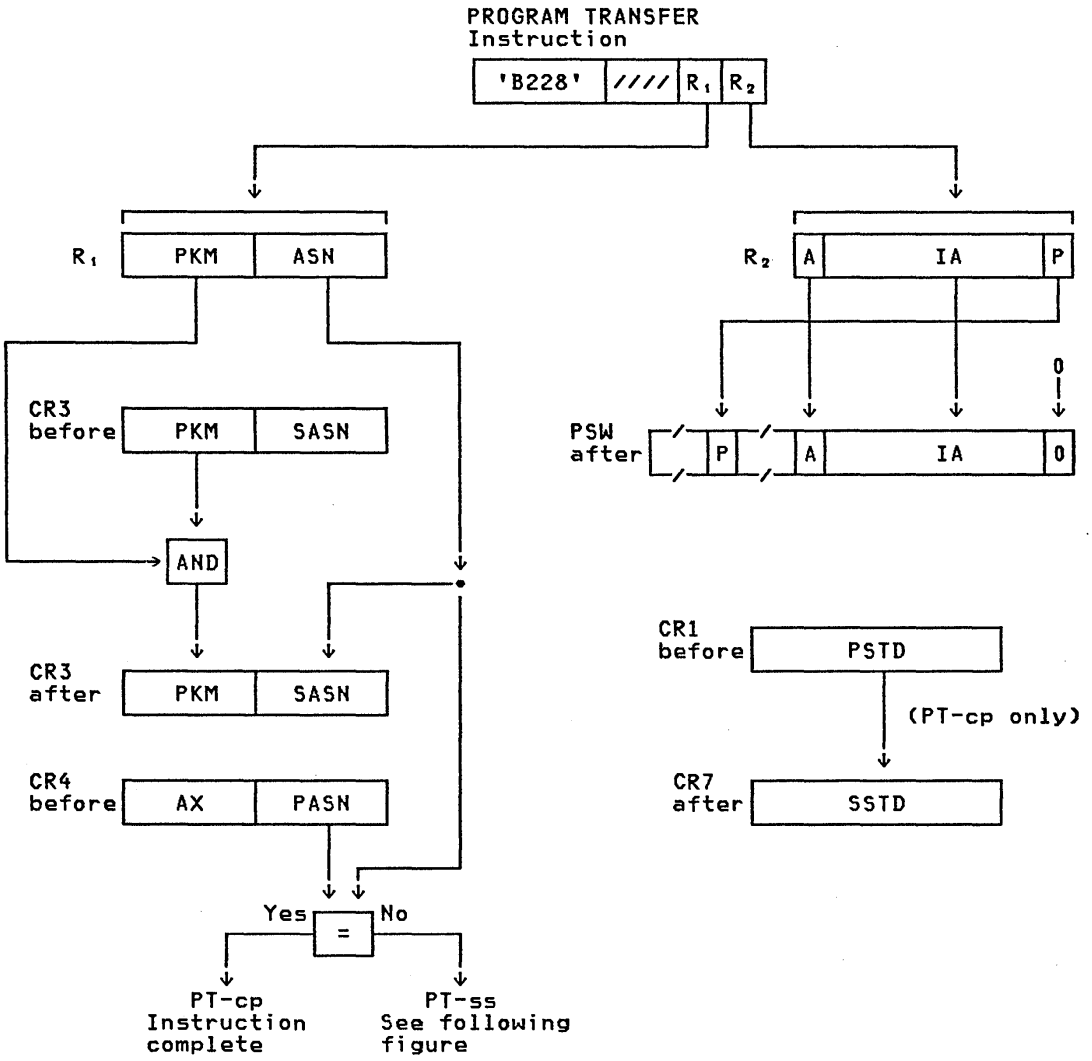
Programming Notes

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 PROGRAM CALL. 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 are made consistent with the restored PASN.

2. With proper authority, and while executing 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.

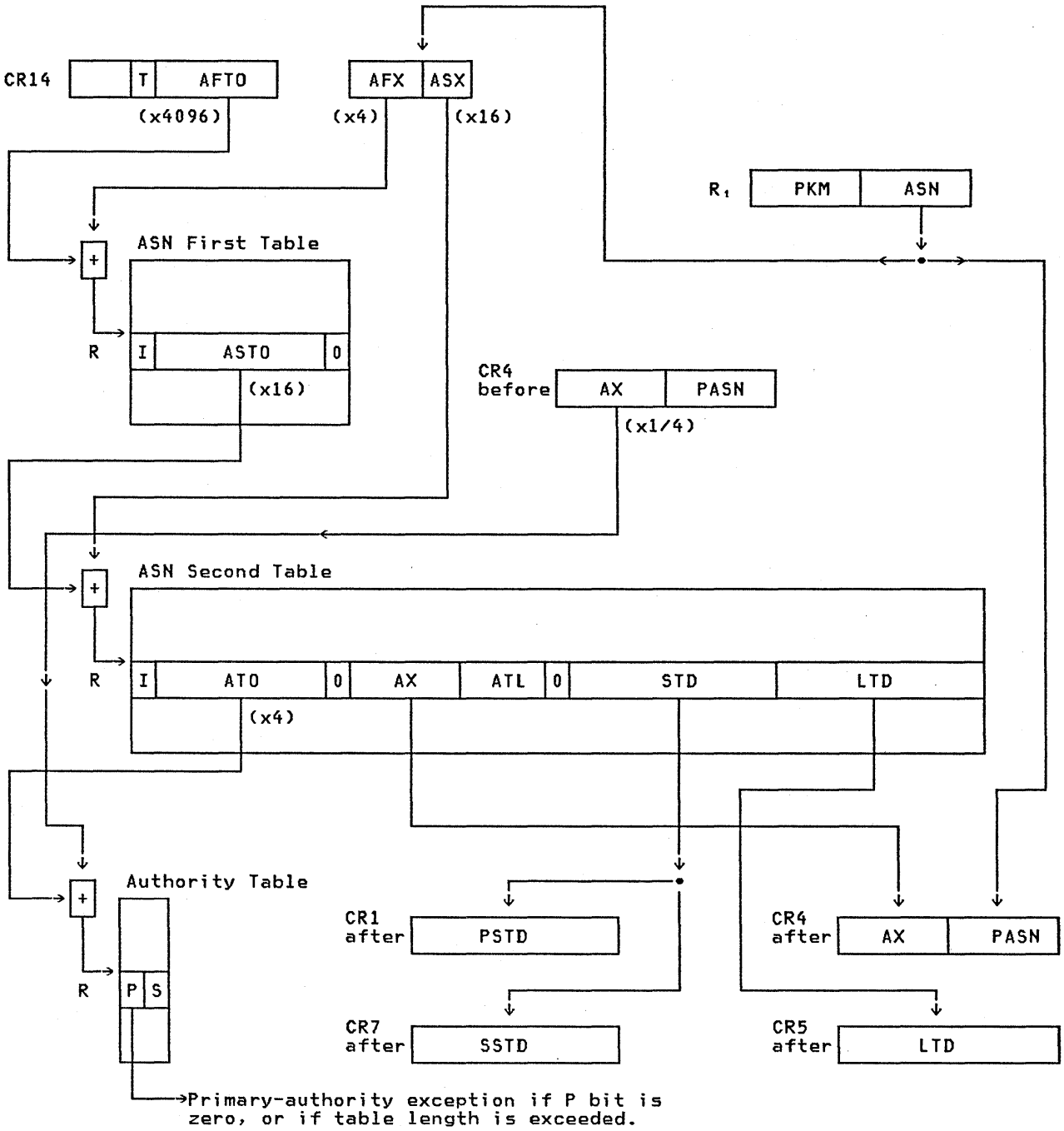
3. Unlike the RR-format branch instructions, a value of zero in the R_2 field for PROGRAM TRANSFER designates general register 0, and branching occurs.

LzLabs GmbH. Ex. 1016-274



Execution of PROGRAM TRANSFER (Part 1 of 2): PT-cp and PT-ss

LzLabs GmbH. Ex. 1016-275

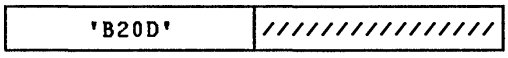


Execution of PROGRAM TRANSFER (Part 2 of 2): PT-ss

LzLabs GmbH. Ex. 1016-276

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. The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. No subsequent instructions, their operands, or dynamic-address-translation entries are fetched by this CPU until the execution of this instruction is complete.

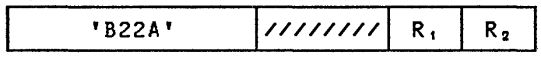
Condition Code: The code remains unchanged.

Program Exceptions:

Privileged operation

RESET REFERENCE BIT EXTENDED

RRBE R_1,R_2 [RRE]



The reference bit in the storage key for the 4K-byte block that is addressed by the contents of the general register designated by the R_2 field is set to zero. The contents of the general register designated by the R_1 field are ignored.

Bits 16-23 of the instruction are ignored.

In 24-bit addressing mode, bits 8-19 of the register designated by the R_2 field specify a block of 4K bytes in real storage, and bits 0-7 and 20-31 of the register are ignored. In 31 bit addressing mode, bits 1-19 of the register designated by the R_2 field specify a block of 4K bytes 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 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:

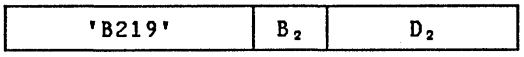
0 Reference bit zero, change bit zero
1 Reference bit zero, change bit one
2 Reference bit one, change bit zero
3 Reference bit one, change bit one

Program Exceptions:

Addressing (operand 2)
Privileged operation

SET ADDRESS SPACE CONTROL

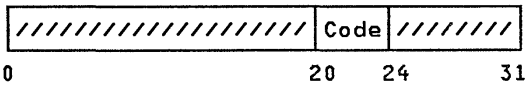
SAC $D_2(B_2)$ [S]



Bits 20-23 of the second-operand address are used as a code to set the address-space-control bit 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 20-22 of the second-operand address must be zero; otherwise, a specification exception is recognized.

The following figure summarizes the operation of SET ADDRESS SPACE CONTROL:

Second-Operand Address



Code	Name of Mode	Result in PSW Bit 16
0000	Primary space	0
0001	Secondary space	1
All others	Invalid	Unchanged

LzLabs GmbH. Ex. 1016-277

A serialization and checkpoint-synchronization function is performed at the beginning and also at the completion of the operation.

The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results.

When the operation is completed, a second checkpoint-synchronization function is performed, as follows. The CPU operation is delayed until all previous checkpoints, if any, for this instruction are canceled.

Special Conditions

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 special-operation exception is recognized in both the problem and supervisor states.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: SET ADDRESS SPACE CONTROL."

Condition Code: Unchanged.

Program Exceptions:

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	Special-operation exception due to DAT being off or the secondary-space control, bit 5 of control register 0, being zero.
8.	Specification exception due to nonzero value in bits 20-22 of the second-operand address.

Priority of Execution: SET ADDRESS SPACE CONTROL

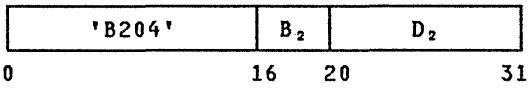
Programming Notes

1. SET ADDRESS SPACE CONTROL is 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 saved by INSERT ADDRESS SPACE CONTROL.

2. Predictable program operation is ensured in secondary mode only when the instructions are fetched from virtual-address locations which translate to the same real address by means of both the primary and secondary segment tables. Thus, a program should not enter secondary mode if it is not aware of the virtual-to-real mapping in both the primary and secondary spaces.

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, low-order bits of the second operand are ignored, and the corresponding positions of the clock which are implemented are set to zeros.

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). 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 either until the bit is set to zero or until any other running TOD clock in the configuration is incremented to a value of all zeros in bit positions 32-63.

LzLabs GmbH. Ex. 1016-278

When the TOD clock is shared by another CPU, the clock remains in the stopped state under control of the TOD-clock-sync control bit of the CPU which set the clock. If, while the clock is stopped, it is set by another CPU, then the clock comes under control of the TOD-clock-sync control bit of the CPU which last set the clock.

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. If the TOD-clock control is set to the secure position, the value and the state of the clock are not changed. The two results are 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 setting of the TOD-clock 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:

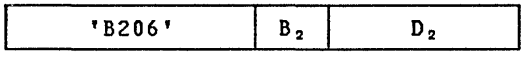
0 Clock value set
1 Clock value secure
2 --
3 Clock in not-operational state

Program Exceptions:

Access (fetch, operand 2)
Privileged operation
Specification

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.

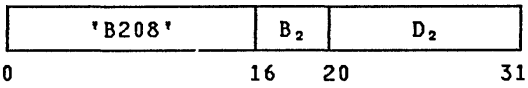
Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)
Privileged operation
Specification

SET 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.

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.

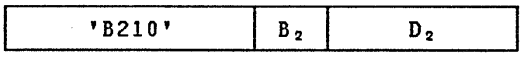
Program Exceptions:

Access (fetch, operand 2)
Privileged operation
Specification

LzLabs GmbH. Ex. 1016-279

SET PREFIX

SPX $D_2(B_2)$ [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 translation-lookaside buffer (TLB) of this CPU is 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 low-order 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, 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 bits to be turned on.

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 operand are ignored.

The translation-lookaside buffer (TLB) is cleared of entries. The TLB appears cleared of its original contents, beginning with the fetching of the next sequential instruction.

A serialization function is performed before or after the operand is fetched and also at the completion of the operation. The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. No subsequent instructions, operands, or dynamic-address-translation entries are fetched by this CPU until the execution of this instruction is completed.

Special Conditions

The operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

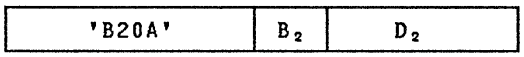
Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)
Addressing (new prefix area)
Privileged operation
Specification

SET PSW KEY FROM ADDRESS

SPKA $D_2(B_2)$ [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.

Condition Code: The code remains unchanged.

Program Exceptions:

Privileged operation (selected PSW-key-mask bit is zero in problem state)

Programming Notes

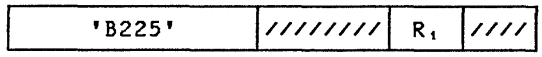
1. The format of SET PSW KEY FROM ADDRESS permits the program to set the PSW key either from the general register designated by the B_2 field or from the D_2 field in the instruction itself.

LzLabs GmbH. Ex. 1016-280

2. When one program requests another program to access a location specified 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 observed, 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 the general register specified by the R_1 field 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 0-15 of the register specified by the R_1 field are ignored.

The contents of bit positions 16-31 of the register specified by the R_1 field are called the new ASN. 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 the figure "Execution of SET SECONDARY ASN."

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 used as a 10-bit AFX to index into the ASN first table. Bits 10-15 of the new ASN (bits 26-31 of the register) are used as a six-bit ASX to index into the ASN second table. The two-level lookup is described in the section "ASN Translation" in Chapter 3, "Storage." The exceptions associated with ASN translation are collectively called "ASN-translation exceptions." These exceptions and their priority are described in Chapter 6, "Interruptions."

The AST entry 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 ASN second-table entry 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 entry in the authority table. The authority-table lookup is described in the section "ASN Authorization" in Chapter 3, "Storage."

The new ASN, bits 16-31 of the general register specified by the R_1 field, is placed in the SASN, bit positions 16-31 of control register 3. The segment-table designation, bits 64-95 of the AST entry, is placed in the SSTD, bits 0-31 of control register 7.

SET SECONDARY ASN Serialization

For both SSAR-cp and SSAR-ss, a serialization and checkpoint-synchronization function is performed at the beginning and also at the completion of the operation.

The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All

LzLabs GmbH. Ex. 1016-281

previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results.

When the operation is completed, a second serialization and checkpoint-synchronization function is performed, as follows. The CPU operation is delayed until all storage accesses due to this instruction have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, for this instruction are canceled.

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 special-operation exception is recognized in both the problem and supervisor states.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: SET SECONDARY ASN."

Condition Code: The code remains unchanged.

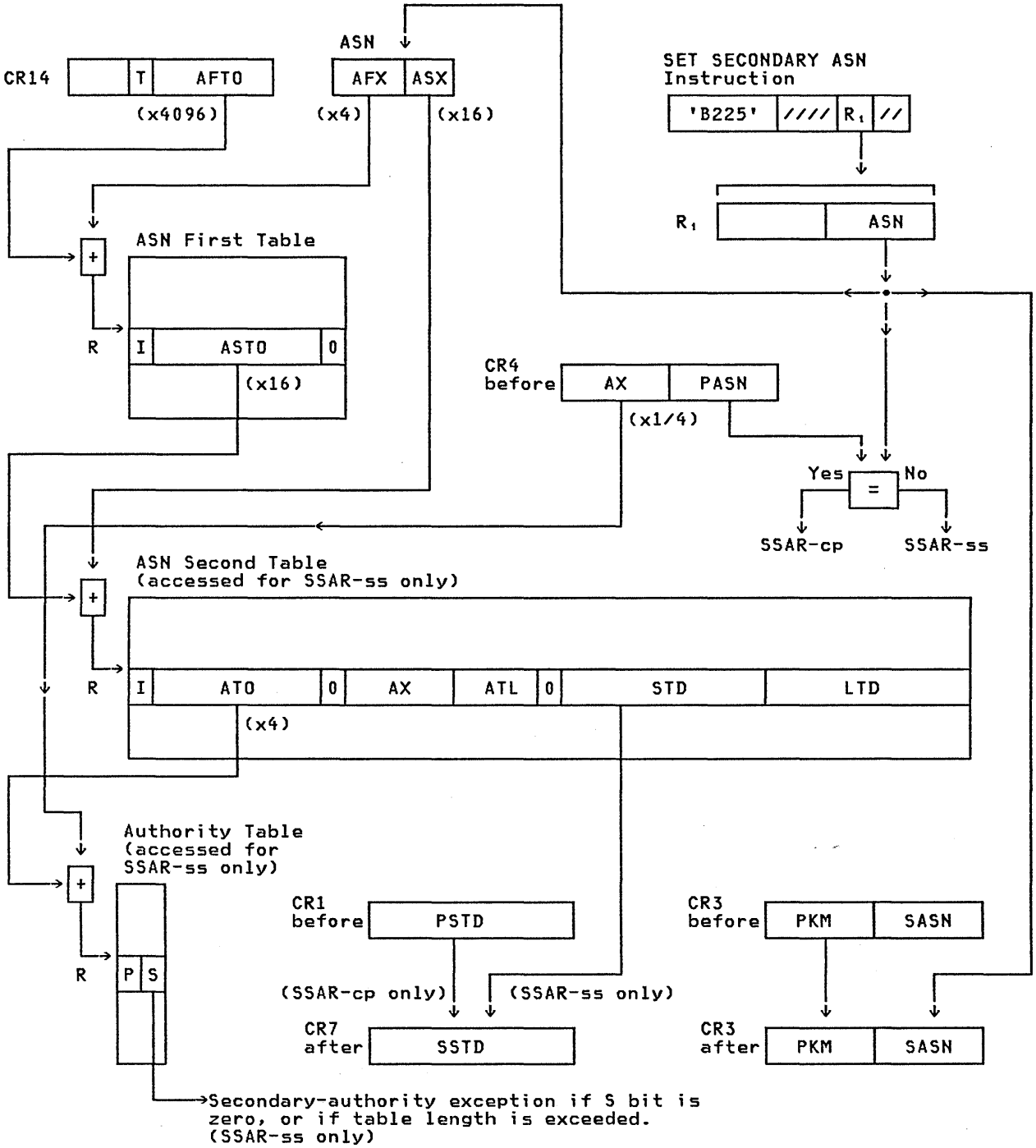
Program Exceptions:

Addressing (authority-table entry, SSAR-ss only)
ASN translation (SSAR-ss only)
Secondary authority (SSAR-ss only)
Special operation
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)
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)

Priority of Execution: SET SECONDARY ASN

LzLabs GmbH. Ex. 1016-282



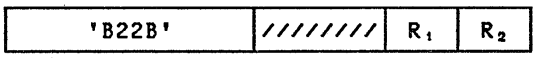
R: Address is real

Execution of SET SECONDARY ASN

LzLabs GmbH. Ex. 1016-283

SET STORAGE KEY EXTENDED

SSKE R_1,R_2 [RRE]



The storage key for the 4K-byte block that is addressed by the contents of the general register designated by the R_2 field are replaced by the contents of the general register designated by the R_1 field.

Bits 16-23 of the instruction are ignored.

In 24-bit addressing mode, bits 8-19 of the register designated by the R_2 field specify a block of 4K-byte bytes in real storage, and bits 0-7 and 20-31 of the register are ignored. In 31-bit addressing mode, bits 1-19 of the register designated by the R_2 field specify a block of 4K-byte bytes 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 new seven-bit storage-key value is obtained from bit positions 24-30 of the register designated by the R_1 field. The contents of bit positions 0-23 and 31 of the register are ignored.

A serialization and checkpoint-synchronization function is performed at the beginning and also at the completion of the operation.

The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results.

When the operation is completed, a second serialization and checkpoint-synchronization function is performed at the completion of the operation, as follows. The CPU operation is delayed until all storage accesses due to this instruction have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, for this instruction are canceled.

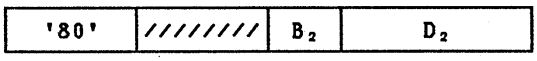
Condition Code: The code remains unchanged.

Program Exceptions:

Addressing (operand 2)
Privileged operation

SET SYSTEM MASK

SSM $D_2(B_2)$ [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 in this case is listed as a program exception in this instruction, is described in the section "Early Exception Recognition" in Chapter 6, Interruptions." 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.

Condition Code: The code remains unchanged.

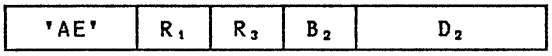
Program Exceptions:

Access (fetch, operand 2)
Privileged operation
Special operation
Specification

LzLabs GmbH. Ex. 1016-284

SIGNAL PROCESSOR

SIGP $R_1,R_3,D_2(B_2)$ [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 the section "CPU Signaling and Response" in Chapter 4, "Control."

The 16-bit binary number contained in bit positions 16-31 of the general register designated by the R_3 field forms the CPU address. Bits 0-15 of the register are ignored.

The register containing the 32-bit parameter is R_1 or R_1+1, whichever is the odd-numbered register. It depends on the order whether a parameter is provided and for what purpose it is used.

A serialization function is performed at the beginning and also at the completion of the operation. The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs, and then the signaling occurs. No subsequent instructions or their operands are accessed by this CPU until the execution of the instruction 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 the general register designated by the R_1 field, and condition code 1 is set.

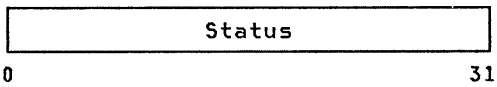
When the access path to the addressed CPU is busy, or the addressed CPU is operational but in a state where it cannot respond to the order code, condition code 2 is set.

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.

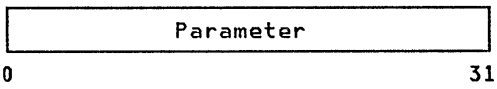
A more detailed discussion of the condition-code settings for SIGNAL PROCESSOR is contained in the section "CPU Signaling and Response" in Chapter 4, "Control."

The format of the operands of the instruction are illustrated below.

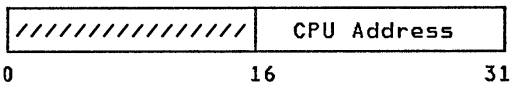
General register designated by R_1:



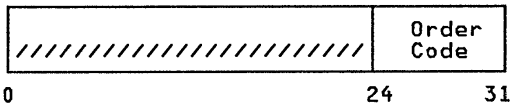
General register designated by R_1 or $R_1 + 1$, whichever is the odd-numbered register:



General register designated by R_3:



Second-operand address:



Resulting Condition Code:

0 Order code accepted
1 Status stored
2 Busy
3 Not operational

Program Exceptions:

Privileged operation

Programming Notes

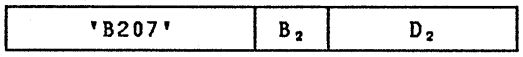
1. To ensure that presently written programs will be executed properly when new facilities using additional bits are installed, only zeros should appear in the unused bit positions of the second-operand address and in bit positions 0-15 of the register designated by the R_3 field.

LzLabs GmbH. Ex. 1016-285

2. 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 the performance therefore may be much slower than for the frequently used orders: restart, set prefix, store status at address, start, stop, stop and store status, 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.

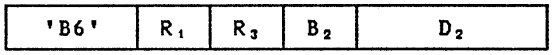
Condition Code: The code remains unchanged.

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 the control register designated by the R_1 field and ending with the control register designated by the R_3 field 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 the control register designated by the R_1 field and continuing up to and including the control register designated by the R_3 field, 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.

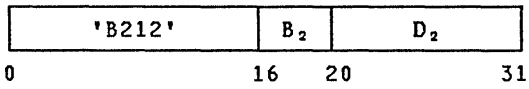
Program Exceptions:

Access (store, operand 2)
Privileged operation
Specification

LzLabs GmbH. Ex. 1016-286

STORE CPU ADDRESS

STAP $D_2(B_2)$ [S]



The CPU address 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.

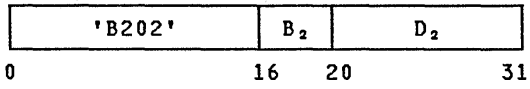
Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)
Privileged operation
Specification

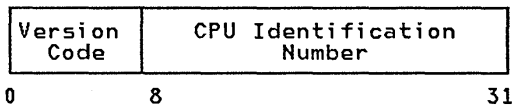
STORE CPU ID

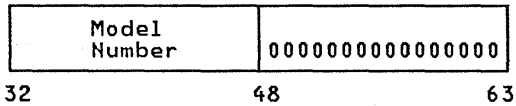
STIDP $D_2(B_2)$ [S]



Information identifying the CPU is stored at the doubleword location designated by the second-operand address.

The format of the information is as follows:





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 contents of the CPU-identification-number field, in conjunction with the model number, permit unique identification of the CPU.

Bit positions 32-47 contain the model number of the CPU. Bit positions 48-63 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

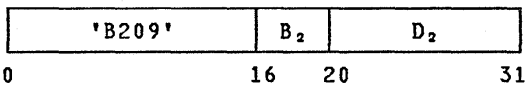
Programming Notes

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. The CPU identification number, in conjunction with the model number, provides a unique CPU identification that can be used in associating results with an individual machine, particularly in regard to functional differences, performance differences, and error handling.

STORE CPU TIMER

STPT $D_2(B_2)$ [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

LzLabs GmbH. Ex. 1016-287

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

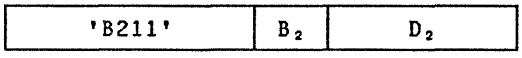
Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)
Privileged operation
Specification

STORE PREFIX

STPX $D_2(B_2)$ [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.

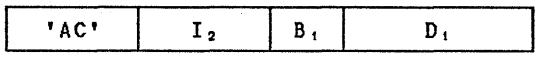
Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)
Privileged operation
Specification

STORE 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:

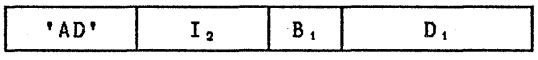
Access (store, operand 1)
Privileged operation

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.

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 in this case is listed as a program exception in this instruction, is described in the section "Early Exception Recognition" in Chapter 6, Interruptions." 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.

LzLabs GmbH. Ex. 1016-288

Condition Code: The code remains unchanged.

Program Exceptions:

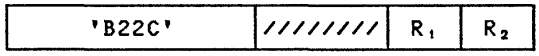
Access (store, operand 1)
Privileged operation
Specification

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.

TEST BLOCK

TB R_1,R_2 [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 block tested is addressed by the contents of the general register designated by the R_2 field. The test for usability is based on the susceptibility to the occurrence of invalid checking-block code.

Bits 16-23 of the instruction are ignored.

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.

The contents of the general register designated by the R_1 field are ignored. Bits 16-23 of the instruction are ignored.

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 24-bit addressing mode, bits 8-19 of the register designated by the R_2 field specify a block of 4K bytes in real storage, and bits 0-7 and 20-31 of the register are ignored. In 31-bit addressing mode, bits 1-19 of the register designated by the R_2 field specify a block of 4K bytes 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 and page protection. Low-address protection does apply. The operation is terminated on addressing and protection exceptions; that is, 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 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 issued 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 other than the value which was present at the time of the interruption, or (3) the block is accessed by another CPU or channel program 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.

LzLabs GmbH. Ex. 1016-289

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 is not significant. However, if, during the execution of TEST BLOCK for an unusable block, that block is accessed by another CPU or channel program, error conditions may be reported to either or both programs.

A serialization function is performed before the block is accessed and again after the operation is completed (or partially completed). The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs, and then the accesses, if any, to the block occur. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed (or partially completed), as observed by other CPU and channel programs.

The priority of the recognition of exceptions and condition codes is shown in the figure "Priority of Execution: TEST BLOCK."

Resulting Condition Code:

0 Block is usable, and the contents of storage in the block have been set to zeros
1 Block is 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.[1]
9.A	Condition code 1, block not usable.[1]
9.B	Protection exception due low-address protection.
9.C	Condition code 0, block usable and set to zeros.

Explanation:

[1] The operation is terminated on addressing and protection exceptions. In this case, the condition code may be unpredictable.

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-on-line procedure to determine if blocks of storage exist which should not be used.

3. The program should use TEST BLOCK when an error is detected in either the data or storage key of a block that has been in use and the program chooses to mark the block unusable. 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 such a state that subsequent prefetches or unintended references to the block do not cause machine-check conditions. The program should ignore the resulting condition code in this case since, depending on the model, on the type of error, and on the

LzLabs GmbH. Ex. 1016-290

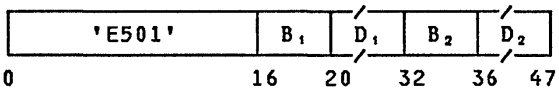
threshold implemented by the model, the condition code may indicate a usable block, even though the program has decided otherwise.

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 may be more effective than TEST BLOCK in validating the storage key.

TEST PROTECTION

TPROT $D_1(B_1),D_2(B_2)$ [SSE]



The location specified by the first-operand address is tested for protection exceptions using the access key specified in 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 and thus is subject to translation when DAT is on. 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 bits 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 if the first-operand effective address is less than 512, then a store access is not permitted. Page protection and fetch-protection override are also 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 DAT is on, an addressing exception is recognized when the address of the segment-table entry, the page-table entry, or the operand real address after translation designates a location which is not available in the configuration. Also, when DAT is on, a translation-specification exception is recognized when the segment-table entry or page-table entry has a format error. When DAT is off, only the addressing exception due to the operand real address applies. For all of these cases, the operation is suppressed.

Resulting Condition Code:

0 Both fetching and storing are permitted
1 Fetching is permitted, but storing is not
2 Neither fetching nor storing are permitted
3 Translation not available

Program Exceptions:

Addressing (operand 1)

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 specified by the first-operand address of the instruction. The instruction takes into consideration all of the protection mechanisms in the machine: key-controlled, page, fetch protection override, and low-address protection. Additionally, since segment translation and page translation may be a program substitute for a protection violation, these situations are

LzLabs GmbH. Ex. 1016-291

used to set the condition code rather than cause a program exception.

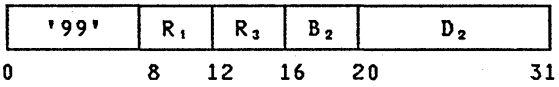
2. See the programming notes under SET PSW KEY FROM ADDRESS for more details and for an alternative approach to testing validity of addresses passed by a calling program. The approach using TEST PROTECTION has the advantage of a test which does not result in interruptions; however, the test and use are 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 page-translation and segment-translation exceptions. Instead, these situations are indicated by means of a condition-code setting. Situations which result in condition codes 1, 2, and 3 for LOAD REAL ADDRESS result in condition code 3 for TEST PROTECTION. However, the instructions differ in several respects. TEST PROTECTION has a logical address and thus is not subject to translation when DAT is off. LOAD REAL ADDRESS has a virtual address which is always translated. TEST PROTECTION may use the TLB for translation of the address, whereas LOAD REAL ADDRESS does not use the TLB.

   When DAT is off for LOAD REAL ADDRESS, the translation specification 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 situation 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 fetch. Since the instruction-fetch portion of an instruction is common for all instructions, access exceptions associated with instruction fetching are not described in the individual instruction definition.

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 specified 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 the register specified by R_1 and continuing up to and including the register specified by R_3, with register 0 following register 15. The trace table and the trace-entry formats are described in the section "Tracing" in Chapter 4, "Control."

When a trace entry is made, a serialization and checkpoint-synchronization function is performed at the beginning and also at the completion of the operation.

The CPU operation is delayed until all previous storage accesses by this CPU have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, are canceled, and the results of all previous stores are released, if held exclusive, to permit other CPU and channel programs to access the results.

When the operation is completed, a second serialization and checkpoint-synchronization function is performed, as follows: The CPU operation is delayed until all storage accesses due to this instruction have been completed, as observed by other CPU and channel programs. All previous checkpoints, if any, for this instruction are canceled, and the results of all stores for this instruction are released, if held exclusive, to permit other CPU and channel programs to access the results.

LzLabs GmbH. Ex. 1016-292

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.

LzLabs GmbH. Ex. 1016-293

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 11-4
  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 11-9
Check-Stop State 11-10
    System Check Stop 11-11
Machine-Check Interruption 11-11
  Exigent Conditions 11-11
  Repressible Conditions 11-11
  Interruption Action 11-12
  Point of Interruption 11-13
Machine-Check-Interruption Code 11-14
  Subclass 11-15
    System Damage 11-15
    Instruction-Processing Damage 11-15
    System Recovery 11-16
    Timing-Facility Damage 11-16
    Degradation 11-16
    Warning 11-16
    Channel Report Pending 11-17
    Service-Processor Damage 11-17
    Channel-Subsystem Damage 11-17
  Time of Interruption Occurrence 11-17
    Backed Up 11-17
  Synchronous Machine-Check Interruption Conditions 11-17
    Processing Backup 11-17
    Processing Damage 11-18
  Storage-Error Type 11-18
    Storage Error Uncorrected 11-18
    Storage Error Corrected 11-18
    Storage-Key Error Uncorrected 11-18
  Machine-Check Interruption-Code Validity Bits 11-19
    PSW-MWP Validity 11-19
    PSW Mask and Key Validity 11-19
    PSW Program-Mask and Condition-Code Validity 11-19
    PSW-Instruction-Address Validity 11-19
    Failing-Storage-Address Validity 11-19
    Floating-Point-Register Validity 11-19
    General-Register Validity 11-19
    Control-Register Validity 11-20
    Storage Logical Validity 11-20
    CPU-Timer Validity 11-20
    Clock-Comparator Validity 11-20
Machine-Check Extended Interruption Information 11-20
  Register-Save Areas 11-20
  Failing-Storage Address 11-21
Handling of Machine-Check Conditions 11-21
  Floating Interruption Conditions 11-21
    Floating Machine-Check-Interruption Conditions 11-21
    Floating I/O Interruptions 11-21
Machine-Check Masking 11-21
    Channel-Report-Pending Subclass Mask 11-22
    Recovery Subclass Mask 11-22
    Degradation Subclass Mask 11-22

LzLabs GmbH. Ex. 1016-295

Timing-Facility-Damage Subclass Mask 11-22
Warning Subclass Mask 11-22
Machine-Check Logout 11-22
Summary of Machine-Check Masking 11-22

The machine-check-handling mechanism provides extensive equipment-malfunction detection to ensure the integrity of system operation and to permit automatic recovery from some malfunctions. 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 entire contents of a checking block, including the redundancy, is called a 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

LzLabs GmbH. Ex. 1016-296

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. 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 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. The CPU operation is delayed until all conceptually previous storage accesses are completed, both for purposes of machine-check detection and as observed by other CPU and channel programs. The checkpoint information for all previous checkpoints, if any, is invalidated. Optionally, a new checkpoint is established. The CPU operation is delayed until all of these actions appear to be completed, as observed by other CPU and 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:

1. CPU reset.
2. All interruptions: external, I/O, machine check, program, restart, and supervisor call.
3. The BRANCH ON CONDITION (BCR) instruction with the M_1 and R_2

LzLabs GmbH. Ex. 1016-297

fields containing all ones and all zeros, respectively.

4. The instructions LOAD PSW, SET STORAGE KEY, SET STORAGE KEY EXTENDED, and SUPERVISOR CALL.

5. All I/O instructions.

6. The instructions MOVE TO PRIMARY, MOVE TO SECONDARY, PROGRAM CALL, PROGRAM TRANSFER, SET ADDRESS SPACE CONTROL, and SET SECONDARY ASN.

7. The three trace functions: branch tracing, ASN tracing, and explicit tracing.

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.

HANDLING OF MACHINE CHECKS

A machine check is caused by a machine malfunction and not by data or

LzLabs GmbH. Ex. 1016-298

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.

Specification 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:

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 in the location.

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

LzLabs GmbH. Ex. 1016-299

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 introduce a valid CBC into a register or 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 in 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 (a store into a checking block without replacing the entire checking block) by either a CPU or a channel program will be permitted.

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 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, one 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 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,

LzLabs GmbH. Ex. 1016-300

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 specified by register zero 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 will be presented, normally with a failing-storage address. The program in this case should initiate proceedings to eliminate use of, and validate the location, to avoid continued degradation.

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.

The figure "Invalid CBC in Storage Keys" 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. 1016-301

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[4], nonzero access key	MC[1]; preserve.	MC and preserve; or complete[3] and correct.
Fetch, zero access key[2]	MC or complete; preserve.	MC or complete; preserve.
Store[4], zero access key[2]	MC or complete; preserve.	MC and preserve; or complete[3] and correct.

Explanation:

[1] The contents of the main-storage location are not changed.

[2] The action shown for an access key of zero is also applicable to references to which key-controlled protection does not apply.

[3] The contents of the reference and change bits are set to ones if the "complete" action is taken.

[4] 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.

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 generated, but a system-recovery condition may be generated.

Invalid CBC in Storage Keys (Part 1 of 2)

LzLabs GmbH. Ex. 1016-302

Explanation (Continued):

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, and no machine-check condition is reported. • The operation is completed, and system recovery, with storage-key error uncorrected and a failing-storage address, is reported. • Instruction-processing damage (either with or without backup), with storage-key error uncorrected and a failing-storage address, 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, and no machine-check condition is reported. • The I/O operation is completed, and system recovery, with storage-key error uncorrected and a failing-storage-address, 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, and no machine-check condition is reported. • An I/O-error condition is reported; system recovery, with or without storage-key error uncorrected and a failing-storage address, is reported.
PD	Instruction-processing damage (either with or without backup), with or without a storage-key error uncorrected and failing-storage address, is reported.

Invalid CBC in Storage Keys (Part 2 of 2)

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, 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

LzLabs GmbH. Ex. 1016-303

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).

Control registers may be validated either singly or in groups by using the instruction LOAD CONTROL.

The CPU timer, clock comparator, and prefix register are validated by SET CPU TIMER, SET CLOCK COMPARATOR, and SET PREFIX, respectively.

The TOD clock is validated by SET CLOCK if the TOD-clock control is in the enable-set position.

Programming Note

Depending on the register, and the model, the contents of a register may be validated by the machine-check interruption or the model may require that a program issue 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 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.

LzLabs GmbH. Ex. 1016-304

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 the current instruction or interruption sequence cannot safely continue. Exigent conditions are of two classes: 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 are classified as two types, nullifying exigent conditions and terminating exigent conditions, according to whether the instructions affected are nullified or terminated. Exigent conditions for interruption sequences are classified as terminating exigent conditions. The terms "nullification" and "termination" have the same meaning as that used in Chapter 6, "Interruptions," 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 classes: recovery, alert, and repressible damage. Each class has 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 divided into three subclasses, according to the function affected: timing-facility damage, channel report pending, and channel-subsystem damage.

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 a damage condition as repressible does not imply

LzLabs GmbH. Ex. 1016-305

that the damage is necessarily less severe than damage classified as an exigent condition. 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 accepts 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. The PSW reflecting the point of interruption is stored as the machine-check old PSW at real location 48. The contents of other registers are stored in register-save areas at real locations 216-231 and 352-511. After the contents of the registers are stored in register-save areas, depending on the model, the registers may be validated with the contents being unpredictable. A failing-storage address may be stored at real location 248. A machine-check-interruption code (MCIC) of eight bytes is placed at real location 232. The new PSW is fetched from real location 112. Additionally, a machine-check logout may have occurred. The machine-generated addresses to access the old and new PSW, the MCIC, extended interruption information, and the fixed-logout area are all real addresses.

The fields accessed during the machine-check interruption are summarized in the figure "Machine-Check-Interruption Locations."

Information Stored (Fetched)	Starting Location*	Length in Bytes
Old PSW	48	8
New PSW (fetched)	112	8
Machine-check-interruption code	232	8
Failing-storage address	248	4
Register-save areas		
CPU timer	216	8
Clock comparator	224	8
Floating-point registers 0, 2, 4, 6	352	32
General registers 0-15	384	64
Control registers 0-15	448	64
Fixed logout area	256	16

Explanation:

* All locations are in real storage.

Machine-Check-Interruption Locations

LzLabs GmbH. Ex. 1016-306

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 in the same interruption code, even though 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.

LzLabs GmbH. Ex. 1016-307

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 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.

Programming Note

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.

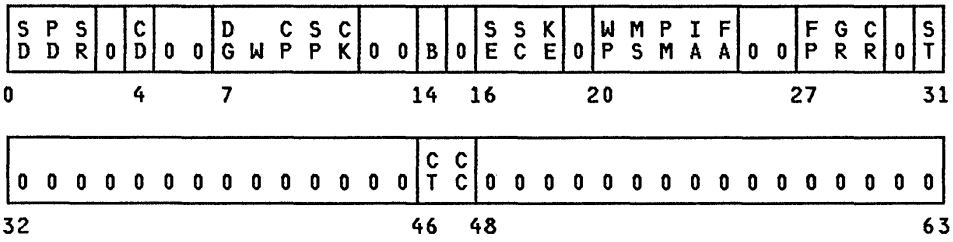
When all the status information stored as a result of an exigent machine-check-interruption condition does not reflect the same point, an attempt is made when possible to choose the point of interruption so that the instruction address which is stored in the machine-check old PSW is valid.

MACHINE-CHECK-INTERRUPTION CODE

On all machine-check interruptions, a machine-check-interruption code (MCIC) is stored at the doubleword starting at real location 232 and has the format shown in the figure "Machine-Check Interruption-Code Format."

Bits in the MCIC which are not assigned, or not implemented by a particular model, are stored as zeros.

LzLabs GmbH. Ex. 1016-308



Bits	Name
0	System damage (SD)
1	Instruction-processing damage (PD)
2	System recovery (SR)
4	Timing-facility damage (CD)
7	Degradation (DG)
8	Warning (W)
9	Channel report pending (CP)
10	Service-processor damage (SP)
11	Channel-subsystem damage (CK)
14	Backed up (B)
16	Storage error uncorrected (SE)
17	Storage error corrected (SC)
18	Storage-key error uncorrected (KE)
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)
27	Floating-point-register validity (FP)
28	General-register validity (GR)
29	Control-register validity (CR)
31	Storage logical validity (ST)
46	CPU-timer validity (CT)
47	Clock-comparator validity (CC)

Note: All other bits of the MCIC are unassigned and stored as zeros.

Machine-Check Interruption-Code Format

Programming Note

The program should not depend on unassigned bits in the machine-check-interruption code being zeros, so as to ensure that existing programs run if and when new facilities using these bits are defined.

SUBCLASS

Bits 0-2, 4, and 7-11 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 of the defined 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 remaining bits in the machine-check-interruption code are not meaningful, and information stored in the register-save areas, machine-check extended-interruption fields, and failing-storage-address field is not meaningful. System damage is a terminating exigent condition.

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 the section "Synchronous Machine-Check-

LzLabs GmbH. Ex. 1016-309

Interruption Conditions" in this chapter.

Instruction-processing damage is a nullifying or terminating exigent condition.

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.

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, the CPU timer, the clock comparator, 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 CPU timer or clock comparator may be validated by the interruption which reports the CPU timer or clock comparator as invalid.

Timing-facility damage is a repressible condition.

Programming Note

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 issued. 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.

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 depends on the model.

Degradation is a repressible condition.

Warning

Bit 8 (W), when one, indicates that damage is imminent in some part of the

LzLabs GmbH. Ex. 1016-310

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.

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 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.

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.

Service-Processor Damage

Bit 10 (SP), when one, indicates that damage has occurred to the service processor. When service-processor damage occurs, a machine-check interruption is made pending at all CPUs in the configuration. Service-processor damage has no subclass-mask bit and is a repressible condition.

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 has no subclass-mask bit and is a floating repressible condition.

TIME OF INTERRUPTION OCCURRENCE

Bit 14 of the machine-check-interruption code indicates when the interruption occurred in relation to the error.

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 depends on the model.

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 validity bits associated with CPU status are indicated as valid, the machine has successfully returned to a checkpoint prior to the malfunction, and no damage has yet occurred. The validity bits in the machine-check-interruption code which must be one for this to be the case are as follows:

LzLabs GmbH. Ex. 1016-311

MCIC Bit	Fields Covered by Bit
20	PSW MWP bits
21	PSW mask and key
22	PSW program mask and condition code
23	PSW instruction address
27	Floating-point registers
28	General registers
29	Control registers
31	Storage logical validity (result fields within current checkpoint interval)
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, 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-ERROR TYPE

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. The failing-storage-address field, when indicated as valid, identifies an address within the storage checking block containing the error, or, for storage-key error uncorrected, within the block associated with the storage key. 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 program or another CPU or channel program, or 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 for this CPU program or another CPU or channel program, or its contents may have been prefetched for a CPU or a channel program, or fetched as the result of a model-dependent storage access. The presence and extent of the storage-error-correction capability depends on the model.

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 has not been changed. The storage key may have been accessed or prefetched for this CPU program or another CPU or channel program, or accessed as the result of a model-dependent storage access.

Programming Note

The storage-error-uncorrected and storage-key-error-uncorrected bits do not in themselves indicate the occur-

LzLabs GmbH. Ex. 1016-312

rence of damage because the error detected may not have affected a result. The accompanying subclass bits of the interruption code indicate the area affected by the error.

MACHINE-CHECK INTERRUPTION-CODE VALIDITY BITS

Bits 20-24, 27-31, 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, 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, CPU timer, and clock comparator, are unpredictable until these registers are validated. (See the section "Invalid CBC in Registers" earlier in this chapter.)

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, 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. 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.

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 the floating-point registers 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.

LzLabs GmbH. Ex. 1016-313

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 I/O operations, or of the storing of the old PSW and other interruption information.

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.

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 three register-validity bits, the two timing-facility-validity bits, and the storage-logical-validity bit are all ones.
- The damage-subclass bits 0, 4, and 11 are zeros.
- The instruction-processing-damage bit is zero or, if one, the backed-up bit is also one.

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. 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 five 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 five 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
352-383	Floating-point registers 0, 2, 4, 6
384-447	General registers 0-15
448-511	Control registers 0-15

LzLabs GmbH. Ex. 1016-314

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 in bit positions 1-31 of the word at real location 248. Bit 0 of that word is set to zero. 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 it 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.

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.

LzLabs GmbH. Ex. 1016-315

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 channel-subsystem 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



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 issuing 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.

Timing-Facility-Damage Subclass Mask

Bit 6 (TM) of control register 14 controls timing-facility-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.

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 the figures "Machine-Check-Condition Masking" and "Machine-Check Control-Register Bits."

LzLabs GmbH. Ex. 1016-316

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	TM	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 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).
- `TM` Timing-facility-damage subclass mask (bit 6 of CR14).
- `RM` Recovery subclass mask (bit 4 of CR14).
- `WM` Warning subclass mask (bit 7 of CR14).

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
Timing-facility-damage subclass mask	6	1
Warning subclass mask	7	0

Machine-Check Control-Register Bits

LzLabs GmbH. Ex. 1016-317

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 12-5

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 are contained 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.

LzLabs GmbH. Ex. 1016-319

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 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, and control registers, the prefix, and the PSW.

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 (System/370 mode or 370-XA 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 370-XA architectural mode of operation or the System/370 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. 1016-320

IML CONTROLS

The IML controls provided with some models perform initial microprogram loading (IML). The IML operation, when provided, may be used to select the 370-XA mode or the System/370 mode of operation.

When the IML operation is completed, the state of the affected CPUs, channel subsystem, 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 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 the section "Interrupt Key" in Chapter 6, "Interruptions.")

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 when the load-clear or load-normal key is activated and the corresponding operation is started. It goes off after the new PSW is loaded successfully. For details, see the section "Initial Program Loading" in Chapter 4, "Control."

LOAD-CLEAR KEY

Activating the load-clear key causes a clear-reset operation to be performed and initial program loading to be started by using the I/O device specified by the load-unit-address controls. In a multiprocessing configuration, a clear reset is propagated to all CPUs and storage units in the configuration, and a subsystem reset is performed on the remainder of the configuration. For details, see the sections "Resets" and "Initial Program Loading" in Chapter 4, "Control."

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 an initial-CPU-reset and a subsystem-reset operation to be performed and initial program loading to be started using the I/O device specified by the load-unit-address controls. In a multiprocessing configuration, a CPU reset is propagated to all other CPUs in the configuration. For details, see the sections "Resets" and "Initial Program Loading" in Chapter 4, "Control."

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 select four hexadecimal digits, which provide the device number used for initial program loading. For details, see the section "Initial Program Loading" in Chapter 4, "Control."

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.

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 the section

LzLabs GmbH. Ex. 1016-321

"Resets" in Chapter 4, "Control"). 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. For details, see the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control."

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 the section "Restart Interruption" in Chapter 6, "Interruptions.")

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 the CPU is in the load state.

START KEY

Activating the start key causes the CPU to perform the start function. (See the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control.")

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 the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control.")

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 the section "Store Status" in Chapter 4, "Control.")

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:

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.

LzLabs GmbH. Ex. 1016-322

SYSTEM-RESET-CLEAR KEY

Activating the system-reset-clear key causes a clear-reset operation to be performed. In a multiprocessing configuration, a clear reset is propagated to all CPUs and storage units in the configuration, and a subsystem reset is performed on the remainder of the configuration. For details, see the section "Resets" in Chapter 4, "Control."

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.

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.

The abnormal setting of a manual control causes the test indicator of the affected CPU to be turned on.

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.

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 indica-

LzLabs GmbH. Ex. 1016-323

tors may be switched to more than one CPU.

There need not be more than one 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. 1016-324

CHAPTER 13. I/O OVERVIEW

Comparison with System/370 13-1
The Channel Subsystem 13-2
  Subchannels 13-2
Attachment of Input/Output Devices 13-3
  Channel Paths 13-3
  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
  Addresses Dependent on Channel-Path Type 13-5
I/O Operations 13-6
  Start-Function Initiation 13-6
  Path Management 13-6
  Channel-Program Execution 13-7
  Conclusion of I/O Operations 13-7
  I/O Interruptions 13-8

COMPARISON WITH SYSTEM/370

Readers familiar with System/370 will find that the greatest difference between systems operating in the System/370 mode and systems operating in the 370-XA mode is in the input/output (I/O) facilities provided. "Input" and "output" are terms used to describe the transfer of information between I/O devices and main storage. An operation involving this kind of transfer is referred to as an input/output (I/O) operation. In the 370-XA mode, the I/O facilities are collectively called the channel subsystem. 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 the 370-XA mode by a new set of I/O instructions, by logical device addressing, and by device-accessing mechanisms that together provide more function, flexibility, and extensibility. Compatibility with System/370 has been maintained in two areas: (1) CCWs, IDAWs, and channel programs, and (2) the physical attachment of control units and I/O devices to the system.

In System/370, with some exceptions, each channel has a single physical path and data-transfer mechanism between the channel and its attached control units, and the path and channel have often been thought of as one. In the 370-XA mode, because the architecture permits up to 256 channel paths to be supported by the channel subsystem, the term "channel path" is specifically used whenever referring to the physical path between the channel subsystem and one or more control units. In most cases, the 370-XA term "channel path" is synonymous with the System/370 term "channel" when "channel" is used to mean the physical path for attachment of control units to the system.

In System/370, a channel has (1) a unique address within its channel set and (2) logically separate and distinct facilities for communicating with its attached I/O devices and with the CPU to which it is connected. For example, when an I/O device is attached to more than one channel, each channel has a separate subchannel that can be used to communicate with the I/O device. Subchannels are never shared among channels, and each subchannel is associated with only one channel path.

In the 370-XA mode, however, a single channel subsystem having a single set of subchannels is provided. 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. Therefore, the channel subsystem has both the attributes of a single channel -- a unique address (since there is only one, addressing is implicit) and a single set of subchannels for all its attached devices -- and the attributes of multiple channels, since it provides for up to 256 channel paths and for up to 64K devices.

Although the logical structures of the I/O facilities provided by the two modes differ, channel programs that can be executed by System/370 channels can be executed by the channel subsystem.

LzLabs GmbH. Ex. 1016-325

Control units that are designed to attach to System/370 channels via the IBM I/O interface can attach to the channel subsystem via the same I/O interface. This interface is described in the System Library publication IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information, GA22-6974.

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 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 addressability is 64K.

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 the 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 addressability is 256.

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.

SUBCHANNELS

The channel-subsystem facilities required for sustaining a single I/O operation are termed a subchannel. The subchannel contains information in the form of a CCW address, channel-path identifier, device number, count, identification of functions pending or in execution, status indications, interruption-subclass code, and path-availability information. I/O operations are initiated with a device by I/O instructions that specify the subchannel associated with the device.

The logical appearance of a device to the program is as the subchannel for that device. Each device has one subchannel per channel subsystem by which the device is accessible. Each device is assigned to a subchannel during an installation procedure. The device may be a physically identifiable unit, or it may be housed internal to a control unit. For example, in certain models of the IBM 3380 Direct-Access Storage, each actuator used in retrieving the 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.

In some models, subchannels are provided in blocks. In 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 the operation with the subchannel has been requested by executing START

LzLabs GmbH. Ex. 1016-326

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.

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 subsystem 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 refer to the same main-storage area.

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 via more than one channel path. Similarly, the I/O device may be accessible by the channel subsystem via more than one control unit, each having one or more channel paths to the channel subsystem.

Devices that are attached to the channel subsystem via multiple channel paths may be accessed by the channel subsystem by using any of the available channel paths. Similarly, devices having the dynamic-reconnection facility and operating in multipath mode may choose any channel path to which they are attached when logically reconnecting to the channel subsystem to continue a chain of I/O operations.

The definition of the type of channel path used by the channel subsystem is given in the System Library publication IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI, GA22-6974.

An I/O operation occurs on a channel path in one of two modes, depending on the facilities provided by the channel path and the I/O device. The modes are burst and byte-multiplex.

In 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 which monopolizes the channel path. The facilities of the channel path capable of operating in 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 exceed the prescribed limit.

In 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 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 burst mode, and slower devices run in byte-multiplex mode. Some control units have a manual switch for setting the desired mode of operation.

For improved performance, some channel paths and control units are provided

LzLabs GmbH. Ex. 1016-327

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 Original Equipment Manufacturers' Information, GA22-6974, for a description of those two facilities.

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 execution 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 execution 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 execution 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 card readers, card punches, 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.

The term "I/O device," as it is used in this publication, refers to an entity with which the channel subsystem can directly communicate. For example, the IBM 2540 Card Reader-Punch is considered to be two separate I/O devices from the point of view of the channel subsystem since the reader portion and the punch portion are individually accessible.

Most types of I/O devices, such as printers, card equipment, 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, the IBM 3705 Communications Controller handles 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 units and by switching control units among channel paths. Such switching equipment provides multiple paths by

LzLabs GmbH. Ex. 1016-328

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 paths to devices is under the control of the channel subsystem and the device, but the 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

A channel-path identifier (CHPID) is a system-unique eight-bit value assigned to each installed channel path of the system. The CHPID identifies the physical channel path. The CHPID is specified by the operand of RESET CHANNEL PATH to select which physical channel path is to be reset. The channel paths by which an I/O device is accessible are indicated in the subchannel-information block (SCHIB) when STORE SUBCHANNEL is executed. The CHPID is also used in operator messages when it is necessary to identify a particular channel path. The maximum number of channel paths and the assignment of CHPIDs to channel paths depends on the system model. A system model may provide as many as 256 channel paths.

SUBCHANNEL NUMBER

A subchannel number is a system-unique 16-bit binary number used to address a subchannel. The subchannel is addressed by seven I/O instructions: CLEAR SUBCHANNEL, HALT SUBCHANNEL, MODIFY SUBCHANNEL, RESUME SUBCHANNEL, START SUBCHANNEL, STORE SUBCHANNEL, and TEST SUBCHANNEL. Each I/O device accessible to the channel subsystem is assigned a dedicated subchannel at installation time. All I/O functions relative to a specific I/O device are specified by the program by addressing the 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 64K 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 system-unique I/O-device identifier called the device number. The device number is a 16-bit binary number that is assigned to a field of the subchannel at the time the device is assigned to the subchannel.

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 takes place between the system and the system operator. For example, the device number is entered by the system operator to specify the input device to be used for initial program loading.

ADDRESSES DEPENDENT ON CHANNEL-PATH TYPE

In addition to the use of the subchannel number and the device number, one or more different sets of addresses, not apparent to the program, may be used by the channel subsystem to communicate with I/O devices. The type of addressing used depends on the specific channel-path type and the protocols provided.

The channel-path type used by the channel subsystem is described in the System Library publication IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI, GA22-6974. For this type of channel path, an I/O device is addressed by the channel subsystem by using a device address consisting of an eight-bit binary number.

The device address identifies the particular I/O device and 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 used as a device address for the respective channel path.

For further information about I/O-device addresses used with the IBM I/O interface, see the publication referred to above.

LzLabs GmbH. Ex. 1016-329

Programming Note

The device number is assigned at device-installation time and may have any value so long as it is system-unique. Device numbers may be assigned installation-unique values in multicomputer installations in order to avoid ambiguity, particularly where a device can be switched between two or more systems.

In installations in which a system may be operated sometimes in the System/370 mode and sometimes in the 370-XA mode, it is advisable to make the 370-XA device number and System/370 device address equivalent to prevent operational problems in such mixed environments.

Additionally, the user must observe any restrictions on device-number assignment required by the control program, support programs, or specified control units or I/O devices. Any restrictions are described in the System Library publication for the program, the I/O device, the control unit, or the system model, as appropriate.

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 input/output operations from other programs that manage or process the input/output data. When START SUBCHANNEL is executed by a CPU, 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 in the subchannel, including the information passed during the execution of the START SUBCHANNEL instruction, to find an accessible path to the device. Once the device has been selected, execution of an I/O operation is accomplished by the decoding and executing 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. Orders are decoded and executed by I/O devices. The control information specifying an order may appear in the modifier bits of a CCW control-command code, 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 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 (System/370-compatible) 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, the channel subsystem next attempts to select the device via an available channel path, as indicated in the subchannel. A control unit that recognizes the device identifier connects itself logically to the channel path and responds to its selection. The channel subsystem subsequently sends the command-code part of the CCW over 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.

LzLabs GmbH. Ex. 1016-330

If the attempted selection does not occur either as a result of a busy indication or no response, 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 no response was received on one or more of the selected channel paths, 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, no response having been received on all of the selected channel paths.

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 one storage area, designated by a single CCW, or to a number of noncontiguous storage areas. In the latter case, generally a list of CCWs is used for execution of the I/O operation, 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 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 2K-byte block of storage is reached, the list is referenced to determine the next block of storage to be used. CCW indirect data addressing permits essentially the same CCW sequences to be used for a program running with dynamic address translation active in a CPU as would be used if the CPU were operating with equivalent contiguous real storage. CCW indirect data addressing permits the program to specify 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.

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 execute 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.

LzLabs GmbH. Ex. 1016-331

At the time the primary-interruption condition is generated, the channel subsystem identifies to the program, via an interruption request, that a subchannel contains information describing the conclusion of an I/O operation at the subchannel. The information, in the form of a subchannel-status word (SCSW), 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 secondary-interruption condition. The SCSW is retrieved by the instruction TEST SUBCHANNEL.

Facilities are provided for the program to initiate execution of a chain of I/O operations with a single START SUBCHANNEL. When the current CCW specifies command chaining and no unusual conditions have been detected in the operation, the receipt of the device-end signal causes the channel subsystem to fetch a new CCW and, if the suspend flag is not specified in the new CCW, to initiate execution of a new command at the device. If the suspend flag is specified, 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 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, the ending signals cause suppression of command chaining and termination of the channel 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 by specifying the suspend (S) flag in the first CCW or in a CCW fetched during command chaining.

Suspension occurs when the channel subsystem fetches a CCW with a valid S flag. 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 not specified 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 HALT SUBCHANNEL or CLEAR SUBCHANNEL. 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. The program is notified of the termination by HALT SUBCHANNEL via an I/O-interruption request generated when the I/O device presents status for the terminated operation, or, if 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 I/O-interruption request is generated by the channel subsystem after the device has been signaled. In the latter case, the device-status field of the SCSW contains 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 to notify the program of the completion of the clear function.

I/O INTERRUPTIONS

Conditions that initiate I/O interruptions are asynchronous to activity in CPUs, and more than one condition can occur at the same time. The conditions are preserved in the subchannels until accepted by a CPU, cleared by a 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 in the subchannel, an I/O-interruption request is made pending for the I/O-interruption subclass specified in the subchannel. The 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 subclass. Each CPU has eight mask bits in control register 6 which control the enabling of that CPU for each of the eight I/O-interruption subclasses, with the I/O mask (bit 6) in the PSW the master I/O-interruption mask for the CPU.

When a CPU performs an I/O interruption, 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

LzLabs GmbH. Ex. 1016-332

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.

A pending I/O interruption may also be cleared by TEST PENDING INTERRUPTION when the corresponding subclass is enabled but the PSW has I/O interruptions disabled or TEST SUBCHANNEL when the CPU is disabled for I/O interruptions from the corresponding subclass.

Regardless of how the pending interruption request is cleared, the program executes TEST SUBCHANNEL to obtain information concerning the execution of the operation and to clear the interruption condition from the subchannel.

In general, 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 from in 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 in the subchannel. In the case of a suspended subchannel, the device again becomes busy when execution of the suspended channel program is resumed.

LzLabs GmbH. Ex. 1016-333

Introduction 14-1
I/O-Instruction Formats 14-1
I/O-Instruction Execution 14-1
  Serialization 14-1
  Operand Access 14-2
  Condition Code 14-2
  Program Exceptions 14-2
Instructions 14-2
  CLEAR SUBCHANNEL 14-3
  HALT SUBCHANNEL 14-4
  MODIFY SUBCHANNEL 14-6
  RESET CHANNEL PATH 14-6
  RESUME SUBCHANNEL 14-8
  SET ADDRESS LIMIT 14-9
  SET CHANNEL MONITOR 14-10
  START SUBCHANNEL 14-11
  STORE CHANNEL PATH STATUS 14-13
  STORE CHANNEL REPORT WORD 14-13
  STORE SUBCHANNEL 14-14
  TEST PENDING INTERRUPTION 14-15
  TEST SUBCHANNEL 14-16

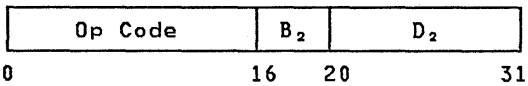
INTRODUCTION

The I/O instructions include all instructions that are provided for the control of channel-subsystem operations. The 13 I/O instructions are listed in the figure "Summary of I/O Instructions." 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 called "Associated Functions," which summarizes the asynchronous functions.

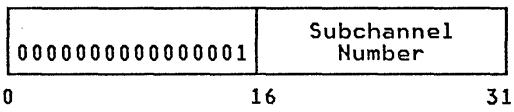
I/O-INSTRUCTION FORMATS

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. The figure "Summary of I/O-Instruction Characteristics" 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 subsystem-identification word has the following format:



Bits 16-31 form the binary number of the subchannel to be used for the function specified by the instruction.

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 the section "CPU Serialization" in Chapter 5, "Program Execution."

LzLabs GmbH. Ex. 1016-335

OPERAND ACCESS

During 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 as to whether fetch accesses are made to fields of an operand or the subchannel (as applicable) when those fields are not needed to complete execution of the I/O instruction. (See the section "Relation Between Operand Accesses" in Chapter 5, "Program Execution.")

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 designated in the figure "Summary of I/O Instruction Characteristics." 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 the section "Deferred-Condition-Code Contents" in Chapter 16, "I/O Interruptions," 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 target subchannel or channel-subsystem facility was busy with a previously initiated function.

Condition Code 3: Instruction execution was ineffective because the target 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 condition code having precedence 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, privileged-operation, and specification exceptions. The figure "Summary of I/O-Instruction Characteristics" shows the exceptions that are applicable to each of the I/O instructions. The execution of the instruction is suppressed for privileged-operation, operand, and specification exceptions. Except as indicated otherwise in the section "Special Conditions" for each instruction, the ending for access exceptions is as described in the section "Recognition of Access Exceptions" in Chapter 6 "Interruptions."

INSTRUCTIONS

The mnemonics, format, and operation codes of the I/O instructions are given in the figure "Summary of I/O Instruction Characteristics." 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 assembly language are shown with each instruction. In the case of START SUBCHANNEL, for example, SSCH is the mnemonic and $D_2(B_2)$ the operand designation.

LzLabs GmbH. Ex. 1016-336

Name	Mnemonic	Characteristics										Op Code
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			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			B233
STORE CHANNEL PATH STATUS	STCPS	S		P	A	SP		¢			ST	B23A
STORE CHANNEL REPORT WORD	STCRW	S	C	P	A	SP		¢			ST	B239
STORE SUBCHANNEL	STSCH	S	C	P	A	SP	OP	¢	GS		ST	B234
TEST PENDING INTERRUPTION	TPI	S	C	P	A[1]	SP		¢			ST	B236
TEST SUBCHANNEL	TSCH	S	C	P	A	SP	OP	¢	GS		ST	B235

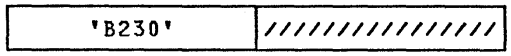
Explanation:

¢ 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.
C Condition code is set.
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. General register 1 is used as a parameter, and general register 2 may be used as a parameter depending on the contents of general register 1.
GS Instruction execution includes the implied use of general register 1 as the SID.
OP Operand exception.
P Privileged-operation exception.
S S instruction format.
SP Specification exception.
ST PER storage-alteration event.

Summary of I/O Instructions

CLEAR SUBCHANNEL

CSCH [S]



The specified subchannel is cleared, the current start or halt function, if any, is terminated at the specified subchannel, and the channel subsystem is signaled to asynchronously perform the clear function at the specified subchannel and at the associated device.

General register 1 contains the subsystem-identification word, which specifies the subchannel that is to be cleared.

If conditions allow, the following functions are performed:

If a start or halt function is in progress, it is terminated at the subchannel.

The status-pending bit of the status-control field is set to zero in the specified subchannel.

The activity-control (AC) field of the specified subchannel is set to zeros, except for the clear-pending bit, which is set to one.

The function-control (FC) field of the specified subchannel is set to zeros, except for the clear-function bit, which is set to one.

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 the section "Clear-Function Execution" in Chapter 15, "Basic I/O Functions."

Condition code 0 is set to indicate that the actions described above have been taken.

LzLabs GmbH. Ex. 1016-337

Associated Functions

Subsequent to the execution of CLEAR SUBCHANNEL, the channel subsystem asynchronously performs the clear function. If conditions allow, the channel subsystem selects 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 being made status-pending (see the section "Clear-Function Execution" in Chapter 15, "Basic I/O Functions").

When the subchannel becomes status-pending as a result of executing 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 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 the section "Interruption-Response Block" in Chapter 16, "I/O Interruptions.")

Measurement parameters are not accumulated and the 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 that are listed below. An operand exception is recognized when bit positions 0-15 of general register 1 do not contain 0001 hex.

Resulting Condition Code:

0	Clear function initiated
1	--
2	--
3	Not operational

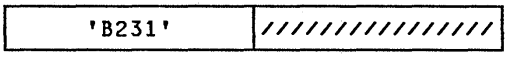
Program Exceptions:

Operand

Privileged operation

HALT SUBCHANNEL

HSCH [S]



The current start function, if any, is terminated at the specified subchannel, and the channel subsystem is signaled to asynchronously perform the halt function at the specified subchannel and at the associated device.

General register 1 contains the subsystem-identification word, which specifies the subchannel that is to be halted.

If conditions allow, the following functions are performed:

If a start function is in progress, it is terminated at the subchannel.

The halt-pending and halt-function bits of the subchannel are set to ones.

When HALT SUBCHANNEL is issued to a subchannel that is subchannel-and-device-active and intermediate-status-pending, the status-pending indication is eliminated (see the discussion of bits 24, 25, and 28 in the section "Activity Control" in Chapter 16, "I/O Interruptions"). 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 the section "Halt-Function Execution" in Chapter 15, "Basic I/O Functions."

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 selects 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.

LzLabs GmbH. Ex. 1016-338

When the subchannel becomes status-pending as a result of executing 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 the section "Interruption-Response Block" in Chapter 16, "I/O Interruptions," and the section "Halt-Function Execution" in Chapter 15, "Basic I/O Functions.")

In some models, path availability is tested as part of the halt function (rather than 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.

When a status-pending condition is eliminated during execution of HALT SUBCHANNEL, this condition is reestablished along with the other status conditions when completion of the halt-function execution 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 while the subchannel was start-pending or when primary status is presented during initiation of the start function. In either case, 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. When the halt-pending bit is stored as one, the halt signal was not issued to the device.

If measurement parameters are being accrued when a start function is terminated by HALT SUBCHANNEL, the measurement parameters continue to be accrued for the subchannel and reflect the extent of subchannel and device usage, if any, required in executing the currently terminated start function. The measurement parameters, if any, are 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 status. (For a description of channel-subsystem monitoring, see the section "Channel-Subsystem Monitoring Facilities" in Chapter 17, "I/O Support Functions.")

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 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. In some models, a subchannel is also not operational for HALT SUBCHANNEL when no channel paths are available for selection of the device. (See the section "Channel-Path Availability" in Chapter 15, "Basic I/O Functions," for a description of channel paths that are available for selection.)

HALT SUBCHANNEL can encounter the program exceptions listed below. An operand exception is recognized when bit positions 0-15 of general register 1 do not contain 0001 hex.

Resulting Condition Code:

0 Halt function initiated
1 Status other than intermediate status pending
2 Busy
3 Not operational

Program Exceptions:

Operand
Privileged operation

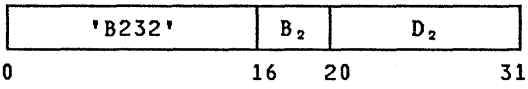
Programming Note

After execution of HALT SUBCHANNEL, the status-pending condition indicating the completion of execution of the halt function may be delayed for an extended period of time, for example, if the device was a magnetic-tape unit executing a rewind command.

LzLabs GmbH. Ex. 1016-339

MODIFY SUBCHANNEL

MSCH $D_2(B_2)$ [S]



The program-modifiable fields of the specified subchannel-information block (SCHIB) are placed in the specified 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 the subsystem-identification word, which specifies the subchannel that is to receive the contents of the program-modifiable fields of the SCHIB. The second-operand address is the logical address of the SCHIB and is designated on a word boundary.

The channel-subsystem operations that are affected by the program-modifiable fields of the SCHIB are (1) the use of the subchannel for I/O processing (E field), (2) interruption processing (interruption parameter and ISC field), (3) path management (D, LPM, and POM fields), and (4) the use of the monitoring and address-limit-checking facilities (measurement-block index and LM and MM fields). Bits 0-1 and 5-7 of word 1 and bits 0-31 of word 6 of the SCHIB operand must be specified as zeros, and bits 9-10 of word 1 must not both be ones. The remaining fields of the SCHIB that are designated by the second-operand address are ignored and do not affect the processing of MODIFY SUBCHANNEL. (See the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions.")

Condition code 0 is set to indicate that the program-modifiable fields of the SCHIB have been placed in the subchannel.

Special Conditions

Condition code 1 is set and no other action is taken when the subchannel is status-pending. (See the section "Status Control" in Chapter 16, "I/O Interruptions.")

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 the section "Function Control" in Chapter 16, "I/O Interruptions.")

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 listed below. The execution of MODIFY SUBCHANNEL is suppressed on all addressing and protection exceptions. An operand exception is recognized when bit positions 0-15 of general register 1 do not contain 0001 hex. An operand exception is also recognized when bits 0-1 and 5-7 of word 1 and bits 0-31 of word 6 of the SCHIB operand are not specified as zeros or when bits 9-10 of word 1 are both specified as ones. A specification exception is recognized when the second operand is not designated on a word boundary.

Resulting Condition Code:

0 SCHIB information placed in the subchannel
1 Subchannel status-pending
2 Subchannel busy
3 Not operational

Program Exceptions:

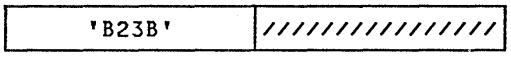
Access (fetch, operand 2)
Operand
Privileged operation
Specification

Programming Note

While a subchannel is disabled, unsolicited status that is presented by the associated device is discarded by the channel subsystem. 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.

RESET CHANNEL PATH

RCHP [S]

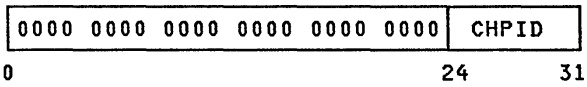


The channel-path-reset facility is signaled to perform the channel-path-reset function on the specified channel path.

LzLabs GmbH. Ex. 1016-340

General register 1 contains, in bit positions 24-31, the channel-path identifier (CHPID) of the channel path on which the channel-path-reset function is to be performed. Bit positions 0-23 of general register 1 are reserved and must contain zeros; otherwise, an operand exception is recognized.

GR1



If conditions allow, the channel-path-reset facility is signaled to asynchronously perform the channel-path-reset function on the specified path. The channel-path-reset function is summarized below in the section "Associated Functions" and is described in detail in the section "Channel-Path Reset" in Chapter 17, "I/O Support Functions."

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 in all subchannels that have access to the specified 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 multipath mode may be able to continue to execute on other channel paths in the multipath group, if any. (See the section "Channel-Path-Reset-Function Execution" in Chapter 15, "Basic I/O Functions.")

The result of performing the channel-path-reset function on the designated channel path is communicated to the program via a channel report (see the section "Channel Report" in Chapter 17, "I/O Support Functions"). If the specified channel path is physically available when condition code 0 is set for RESET CHANNEL PATH, the error-recovery code of the first CRW of the channel report can indicate initialized, permanent error with facility initialized, or permanent error with facility not initialized. In those models that set condition code 0 for RESET CHANNEL PATH when the specified channel path is not physically available, the error-recovery code of the first CRW of the channel report can only indicate permanent error with facility initialized or permanent error with facility not initialized. When either of the two types of permanent error is indicated, the designated channel path is in the check-stop state, and the corresponding PAM bit is zero. In all cases, since the report is made pending as a direct result of action that is taken by the program, the first CRW of the channel report is specified as being solicited.

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 RESET CHANNEL PATH.

Condition code 3 is set and no other action is taken when, on some models, the designated channel path is not operational for RESET CHANNEL PATH. On these models, the channel path is not operational for 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 listed below. An operand exception is recognized when bit positions 0-23 of general register 1 do not contain zeros.

Resulting Condition Code:

0	Channel-path-reset function initiated
1	--
2	Busy
3	Not operational

Program Exceptions:

Operand
Privileged operation

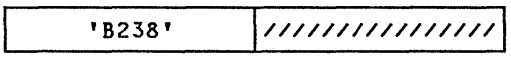
Programming Note

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 via the channel path to which RESET CHANNEL PATH is to be issued must be terminated by HALT SUBCHANNEL or CLEAR SUBCHANNEL. Otherwise, an unsolicited device end presented by the device subsequent to receiving the reset signal could possibly be interpreted by the channel subsystem as a solicited device end and cause command chaining to occur.

LzLabs GmbH. Ex. 1016-341

RESUME SUBCHANNEL

RSCH [S]



The channel subsystem is signaled to perform the resume function on the specified subchannel.

General register 1 contains the subsystem-identification word, which specifies the subchannel on which the resume function is to be performed.

If conditions allow, the following functions are performed:

The resume-pending bit of the subchannel is set to one.

Logically prior to the setting of condition code 0 and only if the subchannel is currently in the suspended state, the path-not-operational mask (PNOM) of the subchannel is set to zeros.

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 the section "Start-Function and Resume-Function Execution" in Chapter 15, "Basic I/O Functions."

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 executes the resume function. Execution of the resume function causes the execution of a currently suspended channel-program to be resumed with the associated device if, for the current CCW, the suspend flag causing suspension has been set to zero by the program. If the suspend flag remains set to one, channel-program execution remains suspended. (See the section "Start-Function and Resume-Function Execution" in Chapter 15, "Basic I/O Functions.")

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 listed below. An operand exception is recognized when bit positions 0-15 of general register 1 do not contain 0001 hex.

Resulting Condition Code:

0 Resume function initiated
1 Subchannel status-pending
2 Resume function not applicable
3 Not operational

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 suspend-and-resume facility is used with devices that require certain commands to be first or to 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 subsequent to the resumption of channel-program execution. One way the program can ensure proper sequencing of commands at the device is by allowing the optional subchannel-suspended I/O interruption.

   The use of the PCI flag in the CCW containing the S flag for notification that the subchannel is suspended is not a reliable method of notification because the PCI I/O interruption may occur before the subchannel has been suspended. The SCSW would indicate that an I/O operation is in progress at the subchannel and device in this case.

LzLabs GmbH. Ex. 1016-342

To ensure consistent results, the suspend flag of the target CCW should be set to zero before executing RESUME SUBCHANNEL. This ensures that a CCW that previously had the suspend flag set to one does not cause the channel program to remain suspended in spite of the RESUME SUBCHANNEL. If RESUME SUBCHANNEL is executed prior to setting the CCW suspend flag to zero, it is possible that the channel subsystem will recognize the resume-pending condition, refetch the CCW with the suspend flag still one, and, accordingly, reset the resume-pending condition and suspend the execution of the channel program, all before the CCW suspend flag is set to zero by the program.

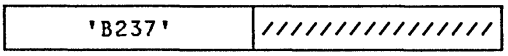
2. Some models recognize a resume-pending condition only after a CCW having a valid S flag set to one is fetched. Therefore, if a subchannel is resume-pending and, during execution of the channel program, no CCW is fetched having a valid S flag set to one, the resume-pending bit remains one until the 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 execution of the resume function.

4. The contents of the CCW fetched during execution of the resume function may be different from the contents of the same CCW when it was previously fetched and contained a valid S flag.

SET ADDRESS LIMIT

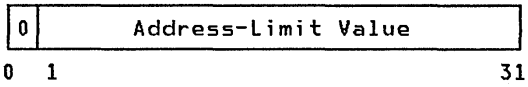
SAL [S]



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.

General register 1 contains the address to be used as the address-limit value. The address is designated on a 64K-byte boundary, and the leftmost bit of general register 1 is zero.

GR1



Associated Functions

The address limit that is recognized by the address-limit-checking facility on a data access to main storage is called the address-limit value. The initial address-limit value (immediately following the system IPL procedure) is zero. This address is used by the address-limit-checking facility until the facility recognizes a signal (caused by SET ADDRESS LIMIT) to use a specified address. The setting of the address-limit value to the address (referred to as the specified address) that is passed by SET ADDRESS LIMIT occurs asynchronously with respect to the execution of the instruction. The effect of the specified address on a data access during channel-program execution depends on whether SET ADDRESS LIMIT was executed prior to, during, or subsequent to the execution of the START SUBCHANNEL that initiated I/O processing for that subchannel. If SET ADDRESS LIMIT is executed prior to START SUBCHANNEL, the access to storage is under control of the specified address. If SET ADDRESS LIMIT is executed during or subsequent to START SUBCHANNEL, control over the access of storage at a specified address is unpredictable for that channel-program execution.

The address-limit value is used by the address-limit-checking facility in a comparison with the absolute storage address each time a data access to main storage is attempted. The result of the address comparison is used to determine whether an address-limit violation exists when the subchannel for which the data access is being made designates address-limit checking. (See the section "Program Check" in Chapter 16, "I/O Interruptions.") A subchannel designates address-limit checking when the address-limit-checking-control bit passed in the operation-request block (ORB) is set to one and the limit-mode bits of the subchannel-information block (SCHIB) are set to some value other than all zeros. (See the sections "Subchannel-Information Block" and "Operation-Request Block" in Chapter 15, "Basic I/O Functions.")

Special Conditions

SET ADDRESS LIMIT can encounter the program exceptions listed below. An

LzLabs GmbH. Ex. 1016-343

operand exception is recognized when the address in general register 1 is not designated on a 64K-byte boundary or when the leftmost bit is not zero.

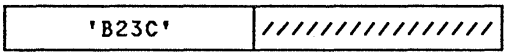
Condition Code: The code remains unchanged.

Program Exceptions:

Operand
Privileged operation

SET CHANNEL MONITOR

SCHM [S]

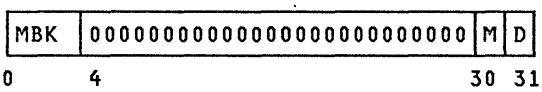


The monitoring modes of the channel subsystem are made either active or inactive, depending on the setting of the measurement-mode-control bits in general register 1. Depending on the setting of the measurement-mode-control bit for the measurement-block-update mode, the channel subsystem is signaled to make the mode active, or the mode is made inactive.

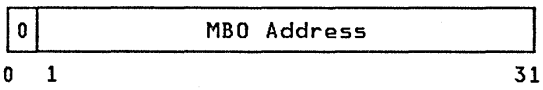
Depending on the setting of the measurement-mode-control bit for the device-connect-time-measurement mode, the mode is made active or inactive.

The format of general registers 1 and 2 is as follows:

GR1



GR2



Bit positions 0-3 of general register 1 contain the measurement-block key (MBK). When the measurement-mode-control bit (M) is one, MBK specifies the access key that is to be used by the channel subsystem when it accesses the measurement-block area. Otherwise, bit positions 0-3 of general register 1 are ignored.

Bit positions 4-29 of general register 1 contain zeros.

Bit 30 (M) of general register 1 is the measurement-mode-control bit that controls the measurement-block-update mode.

When the M bit is one and conditions allow, the measurement-block-update facility is signaled to asynchronously make the measurement-block-update mode active. In addition, the MBO address (in general register 2) and the measurement-block key (MBK) (in general register 1) are passed to the measurement-block-update facility. 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 the section "Channel-Subsystem-Monitoring Facilities" in Chapter 17, "I/O Support Functions."

When the M bit 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.

Bit 31 (D) of general register 1 is the measurement-mode-control bit that controls the device-connect-time-measurement mode.

When the D bit 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 the D bit 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.

Bit position 0 of general register 2 must contain a zero when the M bit is one. Otherwise, bit position 0 of general register 2 is ignored.

Bit positions 1-31 of general register 2 contain the absolute address of the measurement-block origin (MBO). When the M bit is one, the MBO address specifies the beginning address of the measurement-block area and is designated on a 32-byte boundary. The MBO address is to be used by the channel subsystem to locate measurement blocks. When the M bit is zero, the contents of general register 2 are ignored.

If the channel-subsystem timer that is used by the measurement facilities is in the error state, the state is reset. This happens, independent of the setting of the two measurement-mode-control bits. (See the section "Channel-Subsystem-Timing Facility" in Chapter 17, "I/O Support Functions.")

LzLabs GmbH. Ex. 1016-344

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 operations that are performed by the facility depend upon whether or not the mode is already active when the signal is generated.

If the measurement-block-update mode is not active 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 and MBO that were passed when the signal was generated are used to control the storing of measurement data.

If the measurement-block-update mode is active when the signal is generated, the mode remains active, and the MBK and MBO that were passed by a previous SET CHANNEL MONITOR 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 that were passed when the signal was generated are used instead of the MBK and MBO that were passed by a previous SET CHANNEL MONITOR.

In either of the above cases, the measurement-block-update facility recognizes the signal during, or subsequent to, the SET CHANNEL MONITOR that caused the signal to be generated and logically prior to the execution of a start function initiated by a subsequent START SUBCHANNEL for any subchannel that is enabled for measurement by this facility. If a subchannel that is enabled for measurement by this facility has a start function in progress when the signal is generated, it is unpredictable when measurement data for that subchannel is stored with the use of the MBK and MBO that were passed when the signal was generated.

While the measurement-block-update mode is active, measurements of resource utilization are accumulated for subchannels that are enabled for measuring by the measurement-block-update facility. Measurements for a subchannel are accumulated in a single 32-byte measurement block within the measurement-block area. A subchannel is enabled for measurement by the measurement-block-update facility when the measurement-block-update-enable bit is set to one for that subchannel by MODIFY SUBCHANNEL. The measurement block that is used to accumulate measurements for a subchannel is determined by the measurement-block index that is contained in the subchannel.

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 measuring by the device-connect-time-measurement facility. Measurements for a subchannel are provided in the ESW of the IRB. A subchannel is enabled for measurement by the device-connect-time-measurement facility when the device-connect-time-measurement-enable bit is set to one for that subchannel by MODIFY 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 the section "Channel-Subsystem-Monitoring Facilities" in Chapter 17, "I/O Support Functions."

Special Conditions

SET CHANNEL MONITOR can encounter the program exceptions listed below. An operand exception is recognized when bits 4-29 of general register 1 are not all zeros. When bit 30 of general register 1 is one, an operand exception is also recognized if bit 0 of general register 2 is not zero or if the MBO address in general register 2 is not specified on a 32-byte boundary.

Condition Code: The code remains unchanged.

Program Exceptions:

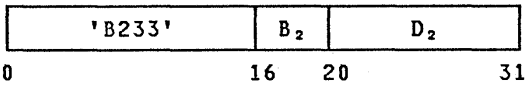
Operand
Privileged operation

Programming Note

When the channel subsystem is initialized, the measurement-block-update and device-connect-time modes are made inactive.

START SUBCHANNEL

SSCH $D_2(B_2)$ [S]



The channel subsystem is signaled to asynchronously perform the start func-

LzLabs GmbH. Ex. 1016-345

tion for the associated device, and the execution parameters that are contained in the specified ORB are placed in the specified subchannel. (See the section "Operation-Request Block" in Chapter 15, "Basic I/O Functions.")

General register 1 contains the subsystem-identification word, which specifies the subchannel that is to be started. The second-operand address is the logical address of the ORB and is designated on a word boundary.

If conditions allow, the following functions are performed:

The execution parameters contained in the ORB are placed in the subchannel.

For some models, when START SUBCHANNEL is executed and the subchannel is status-pending with only secondary status, the status-pending condition is discarded at the subchannel.

The start-pending and start-function bits of the subchannel are set to ones.

Logically prior to the setting of condition code 0, the path-not-operational mask (PNOM) of the subchannel is set to zeros.

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 the section "Start-Function and Resume-Function Execution" in Chapter 15, "Basic I/O Functions."

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. In some models, the fields of the ORB that must contain zeros are also checked asynchronously (rather than during the execution of the instruction). The detection of invalid fields causes the subchannel to become primary-, secondary-, and alert-status-pending with deferred condition code 1 and program check indicated. (See the section "Program Check" in Chapter 16, "I/O Interruptions.") 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. (See the section "Subchannel-Status Word" in Chapter 16, "I/O Interruptions.").

In some models, path availability is tested asynchronously (rather than as part of the execution of the instruction). When no channel path is available for selection, the subchannel becomes primary-and-secondary status-pending 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 TEST SUBCHANNEL.

If conditions allow, the channel subsystem selects a path and initiates the execution of the channel program that is specified in the ORB. (See the section "Start-Function and Resume-Function Execution" in Chapter 15, "Basic I/O Functions.")

Special Conditions

Condition code 1 is set and no other action is taken if the subchannel is status-pending when START SUBCHANNEL is executed. In some models, condition code 1 is not set when the subchannel is status-pending with only secondary status. These models discard the status-pending condition.

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 the section "Function Control" in Chapter 16, "I/O Interruptions").

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 assigned to it, or is not enabled.

A subchannel is also not operational for START SUBCHANNEL, in some models, when no channel path is available for selection. In these models, the lack of an available channel path is detected as part of START SUBCHANNEL execution. In other models, channel path availability is only tested as part of the asynchronous start function.

START SUBCHANNEL can encounter the program exceptions listed below. The execution of START SUBCHANNEL is suppressed on all addressing and protection exceptions. An operand exception is recognized when bit positions 0-15 of general register 1 do not contain 0001 hex. An operand exception

LzLabs GmbH. Ex. 1016-346

is also recognized in some models when bits 5-7, 13-15, and 24-31 of word 1 and bit 0 of word 2 of the ORB are not all zeros. In other models, these bits are not tested for zeros as part of START SUBCHANNEL execution but rather as part of the asynchronous start function. A specification exception is recognized when the second operand is not designated on a word boundary. The execution of START SUBCHANNEL is suppressed.

Resulting Condition Code:

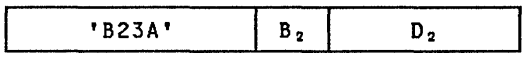
0 Start function initiated
1 Subchannel status-pending
2 Subchannel busy
3 Not operational

Program Exceptions:

Access (fetch, operand 2)
Operand
Privileged operation
Specification

STORE CHANNEL PATH STATUS

STCPS $D_2(B_2)$ [S]



A channel-path-status word of up to 256 bits is stored at the specified location.

The second-operand address is the logical address of the location where the channel-path-status word is to be stored and is designated on a 32-byte boundary.

The channel-path-status word specifies which channel paths are being used in active communication 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 specified 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 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 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-error state or are not physically available (the corresponding PAM bit is zero), the corresponding bits are stored as zeros.

Special Conditions

STORE CHANNEL PATH STATUS can encounter the program exceptions listed below. The execution of STORE CHANNEL PATH STATUS is suppressed on all addressing and protection exceptions. A specification exception is recognized when the second operand is not designated on a 32-byte boundary.

Condition Code: The code remains unchanged.

Program Exceptions:

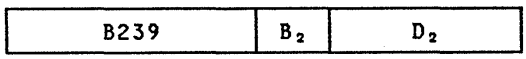
Access (store, operand 2)
Privileged operation
Specification

Programming Note

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.

STORE CHANNEL REPORT WORD

STCRW $D_2(B_2)$ [S]



A CRW containing information affecting the channel subsystem is stored at the specified location.

The second-operand address is the logical address of the location where the CRW is to be stored and is designated on a word boundary.

LzLabs GmbH. Ex. 1016-347

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 of the channel subsystem 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 the section "Channel-Subsystem Recovery" in Chapter 17, "I/O Support Functions."

When one or more channel reports are pending, the instruction causes a CRW to be stored at the specified location and condition code 0 to be set. A pending CRW can only be stored by executing 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 specified location and condition code 1 to be set.

Special Conditions

STORE CHANNEL REPORT WORD can encounter the program exceptions listed below. The execution of STORE CHANNEL REPORT WORD is suppressed on all addressing and protection exceptions. A specification exception is recognized when the second operand is not designated on a word boundary.

Resulting Condition Code:

0 CRW stored; CRW was pending
1 Zeros stored; CRW was not pending
2 --
3 --

Program Exceptions:

Access (store, operand 2)
Privileged operation
Specification

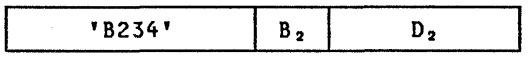
Programming Notes

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 the section "Channel-Subsystem Recovery" in Chapter 17, "I/O Support Functions," 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 specified subchannel is stored in the specified SCHIB.

General register 1 contains the subsystem-identification word, which specifies the subchannel for which the information is to be stored. The second-operand address is the logical address of the SCHIB and is designated on a word boundary.

The information that is stored in the SCHIB consists of the path-management-control word, the SCSW, and three words of model-dependent information. (See the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions.")

The execution of STORE SUBCHANNEL does not change any information contained in the subchannel.

Condition code 0 is set to indicate that control and status information for the specified subchannel has been stored in the SCHIB.

Special Conditions

Condition code 3 is set and no other action is taken when the indicated 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 listed below. An operand exception is recognized when bit positions 0-15 of general register 1 do not contain 0001 hex. A specification exception is recognized when the second operand is not designated on a word boundary.

LzLabs GmbH. Ex. 1016-348

Resulting Condition Code:

0 SCHIB stored
1 --
2 --
3 Not operational

Program Exceptions:

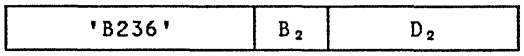
Access (store, operand 2)
Operand
Privileged operation
Specification

Programming Note

Repeated execution of STORE SUBCHANNEL, without an intervening delay to determine, for example, when a subchannel changes state, should be avoided because repeated CPU accesses to the subchannel may delay access of the subchannel by the channel subsystem in updating the subchannel.

TEST PENDING INTERRUPTION

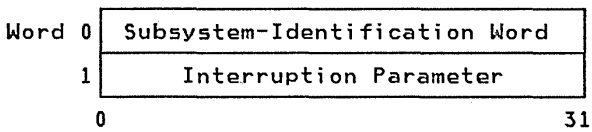
TPI $D_2(B_2)$ [S]



The interruption code for a pending subchannel interruption is stored at the specified location, and the pending interruption request is cleared.

The second-operand address, when nonzero, is the logical address of the location where the interruption code is to be stored and is designated on a word boundary.

If the second-operand address is zero, the interruption code is stored in real locations 184-191. In this case, low-address protection and key-controlled protection do not apply. If the second-operand address is nonzero, the interruption code is stored as specified by the address. The interruption code that is stored during execution of the instruction is defined as follows:



Subsystem-Identification Word: See the section "I/O-Instruction Formats" in this chapter.

Interruption Parameter: Word 1 contains a four-byte parameter which is specified by the program and which previously was passed to the subchannel in word 0 of the ORB or the path-management-control word. When a device presents alert status and the parameter was not passed previously to the subchannel by START SUBCHANNEL or MODIFY SUBCHANNEL, this field contains zeros.

Pending 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 interruption code is not stored, the second operand location is not modified, and condition code 0 is set.

If a pending interruption request is accepted, the interruption code is stored, the pending interruption request is cleared, and condition code 1 is set. The 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 interruption occurs.

Special Conditions

TEST PENDING INTERRUPTION can encounter the program exceptions listed below. The execution of TEST PENDING INTERRUPTION is suppressed on all addressing and protection exceptions. A specification exception is recognized when the second operand is not designated on a word boundary. The execution of TEST PENDING INTERRUPTION is suppressed.

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 Note

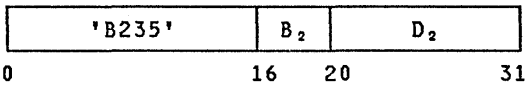
TEST PENDING INTERRUPTION with a second-operand address of zero should only be executed when I/O interruptions are masked off. Otherwise, an interruption code stored at real locations 184-191 by the instruction may be lost if an I/O interruption occurs. The

LzLabs GmbH. Ex. 1016-349

interruption code that identifies the source of the interruption is stored at real locations 184-191, replacing the code that is stored by the instruction.

TEST SUBCHANNEL

TSCH $D_2(B_2)$ [S]



Control and status information for the specified subchannel is stored in the specified IRB.

General register 1 contains the subsystem-identification word, which specifies the subchannel for which the information is to be stored. The second-operand address is the logical address of the IRB and is designated on a word boundary.

The information that is stored in the IRB consists of the SCSW, the extended-status word, and the extended-control word. (See the section "Interruption-Response Block" in Chapter 16, "I/O Interruptions.")

The subchannel is said to be status-pending for TEST SUBCHANNEL if 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 subchannel-control bits in the subchannel (as described in the figure "Subchannel-Control Bits Cleared by TEST SUBCHANNEL"). If an 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 subchannel-control bits are cleared. Condition code 1 is set to indicate that these actions have been taken.

The figure "Subchannel-Control Bits Cleared by TEST SUBCHANNEL" describes which subchannel-control bits are cleared by TEST SUBCHANNEL when the subchannel is status-pending. All other fields in the subchannel remain unchanged.

Field	Subchannel State*				
	Alert Sta Pdg	Int Sta Pdg	Pri Sta Pdg	Sec Sta Pdg	Sta Pdg Alone
Function Control	C	N	C	C	C
Activity Control	Cp	N	Cp	C	Cp
Status Control	Cs	Cs	Cs	Cs	Cs

Explanation:

* Note that the rightmost column applies to status pending when it is alone. The other four status-pending states result in the clearing actions given. These action apply both when a single status-pending state occurs and when a combination of the four status-pending states 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 bits are cleared.
Cs The status-pending bit is cleared.
N Not changed unless function control indicates the halt function. If so, fields are cleared as for status-pending alone.

Subchannel-Control Bits Cleared by TEST SUBCHANNEL

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 assigned to it, or is not enabled.

TEST SUBCHANNEL can encounter the program exceptions listed below. When the execution of TEST SUBCHANNEL is terminated on addressing and protection exceptions, the state of the subchannel is not changed. An operand exception is recognized when bit positions 0-15 of general register 1 do not contain 0001 hex. A specification exception is recognized when the second operand is not designated on a word boundary.

LzLabs GmbH. Ex. 1016-350

Resulting Condition Code:

0 IRB stored; subchannel was status-pending
1 IRB stored; subchannel was 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 in the subchannel, and typically TEST SUBCHANNEL is issued to clear the status. TEST SUBCHANNEL may also be issued 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 when no CPU in the configuration is enabled for I/O interruptions. In such a situation, the start-function completion may never be indicated. For example, if the channel subsystem is holding an interface-control-check (IFCC) condition in abeyance (for any subchannel) because I/O interruptions (for that subchannel) are disabled in all CPUs, 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 is not initiated until the CPU is enabled for the required I/O-interruption subclass, allowing the IFCC condition to be indicated in 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 CPU accesses to the subchannel may delay access of the subchannel by the channel subsystem in updating the subchannel.

LzLabs GmbH. Ex. 1016-351

Control of Basic I/O Functions 15-1
Subchannel-Information Block (SCHIB) 15-1
Path-Management-Control Word 15-2
Subchannel-Status Word 15-6
Model-Dependent Area 15-6
Summary of Modifiable Fields 15-6
Channel-Path Allegiance 15-9
Working Allegiance 15-9
Active Allegiance 15-9
Dedicated Allegiance 15-10
Channel-Path Availability 15-10
Control-Unit Type 15-11
Clear-Function Execution 15-11
Halt-Function Execution 15-13
Start-Function and Resume-Function Execution 15-15
Execution of I/O Operations 15-17
Programming Note 15-17
Blocking of Data 15-18
Operation-Request Block (ORB) 15-19
Channel-Command Word 15-20
Command Code 15-22
Designation of Storage Area 15-22
Chaining 15-23
Data Chaining 15-26
Command Chaining 15-27
Skipping 15-28
Program-Controlled Interruption 15-29
CCW Indirect Data Addressing 15-30
Suspension of Channel-Program Execution 15-31
Commands 15-32
Write 15-33
Read 15-34
Read Backward 15-34
Control 15-35
Sense 15-36
Sense ID 15-38
Transfer in Channel 15-39
Command Retry 15-39
Concluding I/O Operations During Initiation 15-39
Immediate Conclusion of I/O Operations 15-40
Concluding I/O Operations During Data Transfer 15-41
Channel-Path-Reset-Function Execution 15-42

Some I/O instructions designate to the channel subsystem that a specific function is to be performed. Collectively, these designated 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 contained in the subchannel controls how the clear, halt, resume, and start functions are executed. This information is communicated to the program in the subchannel-information block during execution of STORE SUBCHANNEL.

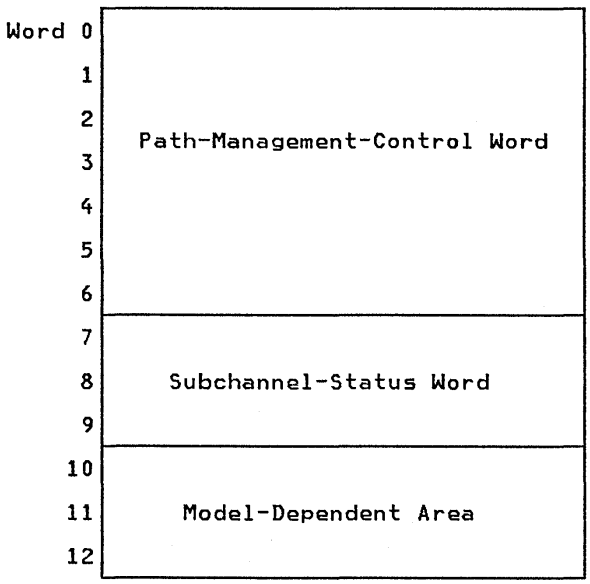
SUBCHANNEL-INFORMATION BLOCK (SCHIB)

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, specifying 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.

STORE SUBCHANNEL is used to store the current PMCW, the SCSW, and model-dependent data of the indicated subchannel. MODIFY SUBCHANNEL alters certain

LzLabs GmbH. Ex. 1016-353

PMCW fields at the subchannel. When the program needs to change the contents of one or more of the PMCW fields, the normal procedure is (1) to issue STORE SUBCHANNEL to obtain the current contents, (2) to perform the required modifications to the PMCW in main storage, and (3) to issue MODIFY SUBCHANNEL to pass the new information to the subchannel. The SCHIB has the following format:



Path-Management-Control Word

The path-management-control word (PMCW) has the format shown in the figure "PMCW Format" when the subchannel is valid (see the discussion of device-number-valid bit later in this section).

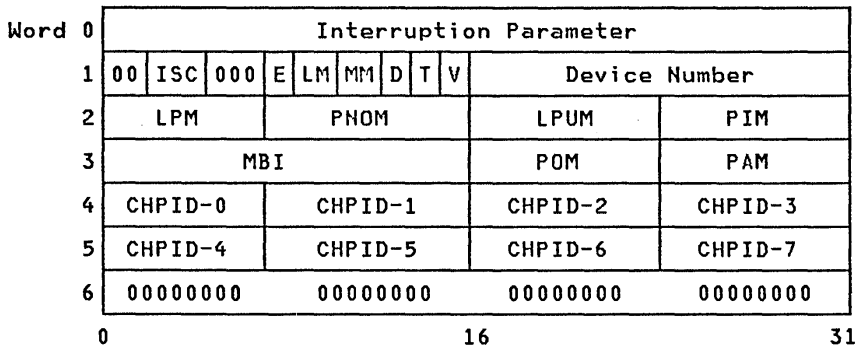
Interruption Parameter: Bits 0-31 of word 0 contain the interruption parameter that is stored as word 1 of the interruption code. The interruption parameter can be set to any value by START SUBCHANNEL and MODIFY SUBCHANNEL. The initial value of the interruption parameter is zero.

Interruption-Subclass Code (ISC): Bits 2-4 of word 1 contain a binary number (0-7) specifying the bit number of the 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 the section "Priority of Interruptions" in Chapter 16, "I/O Interruptions"). 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. They must be zeros when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized.

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 issued to the subchannel cause condition code 3 to be set. The E bit can be set to any value by MODIFY SUBCHANNEL. The initial value of the E-bit is zero.

LzLabs GmbH. Ex. 1016-354



PMCW Format

Limit Mode (LM): Bits 9-10 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 the section "Address-Limit Checking" in Chapter 17, "I/O Support Functions.") 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 the section "Operation-Request Block" later in this chapter.) The definitions of these bits, whose values are used during data transfer, are as follows:

Bits 9 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.

The LM can be set to any of the first three bit combinations shown above by MODIFY SUBCHANNEL. Specification of the reserved bit combination in the operand causes an operand exception to be recognized when MODIFY SUBCHANNEL is executed.

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 be set to any value by MODIFY SUBCHANNEL. The definitions of these bits are as follows:

Bit 11, Measurement-Block-Update Enable:

0 Initialized value. The subchannel is not enabled for the measurement-block update. Storing of measurement-block parameters does not occur.

1 The subchannel is enabled for measurement-block update. If the measurement-block-update mode of the channel subsystem is active, the measurement parameters are accumulated in the measurement block at the time channel-program execution is completed or suspended at the subchannel, provided no error conditions described by subchannel logout have been detected. If the measurement-block-update mode is not active, no measurement-block parameters are 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 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 device-connect-time interval 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 not active no measurement values are stored in the ESW.

The meaning of the measurement-mode (MM) enable bits described above applies when the timing-facility bit for the subchannel is one. When the timing-facility

LzLabs GmbH. Ex. 1016-355

bit is zero, the effect of the MM bits is changed, as described below under "Timing Facility." (For more discussion on measurement modes, see the sections "Measurement-Block-Update Facility" and "Device-Connect-Time-Measurement Facility" in Chapter 17, "I/O Support Functions.")

Multipath Mode (D): Bit 13 of word 1, when one, indicates that the subchannel operates in multipath mode when executing an I/O operation or chain of I/O operations. For proper operation in 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 execution of a start function in multipath mode, a device is allowed to request service from the channel subsystem over any of the channel paths designated in the subchannel as being available for selection (see the discussions of logical-path mask and path-available mask later in this section). Bit 13, when zero, indicates that the subchannel operates in single-path mode when executing an I/O operation or chain of I/O operations. In single-path mode, the entire start function is executed 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 initial value of the D bit is zero.

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 parameter 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 values can be measured for the subchannel. (See the sections "Measurement-Block-Update Facility" and "Device-Connect-Time-Measurement Facility" in Chapter 17, "I/O Support Functions.")

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: Bits 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 paths to the associated device. A bit set to one means that the corresponding path is logically available; a zero means the corresponding path is logically not available. When a channel path is logically not available, the channel subsystem does not use that channel path to initiate execution 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 LPM is ignored whenever a clear, halt, resume, or start function is executed. (See the section "Channel-Path Allegiance" later in this chapter). If the subchannel is idle, the LPM is ignored whenever the control unit initiates a request to present alert status to the channel subsystem. The LPM can be set to any value by START SUBCHANNEL and MODIFY SUBCHANNEL. The initial value of the LPM is the same as that of the path-installed mask (PIM) defined later in this section.

Path-Not-Operational Mask (PNOM): Bits 8-15 of word 2, when ones, indicate those paths having a corresponding path-operational-mask (POM) bit (see below) that was set to one when the device appeared not operational while trying to perform either a start, resume, halt, or clear function. If the PNOM contains all zeros, then a not-operational condition was not encountered on any paths whose corresponding POM bit was one. The initial value of the PNOM is all zeros.

Programming Note: The PNOM field does not always reflect the current state of the associated channel paths. For example, during attempted initiation of a start function, a path-not-operational condition may be specified in the PNOM; however, the channel path may subsequently have been used as a result of the device reconnecting to the channel path to continue command chaining, or the path may have appeared operational to the channel subsystem during execution of a subsequent start function. Consequently, the program should associate the POM with the PNOM whenever attempting to determine those channel paths that are in the path-not-operational state.

LzLabs GmbH. Ex. 1016-356

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. The bit corresponding to the path in use is set whenever one of the following occurs:

1. The first command of a start or resume function is accepted by the device (see the section "Activity Control" in Chapter 16, "I/O Interruptions").

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 that is recognized as an interruption condition, or a condition has been recognized that suppresses command chaining (see the section "Interruption Conditions" in Chapter 16, "I/O Interruptions").

4. An interface-control-check condition has been recognized (see the section "Interface-Control Check" in Chapter 16, "I/O Interruptions"), and no 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 logical 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 logical paths that are physically installed. 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 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 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 the section "SET CHANNEL MONITOR" in Chapter 14, "I/O Instructions") and the subchannel is enabled for the mode (see the discussion of the measurement-mode-enable bits, bits 11-12, earlier in this section). When used, five zero bits are appended on the right and the result added to the measurement-block-origin address specified by SET CHANNEL MONITOR. The calculated address, called the measurement-block address, specifies the beginning of a 32-byte storage area where 16 bytes of measurement parameters are stored (see the section "Measurement Block" in Chapter 17, "I/O Support Functions"). The initial value of the MBI is zero. The MBI may be set to any value by MODIFY SUBCHANNEL.

Path-Operational Mask (POM): Bits 16-23 of word 3, unless modified by the program, indicate the last-known operational state of the device on the corresponding paths. If a POM bit is one, then the device was operational on the corresponding path the last time the device was selected on that channel path, or no attempt has been made to select the device since the bit was last set to one. If a POM bit is zero, the device appeared not operational on the corresponding channel path the last time there was an attempt to select the device to initiate or resume execution of a start function, or the program set it to zero. In this situation, if the device now appears operational, the POM bit is set to one and remains until the device next appears not-operational on that path. The initial value of the POM is all ones. The POM can be set to any value by MODIFY SUBCHANNEL.

Path-Available Mask (PAM): Bits 24-31 of word 3 indicate the physical availability of installed paths. 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 PAM bit is zero, the corresponding channel path may, depending upon the model and the extent of failure, be used during execution of the reset-channel-path function. A PAM bit may be set to zero as a result of reconfiguring the system, or it may occur because the corresponding channel path is in the check-stop state.

The initial value of the PAM reflects the complement of 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 is indicated by a change in the setting of the corresponding PAM bit in 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.

LzLabs GmbH. Ex. 1016-357

Channel-Path Identifiers (CHPIDs): Words 4 and 5 contain eight one-byte channel-path identifiers corresponding to the logical 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: Word 6 of the SCHIB is reserved and is stored as zeros by STORE SUBCHANNEL. Bits 0-31 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 contain a copy of the SCSW, whose format is described in the section "Subchannel-Status Word" in Chapter 16, "I/O Interruptions." The validity of the SCSW fields stored when STORE SUBCHANNEL is executed is the same as if TEST SUBCHANNEL were executed (see the section "TEST SUBCHANNEL" in Chapter 14, "I/O Instructions").

Model-Dependent Area

Words 10-12 contain model-dependent data. The definition and use of information in this area depends on the model and is described in the System Library publication for the model.

Summary of Modifiable Fields

The figure "Modification of Subchannel Fields" lists the initial settings for fields in a subchannel whose device-number-valid bit is set to one, and indicates what modifies the fields.

All of the PMCW fields contain valid information when STORE SUBCHANNEL is issued to an idle subchannel. Subchannel fields that the channel subsystem does not modify contain valid information whenever STORE SUBCHANNEL is issued. The validity of the subchannel fields that are modifiable by the channel subsystem depends upon the state of the subchannel at the time STORE SUBCHANNEL is issued.

LzLabs GmbH. Ex. 1016-358

Subchannel Field	Initial Value[1]	Program Modifies Via	Modified by Channel Subsystem[5]
Interruption parameter	Zeros	MSCH,SSCH	No
Interruption-subclass code	Zeros	MSCH	No
Enabled	Zero	MSCH	No
Limit mode	Zeros	MSCH	No
Measurement mode	Zeros	MSCH	Yes[4]
Multipath mode	Zero	MSCH	No
Timing facility	Installed value[2]	None	No
Device number valid	Installed value[2]	None	No
Device number	Installed value[2]	None	No
Logical-path mask	Equal to path-installed mask value	MSCH,SSCH	No
Path-not-operational mask	Zeros	CSCH,SSCH, RSCH[6]	Yes
Last-path-used mask	Zeros	CSCH	Yes
Path-installed mask	Installed value[2]	None	No
Measurement-block index	Zeros	MSCH	No
Path-operational mask	Ones	CSCH,MSCH	Yes
Path-available mask	Installed values [2] [3]	None	Yes[3]
Channel-path ID 0-7	Installed value	None	No
Subchannel-status word	Zeros	TSCH	Yes
Model-dependent area	*	None	*

Modification of Subchannel Fields (Part 1 of 2)

LzLabs GmbH. Ex. 1016-359

Explanation:

1 These fields are meaningless if the subchannel is not valid. Initializing 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 "CRW Contents" in Chapter 17, "I/O Support Functions.")

2 This information is entered when the channel-subsystem configuration is established.

3 The channel subsystem may modify the PAM to reflect changes in the system configuration caused by partitioning or unpartitioning paths because of reconfiguration or permanent failure of part of the I/O system.

4 When any of the following error conditions associated with the measurement-block-update-mode are detected, the measurement-block-update mode is disabled by the channel subsystem (bit 11, word 1, of the SCHIB zero) in the affected subchannel. The device-connect-time-measurement-mode-enable bit (bit 12, word 1 of the SCHIB) is never modified by the channel subsystem.

   Measurement program check
   Measurement protection check
   Measurement data check
   Measurement key check

5 Subchannel fields that are not normally modifiable by the channel subsystem may be modified by external means. When this occurs, the program is notified of the change via a channel report made pending at the time of the change.

6 The mask is modified via RSCH only when the subchannel is in the suspended state at the time RSCH is executed.

* Model-dependent.

Modification of Subchannel Fields (Part 2 of 2)

Programming Notes

1. System performance may be degraded in the event that the LPM is not used to make paths logically not available that have been indicated to be not operational via the PNOM.

2. If, during execution of a start function, a PAM bit is set to zero because a channel-path failure has been recognized, continued execution 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 execution of the clear function.

3. If the same MBI is placed in more than one subchannel by the program, the monitoring facility updates the same locations with measurement data relating to more than one subchannel. Since the update mechanism on some models may not be synchronized among subchannels, the resulting values of the measurement parameters in main storage may be unpredictable. (See the section "Measurement-Block-Update Facility" in Chapter 17, "I/O Support Functions.")

LzLabs GmbH. Ex. 1016-360

4. Modification of the I/O configuration (reconfiguration) may be accomplished in various ways depending on the model. When the physical availability of a channel path is affected, the reconfiguration procedure results in a change in the setting of the corresponding PAM bit.

5. The definitions of the PNOM and POM 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 every channel path available for selection is found to be not operational 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 which path or paths were found to be not operational, and the corresponding POM bits have been set to zeros. The next START SUBCHANNEL causes the channel subsystem to again attempt selection via all channel paths that are available for selection. If selection is again not successful, 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 paths that are available for selection. (See the section "Channel-Path Availability" later in this chapter for a description of the term "available for selection.") When the N bit in the SCSW is zero, the PNOM is also zeros.

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 execution 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.

Execution of the clear function at a subchannel clears any currently existing allegiance condition in the subchannel for all channel paths.

Execution of the reset-channel-path function clears all currently existing allegiances for that channel path in all subchannels.

When a PAM bit is set to zero, all internal indications of prior allegiance conditions are cleared in all subchannels having access to the designated channel path.

WORKING ALLEGIANCE

A subchannel has a working allegiance for a channel path when the subchannel becomes device-active. Once a working allegiance is established, the subchannel maintains it 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 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 device is no longer device-active. If a dedicated allegiance is recognized or if the subchannel is operating in single-path mode, a working allegiance is maintained only for a single path.

A working allegiance is cleared in any subchannel having access to a channel path if the corresponding PAM bit is set to zero.

ACTIVE ALLEGIANCE

A subchannel has an active allegiance established for a channel path no later than when active communication has been initiated on that 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 except to accept no-longer-busy indications. 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.

LzLabs GmbH. Ex. 1016-361

A working allegiance can become an active allegiance.

DEDICATED ALLEGIANCE

If the PAM bit corresponding to a channel path is one, a dedicated allegiance may be recognized for that channel path. If the PAM bit is zero, a dedicated allegiance cannot be recognized for the corresponding channel path. The subchannel establishes a dedicated allegiance for a channel path when the channel subsystem accepts from the device a status byte containing unit check. A dedicated allegiance is maintained until the subchannel is no longer start-pending (unless it becomes suspended) or resume-pending following execution of the next start-, clear-, or reset-channel-path 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 reset-channel-path function is executed, the subchannel establishes a working allegiance when the dedicated allegiance ends. This occurs when the subchannel becomes subchannel-and-device-active. While a dedicated allegiance exists in a subchannel, only that 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 path.

A currently existing dedicated allegiance is cleared in any subchannel having access to a channel path when the corresponding PAM bit is set to zero 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 via that path. A channel path is said to be not physically available when the corresponding path-available-mask (PAM) bit is zero in all subchannels by which the device is accessible via that channel path (see the description of the PAM in the section "Path-Management-Control Word" earlier in this chapter).

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 (the bits in the LPM and PAM corresponding to the channel path are ones). If a dedicated allegiance exists at a subchannel for the channel path, only that 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 the figure "Path Condition and Path-Availability Status for PIM, PAM and LPM Values" 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.

LzLabs GmbH. Ex. 1016-362

Value of Bit 'n'			Path Condition[2]	Path State
PIM	PAM	LPM		
0	0[1]	-	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	Not active	Available for selection

Explanation:

[1] A PAM bit cannot have the value one when the corresponding PIM bit has the value zero.

[2] If the channel path is recognized as being used in actively communicating with a device, the path condition is described as active. If the channel path is recognized as not being used in active communication, the condition is described as not active.

[3] If a dedicated allegiance exists to this path for this subchannel, the state of this bit is ignored, and the 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 (see the discussion of "Channel Path Terminal" in Chapter 17, "I/O Support Functions").

X Path condition is not meaningful.

- Bit value is not meaningful.

Path Condition and Path-Availability Status for PIM, PAM and LPM Values

CONTROL-UNIT TYPE

In the sections "Clear-Function Execution," "Halt-Function Execution," and "Start-Function and Resume-Function Execution" that follow, 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.

CLEAR-FUNCTION EXECUTION

Subsequent to execution of CLEAR SUBCHANNEL, the channel subsystem performs the clear function. Execution of the clear function consists in (1) executing a path-management operation, (2) modifying fields in the subchannel, and (3) issuing the clear signal to the associated device and causing the subchannel to be made status-pending, indicating conclusion of the clear-function execution.

Clear-Function Path Management: A path management operation is performed by the channel subsystem during execution of the clear function to identify and choose an available channel path or set of available channel paths of which one can be used in issuing the clear signal to the associated device. The actions taken are as follows:

1. If the device to be signaled is currently communicating with the channel subsystem on a channel path and, for the associated subchannel, either an allegiance to that path exists or the device is attempting to establish an allegiance to that path, or no allegiance exists for any channel path, then that channel path is chosen.

2. If the device to be signaled is not communicating with the channel subsystem on any channel path, the channel path or set of channel paths is chosen as follows:

   a. If a dedicated allegiance for a channel path exists in the associated subchannel, that channel path is the only path chosen.

   b. If no channel paths are available for selection and no dedicated allegiance exists in the subchannel for a channel path, a channel path is not chosen.

   c. If a working allegiance exists in the subchannel for a channel path or set of channel paths, the channel path or set of channel paths, as appropriate, is chosen.

   d. If no allegiance exists in the subchannel for any channel path but a last-used channel path is specified that is available for selection, that path is the only path chosen.

   e. If a last-used channel path is specified but it is not available for selection, then,

LzLabs GmbH. Ex. 1016-363

depending upon the model, the remaining set of channel paths available for selection is chosen or no paths are chosen.

f. If no allegiance exists in the subchannel and no last-used channel path is specified, the set of channel paths that is available for selection is chosen. If no paths are available for selection, a channel path is not chosen.

3. When the set of channel paths has been chosen, one of those paths from the set is selected. The state of the selected path may, however, prevent the channel subsystem from signaling the device. In this situation, execution of the clear function is ended, and an interruption condition is recognized. Execution of the clear function may be ended for any of the following reasons:

   a. An active allegiance exists in some other subchannel for the channel path that has been selected.

   b. The device to be signaled is attached to a type-1 control unit, and an allegiance exists for the same path in another subchannel whose device is attached to the same control unit. If the existing allegiance for the other subchannel is recognized to be a working allegiance, execution of the clear function is ended if primary status has not been received for that subchannel. Otherwise, execution of the clear function is not ended, the device to be signaled is selected using that channel path, and the clear signal is issued.

   c. The device to be signaled is attached to a type-3 control unit, and a dedicated allegiance exists for the same channel path in another subchannel whose device is attached to the same control unit.

   d. A busy, path not-operational, or error condition has been recognized on the selected channel path.

4. If the device to be signaled is attached to a type-2 control unit, a channel path is selected from the chosen set of channel paths, subject to actions 1-3, and this occurs independent of the states of other subchannels associated with devices attached to the same control unit or allegiances to channel paths for those same devices.

Programming Note

Subsequent to the execution 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.

Clear-Function Subchannel Modification: Path-management-control fields of the subchannel are modified during execution of the clear function. This modification occurs starting at a point of execution that is (1) logically after channel-path selection has occurred or (2) logically after it is determined that no channel path is available for selection, but (3) prior to attempting actual selection of the device and issuing the clear signal if a channel path was available for selection. The path-management-control fields that are modified are as follows:

1. The path-operational mask (POM) is set to all ones.

2. The last-path-used mask (LPUM) is set to all zeros.

3. The path-not-operational mask (PNOM) is set to all zeros.

Clear-Function Signaling and Completion: Subsequent to selecting the channel path and modifying the path-management-control fields, the channel subsystem attempts to select the device to issue the clear signal and then cause the subchannel to be made status-pending. (See the section "Clear Signal" in Chapter 17, "I/O Support Functions.) If the associated device is currently executing an operation with the channel subsystem, that operation is terminated at the subchannel and, if conditions allow, is also terminated at the device. When the subchannel is set status-pending, execution of the clear function is considered to be completed independent of whether the clear signal was successfully issued to the device.

If the clear signal is not issued to the device because of any of the following reasons, the subchannel is made status-pending with the clear-pending bit set to one.

a. The physically available channel path by which the clear signal is to be issued is not available for selection because (1) the channel

LzLabs GmbH. Ex. 1016-364

path is currently being used to actively communicate with a different device, (2) the channel path has been recognized to be in the terminal state and appears to be in use (see the section "Channel-Subsystem Recovery" in Chapter 17 "I/O Support Functions", or (3) no dedicated allegiance exists and the corresponding bits in the PAM and LPM are not both ones.

b. No control unit responded when the channel subsystem attempted to select the device to issue the clear signal.

c. The control unit or device signaled a busy condition when the channel subsystem attempted to issue the clear signal.

d. An error condition was encountered while attempting to issue the clear signal.

e. The device 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 (except for a working allegiance when the subchannel is no longer subchannel-active).

f. The device 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.

Subsequently, if, during execution of a clear function, a path-not-operational condition has been recognized, the bit settings in the PNOM field are preserved until the subchannel next becomes clear-pending or start-pending, at which time the PNOM is again set to zeros.

HALT-FUNCTION EXECUTION

Subsequent to the execution of HALT SUBCHANNEL, the channel subsystem performs the halt function. Execution of the halt function consists in: (1) executing a path-management operation and (2) issuing the halt signal to the associated device and causing the subchannel to be made status-pending, indicating conclusion of the halt-function execution.

Halt-Function Path Management: A path-management operation is performed by the channel subsystem during execution of the halt function to identify and choose an available channel path or set of available channel paths of which one can be used in issuing the halt signal to the associated device. The actions taken are as follows:

1. If the device to be signaled is currently communicating with the channel subsystem on a channel path and, for the associated subchannel, either an allegiance to that path exists, or the device is attempting to establish an allegiance to that path, or no allegiance exists for any channel path, then that channel path is chosen.

2. If the device to be signaled is not communicating with the channel subsystem on any channel path, the channel path or set of channel paths is chosen as follows:

   a. If a dedicated allegiance for a channel path exists in the subchannel, that channel path is the only path chosen.

   b. If no dedicated allegiance exists in the subchannel for a channel path and no channel paths are available for selection, a channel path is not chosen.

   c. If a working allegiance exists in the subchannel for a channel path or set of channel paths, the channel path or set of channel paths, as appropriate, is chosen.

   d. If no allegiance exists in the subchannel for any channel path but a last-used channel path is specified that is available for selection, that path is the only path chosen.

   e. If a last-used channel path is specified but it it not available for selection, then, depending upon the model, the remaining set of channel paths available for selection is chosen or no paths are chosen.

   f. If no allegiance exists in the subchannel and no last-used channel path is specified, the set of channel paths that is available for selection is chosen. If no paths are available for selection, a channel path is not chosen.

3. When the set of channel paths has been chosen, one of those paths is selected for issuing the halt signal to the device. The state of the selected path may, however, prevent the channel subsystem from signaling the device. In this situation, the halt function remains pending until the selected channel path can be used in signaling the device.

LzLabs GmbH. Ex. 1016-365

4. The halt function remains pending at the subchannel for any of the following reasons:

   a. An active allegiance exists in some other subchannel for the channel path that has been selected.

   b. The device to be signaled is attached to a type-1 control unit, and an allegiance exists for the same channel path in another subchannel whose device is attached to the same control unit. If the existing allegiance for the other subchannel is recognized to be a working allegiance, execution of the halt function remains pending if primary status has not been received for that subchannel. Otherwise, the device to be signaled is selected using that channel path and the halt function is executed.

   c. The device to be signaled is attached to a type-3 control unit, and a dedicated allegiance exists for the same channel path in another subchannel whose device is attached to the same control unit.

5. If the device to be signaled is attached to a type-2 control unit, a channel path is selected from the chosen set of channel paths, subject to the above actions, and this occurs independent of the states of other subchannels associated with devices attached to the same control unit or allegiances to channel paths for those same devices.

Halt-Function Signaling and Completion: Subsequent to selecting the channel path, the channel subsystem attempts to select the device to issue the halt signal and cause the subchannel to be set status-pending with status presented by the device, if any. (See the section "Halt Signal" in Chapter 17, "I/O Support Functions.") When the subchannel is subchannel-and-device-active or only device-active during execution 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, because of an existing condition, 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. In this situation, the status-pending interruption condition is generated after the halt signal is issued or after the channel subsystem has determined that an abnormal condition precludes issuing the halt signal to the device. In the case 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, so the subchannel is made primary-and-secondary-status-pending after the halt signal is issued. The device-status field of the SCSW contains zeros in this case. When the subchannel is set status-pending, execution of the halt function is considered to be completed independent of whether or not the halt signal was successfully issued to the device.

If the channel subsystem is unable to issue the halt signal because of any of the following reasons, the subchannel is made status-pending with combinations of primary, secondary, and alert status as a function of the subchannel state and the status presented by the device, if any. The halt-pending bit, however, is stored as one when TEST SUBCHANNEL is executed.

   a. Prior to recognition of the halt-pending condition, the subchannel becomes status-pending with status other than intermediate status alone.

   b. The installed channel path by which the halt signal is to be issued is not available for selection.

   c. The device appeared not operational on the selected channel path.

   d. An error condition was encountered while attempting to issue the halt signal to the I/O device.

If the control unit or device indicates busy, the halt function is held pending until either the no-longer-busy indication is received or the device attempts to communicate with the channel subsystem to continue the chain of commands. When either the no-longer-busy indication is received by the channel subsystem or the attempt to communicate with the channel subsystem is successful, the halt signal is issued on that channel path.

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 devices that are not active or that are executing a mechanical operation in which data is not transferred across the channel path, such as rewinding tape or positioning a

LzLabs GmbH. Ex. 1016-366

disk-access mechanism, depends upon the control-unit or device model and is described in the appropriate System Library publication. If the device is executing 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 attempts to select 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 execution of the halt function, and the device is logically disconnected from the channel path. If the halt function is addressed to a subchannel executing 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 executing 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 immediately after the halt signal is issued. 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 executing 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 becomes status-pending, only after receiving 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 the section "Subchannel-Status Word" in Chapter 16, "I/O Interruptions").

The termination of an I/O operation by execution of the halt function can result in as many as 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, with the exception that the command address in the associated SCSW indicates the point at which the I/O operation was terminated, and the subchannel-status bits may reflect unusual conditions that were detected. If the halt signal was issued before all data specified for the operation has been transferred, incorrect length is indicated, subject to the control of the SLI flag in the current CCW.

The second interruption condition occurs if device-end status was not presented with the channel-end interruption condition. In this situation, the SCSW contains unpredictable values in the subchannel-key, command-address, and count fields.

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 Note

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.

START-FUNCTION AND RESUME-FUNCTION EXECUTION

Subsequent to 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. Execution of a start or resume function consists in: (1) executing a path-management operation, (2) executing 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 completion of the start-function execution. (Conclusion of a start function is described in Chapter 16, "I/O Interruptions.") Execution of the resume function, in effect, causes a start function to be performed. The difference between the start and resume functions is that the start function initiates execution of a currently pending channel program, while the resume

LzLabs GmbH. Ex. 1016-367

function initiates execution of a suspended channel program, if any. Otherwise, the resume function is executed as if it were a start function (see the discussion on the resume-pending bit in the section "Activity Control" in Chapter 16, "I/O Interruptions").

Start-Function and Resume-Function Path Management: A path-management operation is performed by the channel subsystem during execution of either a start or resume function to identify and select an available channel path that can be used in initiating an I/O operation with the associated 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. If the subchannel is currently start-pending alone, execution 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 selects that path. 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 a no-longer-busy indication is received on that channel path. When the no-longer-busy indication is received, execution of the pending start function is initiated on that channel path.

3. If no channel paths are 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 or device-busy condition on one or more of those paths, or a combination of those conditions on one or more of those 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 path is selected 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 if no allegiances exist as described above, that path is chosen. If no other paths are 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 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 a selection of the device to initiate execution of the first command of a pending channel program, the 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 a no-longer-busy indication is received. The no-longer-busy indications for control-unit busy and device busy are control-unit end and device end, respectively.

8. Except when a dedicated allegiance exists at the subchannel for a channel path and unless multipath mode is specified in the subchannel when the channel subsystem receives a busy indication as described in action 7, the channel subsystem attempts selection of the device by choosing an alternate channel path that is available for selection and continues this process until either the start function is initiated or selection of the device has been attempted on all channel paths that are available for selection. In the latter case, the start function remains pending until a no-longer-busy indication of either control-unit end or device end is received from the device, whichever is applicable. (See the section "Subchannel-Information Block" earlier in this chapter, concerning multipath mode.) If the subchannel

LzLabs GmbH. Ex. 1016-368

has a dedicated allegiance, action 2 applies.

9. When, during the selection attempt to transfer the first command, the device appears not operational and the corresponding bit in the POM is currently one, a path-not-operational condition is recognized, the corresponding bit in the PNOM is set to one, and the corresponding bit in the POM is set to zero (see the section "Subchannel-Information Block" earlier in this chapter). The bit settings in the PNOM field are preserved until the subchannel next becomes clear-pending, start-pending, or resume-pending (if the subchannel was suspended), at which time the PNOM is again set to zeros. If, however, the corresponding bit in the POM is currently zero, the path-not-operational condition is ignored, and the condition is not indicated in the PNOM. 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 path that is available for selection, so 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 which have not been tried, one of these channel paths is chosen to attempt selection of the device and transfer the first command.

   c. If no dedicated allegiance exists, no alternate channel paths are available for selection which 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 either 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 which 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 selections of the device occur according to the LPUM if the multipath mode is not specified in the subchannel. For example, if command chaining is indicated, the channel subsystem transfers the first and all subsequent commands describing a chain of I/O operations over the same path.

EXECUTION OF I/O OPERATIONS

Subsequent to channel-path selection, the channel subsystem, if conditions allow, initiates execution of an I/O operation with the associated device. Execution of additional I/O operations may follow 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 while channel-program execution is suspended at the subchannel (see the section "Suspension of Channel-Program Execution" later in this chapter), the subchannel is active with the execution of the start function from the acceptance of the first command by the device and until the primary-interruption condition is received at the subchannel. Normally, the primary-interruption condition is the channel-end signal or, in the case of command chaining, the channel-end signal for the last CCW of the chain. (See the section "Primary-Interruption Condition" in Chapter 16, "I/O Interruptions.") The operation at the device lasts until the secondary-interruption condition is signaled to the channel subsystem. Normally, the secondary-interruption condition is the device-end signal or, in the case of command chaining, the device-end signal for the last CCW of the chain of operations. (See the section "Secondary-Interruption Condition" in Chapter 16, "I/O Interruptions.")

Programming Note

An I/O operation or chain of I/O operations is normally executed by the channel subsystem and the device operating in single-path mode. In single-path mode, all transfers of commands, data,

LzLabs GmbH. Ex. 1016-369

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 additional mode of execution is possible. This second mode of execution occurs when the channel subsystem and the device having the dynamic-reconnection feature installed execute an I/O operation or chain of I/O operations in multipath mode. To operate in multipath mode, both the subchannel and the device must be set up for multipath-mode operation. The subchannel is set up for multipath-mode operation by using MODIFY SUBCHANNEL to set the multipath-mode (D) bit to one in the subchannel. (See the description of the multipath-mode bit in the section "Subchannel-Information Block" earlier in this chapter.) A device having the dynamic-reconnection feature installed is set up for multipath-mode operation by issuing to the device the specific commands for that purpose, as described in the System Library publication for the device. The general procedures for setting up a device for multipath-mode operation and for returning to single-path mode are as follows:

1. A set-multipath-mode-type 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 paths must be made logically not available for selection by having the corresponding LPM bits set to zeros while the subchannel is operating in multipath mode (that is, while the multipath-mode bit in the subchannel is one).

2. When a set-multipath-mode-type command is issued to a device, the LPM must specify only a single path as logically available in order to disable alternate 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.

3. Procedures analogous to those above should be used whenever the membership of a multipath group is changed.

4. To leave multipath mode and continue processing in single-path mode, either of the following two procedures may be used:

   a. Successful execution of a disband-multipath-mode-type command by the device on any channel path of the multipath group must be followed by either (1) the execution of MODIFY SUBCHANNEL to turn off multipath mode at the subchannel, or (2) the specification of only a single channel path as logically available in the LPM. No start function must be issued to a subchannel operating in multipath mode with multiple 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. Successful execution of a resign-multipath-mode-type command by the device on each channel path of the multipath group (the reverse of setting up the multipath group described in item 1 above) must be followed by either (1) the execution of MODIFY SUBCHANNEL to turn off multipath mode at the subchannel, or (2) the specification of only a single channel path as logically available in the LPM. No start function must be issued to a subchannel operating in multipath mode with multiple path 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.

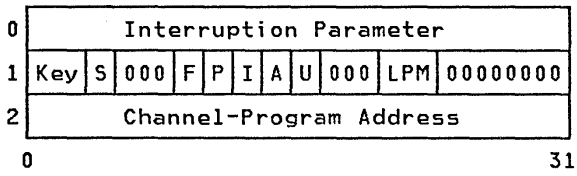
The maximum amount of information that can be transferred in one I/O operation is one block. An I/O operation is terminated when the associated main-storage area is exhausted or the end of the block is reached, whichever occurs first. For some operations, such as writing on a magnetic-tape unit or at an inquiry station, blocks are not defined, and the amount of information transferred is controlled only by the program.

LzLabs GmbH. Ex. 1016-370

OPERATION-REQUEST BLOCK (ORB)

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 passed to the channel subsystem during the execution of START 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.

The ORB has 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. These bits are placed in word 1 of the interruption code at I/O-interruption time and when an interruption request is cleared by 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 a storage key during these storage references. For details, see the section "Key-Controlled Protection" in Chapter 3, "Storage."

Suspend Control (S): Bit 4 of word 1 controls execution of the suspend function for the channel program identified in the ORB. The setting of the S bit applies to all CCWs of the channel program specified by the ORB (see the section "Commands" later in this chapter). When bit 4 is one, suspend control is specified, and channel-program suspension occurs when a valid suspend flag is detected in a CCW. If bit 4 is zero, suspend control is not specified, and the presence of the suspend flag in any CCW of the channel program causes a program-check condition to be recognized.

Reserved: Bits 5-7 of word 1 are reserved for future use and must be zeros; otherwise, depending upon the model, 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) which make up the channel program designated by the channel-program-address field. If bit 8 of word 1 is zero, format-0 CCWs are specified. If it is one, format-1 CCWs are specified. (See the section "Channel-Command Word" later in this chapter, for the definition of the CCW formats).

Prefetch Control (P): Bit 9 of word 1 indicates whether or not unlimited prefetching of CCWs is allowed for the channel program. If this bit is zero, no prefetching is allowed, except in the case of data chaining on output, where the prefetching of one CCW is allowed. If this bit is one, unlimited prefetching is allowed.

Initial-Status-Interruption Control (I): Bit 10 of word 1 indicates whether or not the channel subsystem must verify to the program that the device has accepted the first command associated with a start or resume function. If the I bit is specified as one in the ORB, then when initial status is received and the subchannel becomes active, indicating that the first command has been accepted for this start or resume function, the Z bit (see the section "Subchannel-Status Word" in Chapter 16, "I/O Interruptions") is set to one in this subchannel, and the subchannel becomes intermediate-status-pending.

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 primary-status-pending. Intermediate-status-pending may also be indicated in this case when the command is accepted if the first CCW contained the PCI flag.

Address-Limit-Checking Control (A): Bit 11 of word 1 indicates whether or not address-limit checking is specified for the channel program. If 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 the section "Path-Management-Control Word" earlier in this chapter). If 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.

LzLabs GmbH. Ex. 1016-371

Suppress-Suppended-Interruption Control (U): Bit 12 of word 1, when one, indicates that the channel subsystem suppresses the generation of a subchannel-suspended-interruption condition when the subchannel is suspended. When bit 12 is zero, the channel subsystem generates an intermediate-interruption condition whenever the subchannel is suspended during execution of the channel program.

Reserved: Bits 13-15 of word 1 are reserved for future use and must be zeros; otherwise, an operand exception or a program-check condition is recognized, depending on the model.

Logical-Path Mask (LPM): Bits 16-23 of word 1 are preserved unmodified in the subchannel and indicate 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 path is to be considered as being logically available; a zero specifies that the corresponding path is recognized as being logically not available. If a path is specified by the program as being logically not available, the channel subsystem does not use that path to execute clear, halt, resume, or start functions when requested by the program, except when a dedicated-allegiance condition exists for a 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

Reserved: Bits 24-31 of word 1 are reserved for future use and must be set to zeros; otherwise, an operand exception or a program-check condition is recognized, depending on the model.

Channel-Program Address: Bits 0-31 of word 2 designate the location of the first CCW in absolute storage. If format-0 CCWs have been specified in bit 8 of word 1, then bits 0-7 of word 2 must be zeros. If format-1 CCWs have been specified, then bit 0 of word 2 must be zero. If bit 0 is not zero, an operand exception or a program-check condition is recognized, depending on the model. If format-0 CCWs have been specified and bits 1-7 do not contain zeros, a program-check condition is recognized.

The three rightmost bits of the channel-program address must be zeros, specifying the CCW on a doubleword boundary; otherwise, a program-check condition is recognized.

If the CCW address specifies a location protected against fetching or specifies 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.

Programming Notes

1. Bit positions of the ORB which 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 by the channel subsystem and executed in the sequence specified by the CPU program. Contiguous CCWs are linked by the use of the chain-data or chain-command flags, and noncontiguous CCWs may be 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 the section "Data Chaining" later in this chapter). When chaining is not specified, a CCW is no longer current after TEST SUBCHANNEL clears the start-function bit in the subchannel.

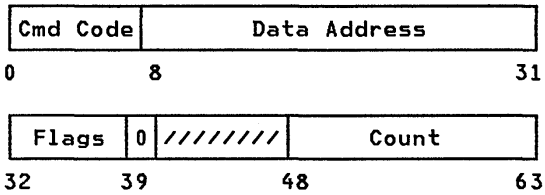
The location of the first CCW of a channel program is specified in the ORB that is the operand of START SUBCHANNEL. The first CCW is fetched subsequent to the execution of the instruction. The format of the CCWs fetched by the channel subsystem is specified by bit 8 of word 1 of the ORB. Each additional 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. 1016-372

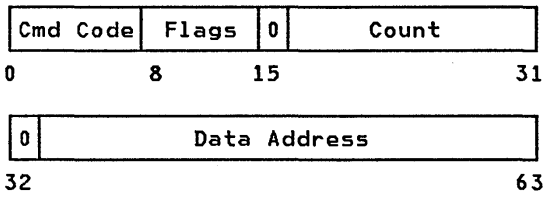
CCWs have either of two different formats. The two formats do not differ in the information contained in the CCW but only in the arrangement of the fields within the CCW.

The formats are designated format-0 and format-1 CCWs and are defined as follows:

Format-0 CCW



Format-1 CCW



Format-0 CCWs can be located anywhere in the first 16,777,216 bytes of main storage.

Format-1 CCWs can be located anywhere in main storage.

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) specify a location in absolute storage. It is the first location referred to in the area designated by the CCW. If a byte count of zero is specified, this field is not checked.

Chain-Data (CD) Flag: Bit 32 (format 0) or bit 8 (format 1), when one, specifies chaining of data. It causes the storage area designated by the next CCW to be used with the current I/O operation. When the CD flag is on 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 zero, specifies chaining of commands. It causes the operation specified by the command code in the next CCW to be initiated on 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 one, and the CD flag is zero, command chaining takes place, regardless of the presence of an incorrect-length condition. This bit is indicated in all CCWs where suppression of an incorrect-length indication is required.

Skip (SKIP) 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 by using the appropriate subchannel when the CCW 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 execution of the CCW containing the S flag. The S flag is valid when bit 4, 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.

Bit position 39 (format 0) or bit position 15 (format 1) of every CCW other than a format-0 CCW specifying transfer in channel must contain zeros. Additionally, if indirect data addressing is specified, bits 30-31 (format 0) or bits 62-63 (format 1) of the CCW must be zeros, indicating a word boundary, and bit 0 of the first entry of the indirect-data-address list must be zero. Otherwise, depending upon the model, a program-check condition may be generated (see the section "CCW Indirect Data Addressing" later in this chapter). 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.

LzLabs GmbH. Ex. 1016-373

The contents of bit positions 40-47 of a format-0 CCW are ignored.

Programming Note

Bit position 39 of a format-0 CCW or bit position 15 of a format-1 CCW, which presently must contain zero, may in the future be assigned for the control of new functions. It is recommended, therefore, that this bit position 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. A detailed description of each command appears under "Commands," later in this chapter.

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 situation, 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 conditions 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 the figure "Command-Code Assignment." The symbol x indicates that the bit position is ignored; m identifies a modifier bit.

Code	Command
xxxx 0000	Invalid
mmmm 0100	Sense
1110 0100	Sense ID
xxxx 1000	Transfer in channel[1]
0000 1000	Transfer in channel[2]
mmmm 1000	Invalid[3]
mmmm 1100	Read backward
mmmm mm01	Write
mmmm mm10	Read
mmmm mm11	Control

Explanation:

m Modifier bit.

x Bit is ignored.

[1] Format-0 CCW.

[2] Format-1 CCW.

[3] Format-1 CCW with any of bits 0-3 nonzero.

Command-Code Assignment

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

Note: For a description of the storage area associated with a CCW when indirect data addressing is invoked, see the section "CCW Indirect Data Addressing" later in this chapter.

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 appears in 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

LzLabs GmbH. Ex. 1016-374

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 0, 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 16M bytes of storage, and format-1 CCWs may be located in any part of available storage provided in both cases that 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 provided and 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.

In the event the channel subsystem refers to a location not provided in the system, the program-check condition is generated. When the first CCW designated by the channel-program address is at a nonexistent location, the start function is not initiated at the device, the status portion of the SCSW is updated with the program-check indication, and the subchannel becomes primary-and-secondary status-pending, with a deferred condition code 1 indicated. Invalid data addresses, as well as any invalid CCW addresses detected on chaining or subsequent to the execution of START SUBCHANNEL, are indicated to the program with the interruption conditions at the conclusion of the operation or chain of operations.

During an output operation, the channel subsystem may fetch data from the 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 the next CCW 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, then any number of CCWs 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 of data or CCWs do not affect the execution of the operation and do not cause error indications until the I/O operation actually attempts to use the information. If the operation is concluded by the I/O device or by 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, 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 contains a zero, (2) data chaining is either not specified or is 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, 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.

CHAINING

When the channel subsystem has performed the transfer of information specified by a CCW, it can continue executing 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 specified by the original START SUBCHANNEL.

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

LzLabs GmbH. Ex. 1016-375

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 the figure "Subchannel-Chaining Action."

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 is used for the type of chaining specified in the CCW preceding the transfer in channel.

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.

LzLabs GmbH. Ex. 1016-376

Flags in Current CCW			Action in the Subchannel upon Exhaustion of Count or Receipt of Channel End					
			Immediate Operation			Regular Operation		
CD	CC	SLI	I	II	III	IV	V	VI
0	0	0	End, -	End, -	End, ϕ	Stop, IL	End, -	End, IL
0	0	1	End, -	End, -	End, -	Stop, -	End, -	End, -
0	1	0	CC	CC	End, ϕ	Stop, IL	CC	End, IL
0	1	1	CC	CC	CC	Stop, CC	CC	CC
1	0	0	End, -	PC	End, IL	CD	*	End, IL
1	0	1	End, -	PC	End, IL	CD	*	End, IL
1	1	0	End, -	PC	End, IL	CD	*	End, IL
1	1	1	End, -	PC	End, IL	CD	*	End, IL

Explanation:

I — Immediate operation with format-0 CCWs. The count field must contain a nonzero value or the operation is terminated by a program-check condition.

II — Immediate operation with format-1 CCW and CCW count field zero.

III — Immediate operation with format-1 CCW and CCW count field nonzero.

IV — Count exhausted, and end of block at device not reached.

V — Count exhausted, and channel end from device.

VI — Count not exhausted, and channel end from device.

End — Operation is terminated.

Stop — Device is signaled to terminate data transfer, but subchannel remains active until channel end is received.

IL — Incorrect length is indicated with the subsequent interruption condition generated at the subchannel.

\- — Incorrect length is not indicated with the subsequent interruption condition generated at the subchannel.

CC — Command chaining is performed by the channel subsystem upon receipt of device end.

Subchannel-Chaining Action (Part 1 of 2)

LzLabs GmbH. Ex. 1016-377

Explanation (Continued):

CD — The chain-data flag causes the channel subsystem to immediately fetch a new CCW for the same operation, and the operation normally continues. However, any CCW that has a count field containing the value zero and that either is fetched while data chaining or has the chain-data flag set to one causes the channel subsystem to terminate the operation with a program-check condition.

PC — The channel subsystem recognizes a program-check condition when a format-1 CCW having a count field containing the value zero either has the chain-data flag set to one or is fetched while data chaining. Therefore, these situations cannot occur validly.

◊ — The incorrect-length indication may or may not be indicated.

* — The situation where the residual count is zero but data chaining is indicated at the time the device provides channel end cannot validly occur. When data chaining is indicated, the new CCW takes control of the operation after transferring the last byte of data designated by the current CCW, but before the device provides the next request for data or status transfer. As a result, the channel end from the device is recognized only after the new CCW, which cannot contain a count of zero unless a programming error has been made, is in control of the operation.

Subchannel-Chaining Action (Part 2 of 2)

Programming Note

When bit 9 of ORB word 1 is one, unlimited fetching of chained CCWs 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 issued 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. Execution of the operation at the I/O device is not affected. When 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

LzLabs GmbH. Ex. 1016-378

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 execution 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 the section "Operation-Request Block" earlier in this chapter.)

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 may be prefetched at a time.

   If the ORB for the I/O operation specifies that prefetching is allowed, any number of CCWs 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. Therefore, neither the program nor the I/O device is aware of any differences whether or not prefetching of CCWs is being performed by the channel subsystem.

   When prefetching has been specified in the ORB, the result of modifications to CCWs after START SUBCHANNEL has been issued 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 enables the program to place in main storage selected portions of a block of data.

   When, during an input operation, the program specifies data chaining to a location into 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 specify 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 execution 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 in 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 facility, 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. Refer to the appropriate System Library publication for the system model to determine the most reasonable minimum byte count that can be handled by the channel subsystem.

Command Chaining

During command chaining, the new CCW fetched by the channel subsystem speci-

LzLabs GmbH. Ex. 1016-379

fies 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 specify an S flag and if no unusual conditions are detected, the channel subsystem initiates the new operation. The presence of either condition 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: device end, device end and status modifier, device end and channel end, and device end and channel end and status modifier. In the former two cases, a channel end is signaled before device end, with all other status bits off. 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 on.

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 used, the first CCW associated with the operation specifies the operation to be executed, and the last CCW indicates whether another operation follows.

Programming Note

Command chaining makes it possible for the program to initiate transfer of multiple blocks of data by means of a single START SUBCHANNEL. It also permits a subchannel to be set up for execution of auxiliary functions, 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 selected portions of a block of information from an I/O device.

LzLabs GmbH. Ex. 1016-380

PROGRAM-CONTROLLED INTERRUPTION

The program-controlled-interruption (PCI) function permits the program to cause an I/O interruption during execution of an I/O operation. The function is controlled by the PCI flag in the CCW. The flag can be on either in the first CCW specified by the channel-program address or in a CCW fetched during chaining. Neither the PCI flag nor the associated interruption affects the execution of the current operation.

Whenever the PCI flag in the CCW is one, the subchannel becomes status-pending, with the intermediate-interruption condition indicated, and an I/O-interruption request is generated. When the first CCW associated with an operation contains the PCI flag, the subchannel becomes status-pending only after initial status is received indicating that the command has been accepted. The PCI flag in a CCW fetched on data chaining causes the status-pending condition to occur after all data designated by the preceding CCW has been transferred. The PCI flag on in a CCW fetched by command chaining may cause an interruption as early as immediately upon that CCW taking control of the operation. The time of the interruption, however, depends on the current activity in the system and may be delayed even if the subchannel is not masked. No predictable relation exists between the time the interruption due to the PCI flag occurs and the progress of data transfer to or from the area designated by the CCW. However, all the fields within the SCSW pertain to the same instant of time.

If chaining occurs before the interruption due to the PCI flag has taken place, the PCI condition is carried over to the new CCW. This carryover occurs both on data and command chaining and, in either case, the condition is propagated through the transfer-in-channel command. The PCI conditions are not stacked; that is, if another CCW is fetched with a PCI flag before the intermediate-status condition due to the PCI flag of the previous CCW has been cleared, only one interruption takes place.

An SCSW containing the PCI bit may be signaled by an interruption while the operation is still proceeding or upon the termination of the operation. All PCIs are presented to the system by TEST SUBCHANNEL even while the subchannel is active.

When an SCSW indicating intermediate-status-pending is stored by TEST SUBCHANNEL before the operation or chain of operations has been concluded (that is, when the activity-control field indicates subchannel-and-device-active or suspended), the command address is eight higher than the address of the CCW causing the intermediate-status-pending condition or eight higher than some CCW fetched subsequently. In this situation, the device-status field contains zeros, and count is unpredictable. If the channel subsystem has detected any actually or potentially unusual conditions, such as channel-data check, or protection check, by the time the SCSW is stored, the corresponding subchannel-status bit may be one. The unusual condition in the subchannel is not reset by TEST SUBCHANNEL unless the subchannel is also status-pending with primary status and, unless it is primary-status-pending, it may or may not be indicated again. If the unusual condition actually occurs, the condition is indicated when the subchannel subsequently becomes primary-status-pending upon the conclusion of the operation at the subchannel.

The primary-status bit set to one in the SCSW indicates that the operation or chain of operations has been concluded at the subchannel. In this case, the status-control field indicates the primary-interruption condition in addition to the intermediate condition. The SCSW in this case has its regular format with the PCI bit added.

When the interruption condition due to the PCI flag has been delayed until the operation at the device has been concluded, a single interruption occurs, and TEST SUBCHANNEL stores an SCSW containing both the PCI and the ending status.

The setting of the PCI flag is inspected in every CCW except for the format-0 CCW specifying transfer in channel, where it is ignored. In a format-1 CCW specifying transfer in channel, the PCI flag must be zero. The PCI flag is also 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 the PCI flag on may, if retried because of command retry, cause multiple PCI interruptions to occur. (See the section "Command Retry" later in this chapter.)

LzLabs GmbH. Ex. 1016-381

CCW INDIRECT DATA ADDRESSING

CCW indirect data addressing, a companion facility to dynamic address translation, provides assistance in translating data addresses for I/O operations. It permits a single channel-command word to control the transfer of data that spans noncontiguous pages in real main storage. The use of CCW indirect data addressing also allows the program to specify data addresses above 16M for both format-0 and format-1 CCWs.

CCW indirect data addressing is specified by a flag bit 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, called indirect-data-address words (IDAWs), each of which contains an absolute address designating a data area within a 2K-byte block of main storage.

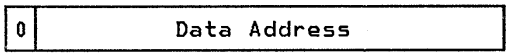
When the indirect-data-addressing bit in the CCW is one, the data-address field of the CCW specifies the location of the first IDAW to be used for data transfer for the command. Additional IDAWs, if needed for completing the data transfer for the CCW, are in successive locations in storage. The number of IDAWs required for a CCW is determined by the count field of the CCW and by the data address in the initial IDAW. When, for example, the CCW count field specifies 4K bytes and the first IDAW specifies a location in the middle of a 2K-byte block, three IDAWs are required.

Each IDAW is used for the transfer of up to 2K bytes. The IDAW specified 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 is reached. The control of data transfer is then passed to the next IDAW. The second and any subsequent IDAWs must specify, depending on the command, the first or last byte (for read backward) of a 2K-byte block. Thus, for read, write, control, sense ID, and sense commands, these IDAWs have zeros in bit positions 21-31. For a read-backward command, these IDAWs have ones in bit positions 21-31.

Except for the unique restrictions on the specification 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.

Any of the IDAWs pertaining to the current CCW or a prefetched CCW may be prefetched. An IDAW takes control of 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 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 0 with the transfer of the last byte of data for the previous IDAW for that CCW. Errors detected in prefetched IDAWs are not indicated until the IDAW takes control of the data transfer.

The format of the IDAW and the significance of its fields are as follows:



Bit 0 is reserved for future use and must be zero. Otherwise, a program-check condition may be recognized, as described below.

Bits 1-31 specify the location of the first byte to be used in the data transfer. In the first IDAW for a CCW, any location can be specified. For subsequent IDAWs, depending on the command, either the first or the last location of a 2K-byte block located on a 2K-byte boundary must be specified. For read, write, control, and sense commands, the beginning of the block must be specified, that is, bits 21-31 of the IDAW must be zeros. For a read-backward command, the end of the block must be specified (bits 21-31 all ones). Improper data-address specification causes the program-check condition to be generated and the operation to be terminated.

When the IDAW flag of the CCW is set to one and any of the following conditions occurs:

1. The address in the CCW does not designate the first IDAW on an integral word boundary,

2. The address in the CCW designated a storage location which is not available.

3. Access to the storage location specified by the address in the CCW is prohibited by protection, or

LzLabs GmbH. Ex. 1016-382

4. Bit 0 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.

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 in the ORB (bit 4 of word 1). The suspend function is signaled when the suspend-control bit is set to one in the ORB and a valid CCW containing the 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) if the CCW is fetched while data chaining (see the section "Data Chaining" earlier in this chapter, 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 the section "Channel-Command Word" earlier in this chapter, 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.

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 that 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 the section "RESUME SUBCHANNEL" in Chapter 14, "I/O Instructions" and the section "Start-Function and Resume-Function Execution" earlier in this chapter).

When a valid CCW that contains a valid S flag 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 to perform the resume function when the subchannel is next recognized to be resume-pending.

3. The accrued values of the measurement-block parameters, including the start-subchannel and sample count, are added to the accumulated values in the measurement block for the subchannel if the measurement-block-update mode is active and the subchannel is enabled for the mode. The start-subchannel count is the only parameter updated in the measurement block if the timing facility is not available for the subchannel. (See the section "Channel-Subsystem Monitoring Facilities" in Chapter 17, "I/O Support Functions" for more information.)

4. The subchannel is placed in the suspended state.

5. If the subchannel is not resume-pending at this point, the subchannel-suspended-interruption condition is recognized if the suppress-suspended-interruption bit of the ORB is zero; otherwise, the resume function is performed instead.

When a valid CCW that contains a valid S flag 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 a valid S flag and the resume-pending condition is not recognized, the suspend function is performed and causes the following actions to occur in the order given:

LzLabs GmbH. Ex. 1016-383

1. Channel-program execution is suspended prior to selection of the device.

2. The subchannel is set up to perform the resume function 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 START SUBCHANNEL count and the accrued function-pending time (a function of the setting of the timing-facility bit) are incremented in the measurement block for the subchannel.

4. The subchannel is placed in the suspended state.

5. If the subchannel is not resume-pending at this point, the subchannel-suspended-interruption condition is recognized if the suppress-suspended-interruption bit of the ORB is zero; otherwise, the resume function is performed instead.

When the first CCW of a channel program contains a valid S flag and the resume-pending condition is recognized, the resume function is performed instead of the suspend function.

Programming Notes

1. A suspended subchannel is busy for MODIFY SUBCHANNEL and START SUBCHANNEL because the start-function bit remains set to one in the subchannel while the subchannel is in the suspended state.

2. In certain situations, normal resumption of a suspended channel program may not be desired. Normal termination of the suspended channel-program execution may be accomplished by:

   a. Executing HALT SUBCHANNEL addressed to the subchannel.

   b. Modifying the CCWs in storage such that when channel-program execution is resumed, the command issued to the device is a control command with modifier bits all zeros (no-operation) and with no chain-command flag; and by issuing RESUME SUBCHANNEL.

3. If the suspended interruption is suppressed, the N 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 the PNOM and the N bit to be reset to zeros. The PNOM and the N bit are not modified when RESUME SUBCHANNEL is issued to a subchannel that is not in the suspended state.

COMMANDS

The figure "Command Codes" 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.

LzLabs GmbH. Ex. 1016-384

Name	Code	Flags
Write	MMMM MM01	CD CC SLI PCI IDA S
Read	MMMM MM10	CD CC SLI SKIP PCI IDA S
Read backward	MMMM 1100	CD CC SLI SKIP PCI IDA S
Control	MMMM MM11	CD CC SLI PCI IDA S
Sense	MMMM 0100	CD CC SLI SKIP PCI IDA S
Sense ID	1110 0100	CD CC SLI SKIP PCI IDA S
Transfer in channel	XXXX 1000	(See note below)

Explanation:

CC	Chain command
CD	Chain data
IDA	Indirect data addressing
M	Modifier bit
PCI	Program-controlled interruption
S	Suspend
SKIP	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.

Command Codes

All flags have individual significance, except that the CC and SLI flags are ignored when the CD flag is on. 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. When executing format-1 CCWs, it is unpredictable regarding whether or not the incorrect-length indication is suppressed. The PCI flag is ignored during initial program loading. All flags, except the PCI flag, are ignored when the S flag is one.

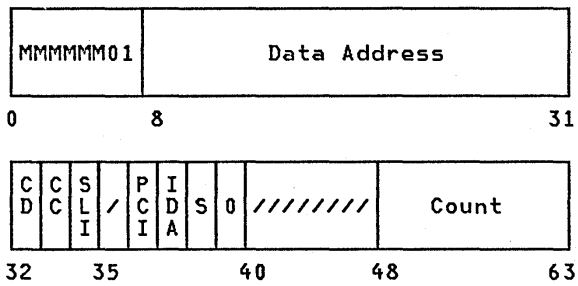
Each command is described below, with an illustration of its CCW formats.

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.

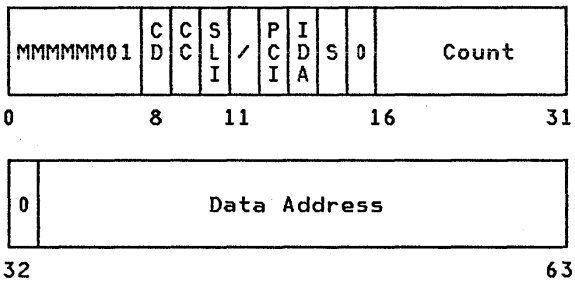
Write

Format 0



LzLabs GmbH. Ex. 1016-385

Format 1



A write operation is initiated at the I/O device, and the subchannel is set up to transfer data from main storage to the I/O device. Data in storage is fetched in an ascending order of addresses, starting with the address specified in the CCW.

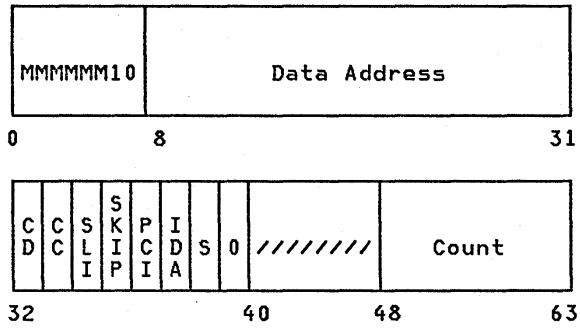
A CCW used in a write operation is inspected for the CD, CC, SLI, PCI, IDA, and S flags. The setting of the skip flag is ignored. Bit positions 0-5 of the CCW contain modifier bits.

Programming Note

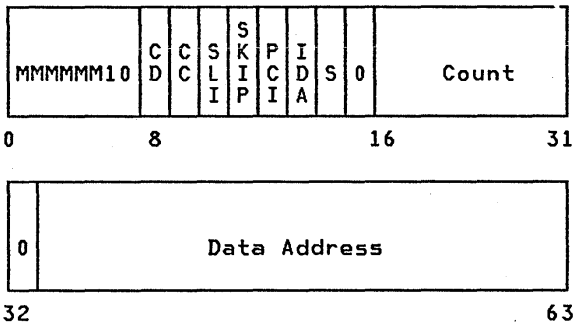
When writing on devices for which the block length is not defined, such as a magnetic-tape unit or an inquiry station, the amount of data written is controlled only by the count in the CCW. Every operation terminated under count control causes the incorrect-length indication, unless the indication is suppressed by the SLI flag.

Read

Format 0



Format 1



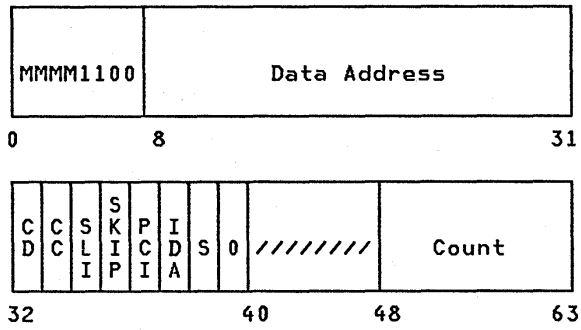
A read operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to main storage. For devices such as magnetic-tape units, disk storage, and card equipment, the bytes of data within a block are provided in the same sequence as written by means of a write command. Data in storage is placed in an ascending order of addresses, starting with the address specified in the CCW.

A read command code containing zeros for the six modifier bits is also called an initial-read command. This command is used by those devices that can perform the initial-program-loading function if the command is the first to be executed after a system-reset signal is received.

A CCW used in a read operation is inspected for every one of the seven flags -- CD, CC, SLI, SKIP, PCI, IDA, and S. Bit positions 0-5 of the CCW contain modifier bits.

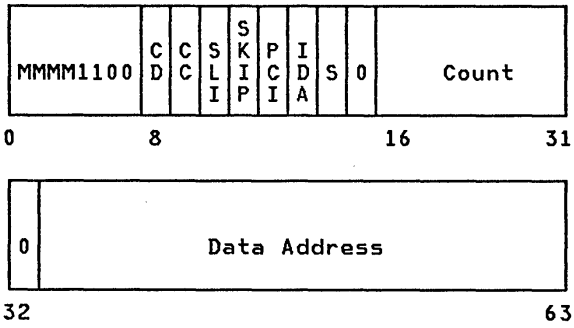
Read Backward

Format 0



LzLabs GmbH. Ex. 1016-386

Format 1

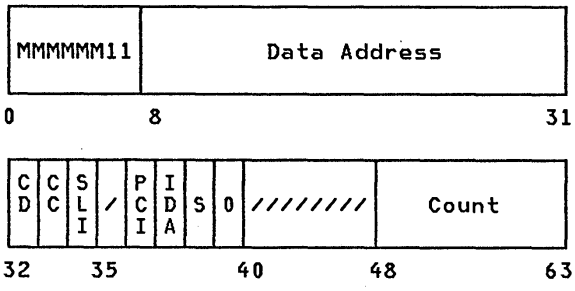


A read-backward operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to main storage. On magnetic-tape units, read backward causes reading to be performed with the tape moving backward. The bytes of data within a block are sent in a sequence opposite to that on writing. The bytes are placed in storage in a descending order of addresses, starting with the address specified in the CCW. The bits within an eight-bit byte are in the same order as sent to the device on writing.

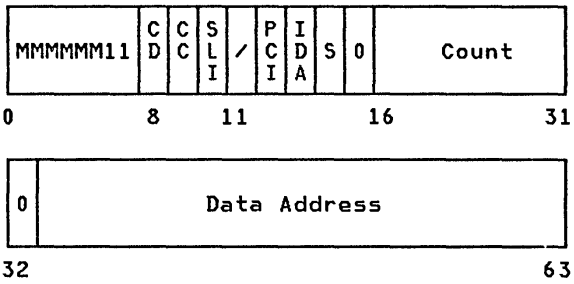
A CCW used in a read-backward operation is inspected for every one of the seven flags -- CD, CC, SLI, SKIP, PCI, IDA, and S. Bit positions 0-3 of the CCW contain modifier bits.

Control

Format 0



Format 1



A control operation is initiated at the I/O device, and the subchannel is set up to transfer data from main storage to the device. The device interprets the data as control information. The control information, if any, is fetched from storage in an ascending order of addresses, starting with the address specified in the CCW. A control command may be used to initiate at the device an I/O operation not involving transfer of data, such as backspacing or rewinding magnetic tape or positioning a disk-access mechanism.

For many control functions, the entire operation is specified by the modifier bits in the command code, and the function is performed over the channel path as an immediate operation (see the section "Immediate Conclusion of I/O Operations" later in this chapter). If the command code does not specify the entire control function, the data-address field of the CCW designates the location containing the required additional information. This control information may include an order code further specifying the operation to be performed or an address, such as the disk address for the seek function, and is transferred in response to requests by the device.

A control-command code containing zeros for the six modifier bits is defined as a no-operation. If the command is accepted, the no-operation order causes the addressed device to respond with channel end and device end without causing any action at the device. The order can be executed as an immediate operation, or the device can delay the status until after the initial selection sequence is completed. Other operations that can be initiated by means of the control command depend on the type of I/O device. These operations and their codes are specified in the System Library publication for the device.

A CCW used in a control operation is inspected for the CD, CC, SLI, PCI, IDA, and S flags. The setting of the skip flag is ignored. Bit positions 0-5 of the CCW contain modifier bits.

LzLabs GmbH. Ex. 1016-387

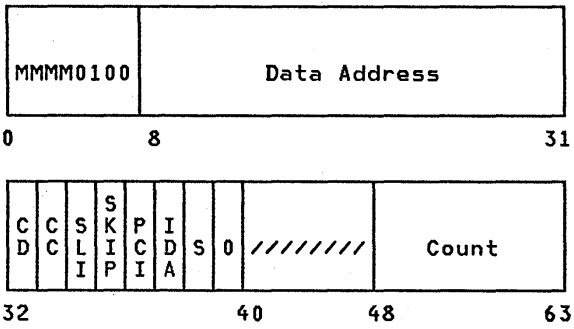
Programming Notes

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, and no unusual conditions are encountered subsequent to signaling the device to terminate data transfer, then the new operation is initiated upon receipt of device end from the device.

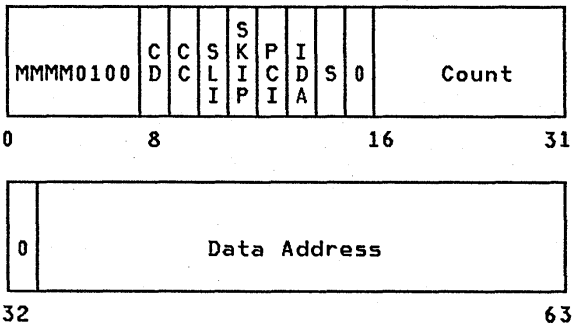
2. If format-0 CCWs are being used and the operation is executed as an immediate operation, incorrect length is not indicated, regardless of the setting of the SLI flag. If format-1 CCWs are being used, it is unpredictable whether incorrect length is indicated when a nonzero count is specified and the operation is executed as an immediate operation.

Sense

Format 0



Format 1



A sense operation is initiated at the I/O device, and the subchannel is set up to transfer sense data from the device to storage. The data is placed in storage in an ascending order of addresses, starting with the address specified in the CCW.

The basic sense command is specified when the modifier bits are all zeros. Data transferred during a basic sense operation provides information concerning both unusual conditions detected by the device and the status of the device. The information provided by the basic sense command is more detailed than that supplied by the device-status byte and may describe reasons for the unit-check indication. It may also indicate, for example, if the device is in the not-ready state, if the tape unit is in the file-protected state, or if magnetic tape is positioned beyond the end-of-tape mark.

The first six bits of the first sense-data byte (sense byte 0) are common to all I/O devices. The six bits, when set to ones, designate the following:

Bit	Designation
0	Command reject
1	Intervention required
2	Bus-out check
3	Equipment check
4	Data check
5	Overrun

The following is the meaning of the first six bits:

Command Reject: The device has detected a programming error. A command has been received which the device is not designed to execute, such as read backward issued to a direct-access-storage device, or which the device cannot execute because of its present state, such as write issued to a file-protected tape unit. Command reject is also indicated when the program issues an invalid sequence of commands, such as write to a direct-access-storage device without previously designating the data block.

Command reject may also be indicated when invalid data is transferred and the data is treated as an extension of the command. For example, command reject is indicated when an invalid seek argument is transferred to a direct-access storage device.

Intervention Required: The last operation could not be executed because of a condition requiring some type of intervention at the device. This bit set to one indi-

LzLabs GmbH. Ex. 1016-388

cates conditions such as an empty hopper in a card punch or the printer being out of paper. It is also set to one when the addressed device is in the not-ready state, is in test mode, or on some control units when the device is not provided on the control unit.

Bus-Out Check: The device has received a data byte or a command code with invalid parity over the channel path. During writing, bus-out check indicates that incorrect data may have been recorded at the device, but the condition does not cause the operation to be terminated prematurely unless the operation is such that an error precludes meaningful continuation of the operation. Parity errors on command codes and control information cause the operation to be immediately terminated and suppress checking for command-reject and intervention-required conditions.

Equipment Check: During the last operation, the device has detected equipment malfunctioning, such as an invalid card-hole count or a printer-buffer parity error.

Data Check: The device has detected a data error other than one included in bus-out check. Data check identifies errors associated with the recording medium and includes conditions such as reading an invalid card code or detecting invalid parity on data recorded on magnetic tape.

On an input operation, data check indicates that incorrect data may have been placed in main storage. The device forces correct parity on data sent to the channel subsystem. On writing, this condition indicates that incorrect data may have been recorded at the device. Unless the operation is of a type where the error precludes meaningful continuation, data errors on reading and writing do not cause the operation to be terminated prematurely.

Overrun: The overrun condition occurs when the channel subsystem fails to respond to the control unit in the anticipated time interval to a request for service from the I/O device. When the total activity initiated by the program exceeds the capability of the channel subsystem, an overrun may occur when data is transferred to or from a control unit that is either using the data-streaming facility or is not buffered. An overrun condition also may occur when the device receives the new command too late during command chaining. The data-streaming facility is described in the System Library publication IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI, GA22-6974. Refer to the System Library publication for the system model for information concerning the availability of the data-streaming facility for that model.

All information significant to the use of the device normally is provided in the first sense byte. Any bit positions following those used for programming information may contain diagnostic information, and the total number of sense bytes provided by the device for the basic sense command (command code 04 hex) may extend up to 32 bytes, as needed. The number and the meaning of the sense bytes extending beyond the first sense byte are peculiar to the type of I/O device and are specified in the System Library publication for the device.

The basic sense command initiates a sense operation on all devices and cannot cause the command-reject, intervention-required, data-check, or overrun bit to be set to one. If the control unit detects an equipment malfunction, or invalid parity on the sense-command code, the equipment-check or bus-out-check bit is set to one, and unit check is indicated in the device-status byte.

Devices that can provide special diagnostic sense information or that can be instructed to perform other special functions by use of the sense command may define modifier bits for the control of these functions. The special sense operations may be initiated by a unique combination of modifier bits (see the section "Sense ID" later in this chapter), or a group of codes may specify the same function. Any remaining sense-command codes may be considered invalid, thus causing the unit-check indication, or may cause the same action as the basic sense command, depending upon the type of device.

The sense information pertaining to the last I/O operation or device action may be reset any time after the completion of a sense command addressed to that device. Except for the no-operation command, any other command addressed to the device may be allowed to reset the sense information, provided that the busy bit is not included in the initial status. The sense information may also be changed as a result of asynchronous actions, for example, when the device changes from the not-ready to ready state.

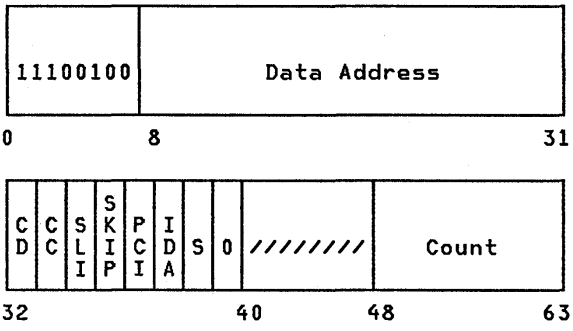
A CCW used in a sense operation is inspected for every one of the seven flags -- CD, CC, SLI, SKIP, PCI, IDA,

LzLabs GmbH. Ex. 1016-389

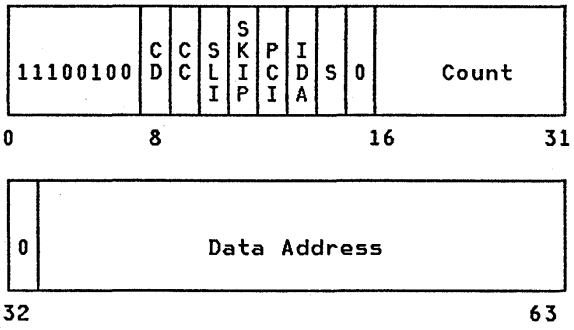
and S. Bit positions 0-3 of the CCW contain modifier bits.

Sense ID

Format 0



Format 1



Execution of the sense-ID command proceeds exactly as for a read command, except that the data is obtained from sensing indicators rather than from a record source. The data transferred can be up to seven bytes in length.

The control unit and I/O device may properly execute the sense-ID command, may execute the command as the basic sense command, or may reject the sense-ID command with unit-check status. Refer to the System Library publication for the control unit and device.

The sense-ID command does not initiate any operations other than the sensing of the type/model number. If the control unit and I/O device are available, then execution of the sense-ID command is performed even if the device is absent or not ready.

Basic sense data may be reset as a result of executing the sense-ID command.

The bytes sent in response to the sense-ID command are defined as follows:

Byte 0 FF hex
Byte 1 Control-unit type number
Byte 2 Control-unit type number
Byte 3 Control-unit model number
Byte 4 Device type number
Byte 5 Device type number
Byte 6 Device model number

All unused sense bytes are set to zeros.

Bytes 1 and 2 contain the four-decimal-digit control-unit type number that corresponds directly with the control-unit type number attached to the control unit.

Byte 3 contains the control-unit model number, if applicable. If not applicable, byte 3 is a byte of all zeros.

Bytes 4 and 5 contain the four-decimal-digit device type number that corresponds directly with the device type number attached to the I/O device.

Byte 6 contains the device model number, if applicable. If not applicable, byte 6 is a byte of all zeros.

Whenever a control unit is not separately addressable from the attached device or devices, the response to the sense-ID command is a concatenation of the control-unit type number and the device type number.

If a control unit can be addressed separately from the attached device or devices, then the sense data source for the sense-ID command is four bytes in length. (A subchannel is assigned for this type of control unit.) If a control unit is addressed, the response to the sense-ID command is bytes 0, 1, 2, and 3. If the device is addressed, the response to the sense-ID command is bytes 0, 4, 5, and 6.

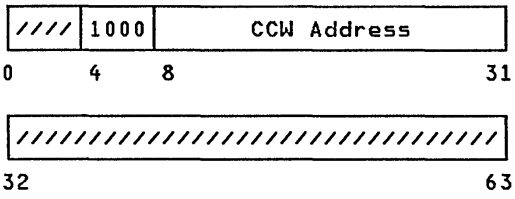
For communication controllers utilizing indirect addressing to end devices, and for cases where the control unit and device are not distinct, the sense data source consists solely of bytes 0, 1, 2, and 3.

A CCW used in a sense-ID operation is inspected for every flag -- CD, CC, SLI, SKIP, PCI, IDA, and S.

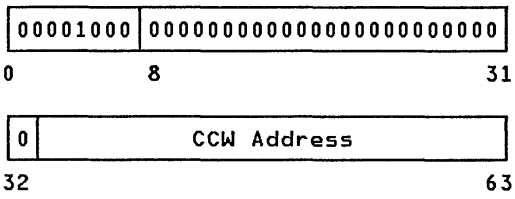
LzLabs GmbH. Ex. 1016-390

Transfer in Channel

Format 0



Format 1



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.

To address a CCW on integral boundaries for doublewords, a CCW specifying transfer in channel must contain zeros in the three low-order bit positions of the data-address field. Furthermore, a CCW specifying a 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 specified in transfer in channel, the program-check condition is generated. When the transfer-in-channel command designates a CCW in 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, whereas 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, 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 the device end is presented alone, the previous command is reissued. 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 (see the section "Status Modifier" in Chapter 16, "I/O Interruptions"). 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 channel-end, unit-check, and status-modifier status indications, along with any other appropriate status.

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 DURING INITIATION

After the addressed subchannel has been determined to be in a state where START SUBCHANNEL can be executed, certain tests are performed on the validity of the information specified by the program

LzLabs GmbH. Ex. 1016-391

and on the logical availability of the associated I/O 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 primary-, secondary-, and alert-status-pending. 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 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 independently 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 execution of the start function, an interruption condition is generated by the channel subsystem and placed in 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 function of indications provided in the ORB and CCW and status presented by the device. A 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 the section "Program-Controlled Interruption" later in this chapter).

Whenever command chaining is specified after an immediate operation and no unusual conditions have been detected during the execution, 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. In the case of format-0 CCWs, incorrect length is not indicated to the program, and command chaining is performed when so specified. In the case of format-1 CCWs, it is unpredictable whether incorrect length and command chaining are under control of the chain-command and SLI flags if a nonzero count was specified in the CCW.

Programming Note

I/O operations for which the entire operation is specified in the command code may be executed as immediate operations. Whether the command is executed as an immediate operation depends on the operation and type of device and is specified in the System Library publication for the device.

LzLabs GmbH. Ex. 1016-392

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 the channel subsystem detects an equipment malfunction, the operation is concluded by CLEAR SUBCHANNEL or HALT SUBCHANNEL, or primary-interruption status (usually the channel-end signal) is received from the device. When no command chaining or command retry is specified or when chaining is suppressed because of unusual conditions, primary-interruption status causes the operation at the subchannel to be terminated and an interruption condition to be generated. The status bits in the associated SCSW indicate the primary status and the unusual conditions, if any. The device can signal primary-interruption status at any time after the initiation of the I/O operation, and the signal 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, CLEAR SUBCHANNEL, HALT SUBCHANNEL, and RESET CHANNEL PATH, the channel subsystem signals the device to conclude execution of an I/O operation during data transfer whenever any of the following conditions occurs:

- The storage areas specified 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 a 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 generates the channel-end condition.

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 provide the channel-end signal 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 concluded before the recording medium has been advanced. However, since the operation has been initiated at the I/O device, the device generates a channel-end primary-interruption condition. Subsequently, the device also generates a device-end secondary interruption condition. Whether a block at the device is advanced when no data is transferred depends on the type of device and is specified in the System Library publication for the device.

When command chaining takes place, the subchannel is in the subchannel-and-device-active states from the time the first I/O operation is initiated at the device until the device signals the channel-end condition of the last I/O operation of the chain. The subchannel remains in the device-active state until the device signals the device-end condition of 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 signal for the operation is received, the chain is ended as if the operation during which the condition occurred were the last operation of the chain. The device-end signal subsequently is processed as a secondary-interruption condition. When the device signals unit check or unit exception with control-unit end or device end as

LzLabs GmbH. Ex. 1016-393

the primary-interruption condition, 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 as a function of the PCI flag specified in the CCW. The channel-end indication which was presented to the channel subsystem previously when command chaining was signaled is not made available to the program.

CHANNEL-PATH-RESET-FUNCTION EXECUTION

Subsequent to the execution of RESET CHANNEL PATH, the channel-path-reset function is performed. Execution of the function consists in: (1) issuing the reset signal on the designated channel path and (2) causing a channel report to be made pending, indicating completion of the channel-path-reset function.

Channel-Path-Reset-Function Signaling: Subsequent to channel-path selection, the channel subsystem issues the reset signal on the specified channel path. As part of this operation, the following actions are taken:

1. All internal indications relative to prior control-unit busy, device busy, and allegiance conditions for the designated channel path are reset. These indications are reset in 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 the discussion of the reset signal in the section "I/O-System Reset" in Chapter 17, "I/O Support Functions," for a detailed description) or, because of a malfunction, the reset signal could not be issued, the corresponding bit in the path-available mask (PAM) is set to zero in each applicable subchannel.

3. If an I/O operation is in progress at the device and the device is actively communicating over the channel path in the performance of that I/O operation when the reset signal is received on that path, the I/O operation is reset, and the control unit and device immediately terminate current communication with the channel subsystem. (But see programming note 2 below.)

4. If an I/O operation is in progress in multipath mode at the device and the device is not currently communicating over the channel path in performance of that I/O operation when the reset signal is received, then whether the I/O operation is reset depends on whether there is another channel path that 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 path over which the system reset is received is removed from the current set of paths that form the multipath group. If the channel path over which the reset signal is received is either the only channel path of a multipath group or the device is operating in 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 currently being used is the same channel path as specified when execution of the channel-path-reset function is performed, the subchannel may or may not accurately reflect the progress of the I/O operation up to that point in time. The subchannel remains in the state that exists at the time channel-path reset 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, an attempt has been made by the channel subsystem to issue the reset signal, or the channel subsystem has determined that the reset signal cannot be issued, the reset function is completed. (See the section "Reset Signal" in Chapter 17, "I/O Support Functions.")

As a result of the channel-path-reset function being performed, a channel report is made pending (see the section "Channel-Subsystem Recovery" in Chapter 17, "I/O Support Functions") 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 corresponding bit in the PAM for all subchannels having access to that path was already zero, the channel report indicates that the channel path has been initialized

LzLabs GmbH. Ex. 1016-394

but is not physically available for use. If, because of a malfunction or because the specified channel path is not in the configuration, the system-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 multipath mode when channel-path reset is performed on a channel path of the multipath group, it is possible for the I/O operation to be continued on a remaining 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. In such a situation, the device end is not made available to the program and the reselection of the device by the channel subsystem may be treated by the device as the beginning of a new channel program. This possibility can be avoided by issuing CLEAR SUBCHANNEL or HALT SUBCHANNEL to the affected subchannels prior to issuing RESET CHANNEL PATH.

3. Execution 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 in a control unit or a malfunction in the physical path to the control unit.

LzLabs GmbH. Ex. 1016-395

Interruption Conditions 16-2
Unsolicited Interruption Condition 16-3
Solicited Interruption Condition 16-3
Intermediate Interruption Condition 16-4
Primary Interruption Condition 16-4
Secondary Interruption Condition 16-4
Alert Interruption Condition 16-5
Priority of Interruptions 16-5
Interruption Action 16-6
Interruption-Response Block (IRB) 16-6
Subchannel-Status Word (SCSW) 16-7
Extended-Status Word 16-8
Extended-Control Word 16-8
Subchannel-Status-Word Contents 16-8
Subchannel Key 16-8
Suspend Control (S) 16-8
Extended-Status-Word Format (L) 16-8
Deferred-Condition-Code Contents 16-9
Format (F) 16-11
Prefetch (P) 16-11
Initial-Status-Interruption Request (I) 16-12
Address-Limit-Checking Control (A) 16-12
Suppress-Suspended Interruption (U) 16-12
Subchannel-Control-Field Contents 16-12
Zero Condition Code (Z) 16-12
Extended Control (E) 16-12
Path Not Operational (N) 16-13
Function Control 16-13
Activity Control 16-14
Status Control 16-16
Device-Status Conditions 16-18
Attention 16-19
Status Modifier 16-19
Control-Unit End 16-19
Busy 16-21
Channel End 16-21
Device End 16-21
Unit Check 16-22
Unit Exception 16-23
Subchannel-Status Conditions 16-23
Program-Controlled Interruption 16-24
Incorrect Length 16-24
Program Check 16-24
Protection Check 16-25
Channel-Data Check 16-26
Channel-Control Check 16-26
Interface-Control Check 16-27
Chaining Check 16-27
CCW-Address-Field Contents 16-28
Count-Field Contents 16-33
Extended-Status-Word Contents 16-35
Extended-Status Format 0 16-36
Extended-Status Format 1 16-40
Extended-Status Format 2 16-40
Extended-Status Format 3 16-41
Extended-Control Word 16-43

When an I/O operation or sequence of I/O operations initiated by 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 TEST PENDING INTERRUPTION. (During certain abnormal situations, these conditions can be brought to the attention of the program via a machine-check interruption. See the section "Channel-Subsystem Recovery" in Chapter 17, "I/O Support Functions," for details.) The

LzLabs GmbH. Ex. 1016-397

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 in execution 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 executing 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 executing a clear, halt, or channel-path-reset function as a result of the program issuing 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.

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 TEST PENDING INTERRUPTION, TEST SUBCHANNEL, CLEAR SUBCHANNEL, or 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 TEST PENDING INTERRUPTION. In either case, the subchannel remains status-pending until the associated interruption condition is cleared when TEST SUBCHANNEL is executed or the subchannel is reset.

An I/O-interruption condition is normally cleared by TEST SUBCHANNEL. If TEST SUBCHANNEL is issued to 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 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 START SUBCHANNEL
- A valid suspend flag in the first CCW fetched that initiates channel-program execution for either a 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 none of the device-status bits are on.

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

LzLabs GmbH. Ex. 1016-398

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 be set status-pending.

An interruption condition caused by the device may be accompanied by multiple device-status conditions. Further, 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 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 which is unrelated to the execution 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.

Solicited Interruption Condition

A solicited interruption condition is any interruption condition generated as a direct consequence of executing or attempting to execute 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. The figure "Interruption Condition for Status-Control-Bit Combinations" describes the interruption condition that results from any combination of bits in the status-control field of the SCSW.

LzLabs GmbH. Ex. 1016-399

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	I	S	S	S	S	S	S	I	S	S

Explanation:

0 Indicates the bit stored as zero

1 Indicates the bit stored as one

E Unsolicited or solicited interruption condition

S Solicited interruption condition

I Combination invalid

Interruption Condition for Status-Control-Bit Combinations

INTERMEDIATE INTERRUPTION CONDITION

An intermediate interruption condition is a solicited interruption condition that indicates that an event for which the program had previously requested notification has occurred. 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, execution or suspension of the I/O operation continues, unaffected by the setting of the intermediate-status bit.

An intermediate interruption condition can occur only when the subchannel is in one of the following states:

1. Subchannel-active
2. Status-pending with primary status alone
3. Status-pending with primary status together with alert status and/or secondary status
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 execution of the start function is completed at the subchannel. A primary interruption condition is described by the SCSW stored as a result of executing TEST SUBCHANNEL while the subchannel is primary-status-pending. 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, 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 issuing TEST SUBCHANNEL, the subchannel accepts the START SUBCHANNEL instruction. (See the section "START SUBCHANNEL" in Chapter 14, "I/O Instructions.")

SECONDARY INTERRUPTION CONDITION

A secondary interruption condition is a solicited interruption condition that normally indicates the termination of an I/O operation at the device. A secondary interruption is generated by the channel subsystem when the start function is terminated because a solicited alert condition is recognized prior to initiating the first I/O operation at the device. A secondary interruption

LzLabs GmbH. Ex. 1016-400

condition is described by the SCSW stored as a result of executing TEST SUBCHANNEL while the subchannel is secondary-status-pending. Once the device has signaled the secondary interruption condition to the channel subsystem, execution of 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 execution of a halt, resume, or start function. An alert interruption condition is described by the SCSW stored as a result of executing TEST SUBCHANNEL while the subchannel is alert-status-pending. 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. Whenever the subchannel is idle and zero status is presented by the device, the status is discarded.

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 interruption-subclass code, and another establishes within a given CPU the priority among requests from subchannels with different interruption-subclass codes. The channel subsystem requests an I/O interruption only after it has established priority among requests from its subchannels. The conditions responsible for the requests are preserved at the subchannels until cleared by a CPU issuing TEST SUBCHANNEL or CLEAR SUBCHANNEL or until I/O-system reset is performed.

Assignment of priority by the channel subsystem to requests for I/O interruptions is in the order that the need for interruption requests 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 I/O interface channel-path type, 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 I/O interface is not related to its device address.

The channel subsystem requests an I/O interruption by logically enqueuing the corresponding subchannel on the appropriate I/O-interruption-subclass queue, as specified by the I/O-interruption-subclass code contained in the subchannel.

Among subchannels requesting I/O interruptions with the same interruption-subclass code (that is, that are in the same interruption-subclass queue), priority for interruption is based on position within the queue. The highest priority is implicitly assigned to the subchannel that is logically enqueued first. The subchannel that is logically enqueued first is determined by the subchannel that first identifies itself with an interruption condition.

When the highest-priority interruption request in a given interruption-request queue is accepted by a CPU, the corresponding subchannel is logically removed from that subclass queue so that the interruption request is not re-presented.

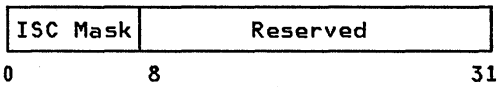
The priority of interruption handling by a CPU can be modified by TEST SUBCHANNEL and CLEAR SUBCHANNEL. When either of these instructions is issued to a subchannel that has an interruption request pending, the subchannel is logically removed from the appropriate interruption-subclass queue, regardless of its priority within the queue. The relative priority of the remaining subchannels logically in the queue is unchanged.

The assignment of priority among requests for interruption from subchannels logically enqueued on different interruption-subclass queues is determined by the CPU according to the interruption-subclass-code numerical value (with zero having highest priority), in conjunction with the I/O-interruption-subclass-enable-mask settings in control register 6. The numerical value of the interruption-subclass code is directly related to the bit position in the I/O-interruption-subclass-enable-mask register of a CPU. If in any CPU an interruption-subclass-enable-mask bit is zero, then all subchannels having an 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 the lowest-numbered interruption-subclass queue that is not masked off by a corresponding bit in control register 6 of that CPU.

LzLabs GmbH. Ex. 1016-401

Programming Notes

1. The I/O-interruption-subclass-enable masks are in control register 6, which has the following format:



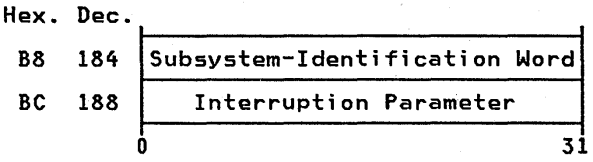
2. Control register 6 is set to all zeros during initial CPU reset.

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 the section "Point of Interruption" in Chapter 5, "Program Execution"). If the channel subsystem established the priority among requests for interruption from subchannels while the CPU was disabled for I/O interruptions, the interruption occurs immediately after 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 has established the priority among requests for interruption from subchannels while the I/O-interruption subclass-mask bit is zero for each subchannel which is status-pending, the interruption occurs immediately after completion of the instruction which 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 completion of the instruction enabling the CPU. A delay can occur regardless of how long the interruption condition has existed at the subchannel.

The interruption causes the current PSW to be stored as the old PSW at real location 56 and causes the I/O-interruption code associated with the interruption to be stored at real locations 184-191 of the CPU allowing the interruption. Subsequently, a new PSW is loaded from real location 120, 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:



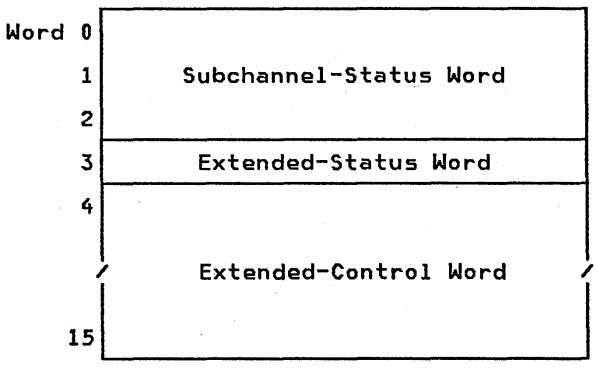
At the time a CPU is taking an interruption from an interruption-subclass queue, no other CPU can take an interruption from the same queue. However, other CPUs may take interruptions from other interruption-subclass queues. After interruption information has been taken from an interruption-subclass queue, a previously contending CPU is permitted to take interruption information from the same queue.

Programming Note

The I/O-interruption-subclass code for all subchannels is set to zero during the performance of I/O-system reset. It may be set to any of values 0-7 by using MODIFY SUBCHANNEL. (The operation of the instruction is described in the section "MODIFY SUBCHANNEL" in Chapter 14, "I/O Instructions.")

INTERRUPTION-RESPONSE BLOCK (IRB)

The interruption-response block (IRB) is the operand of TEST SUBCHANNEL. The two low-order bits of the IRB address are zeros, specifying 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. The format of the IRB is as follows:



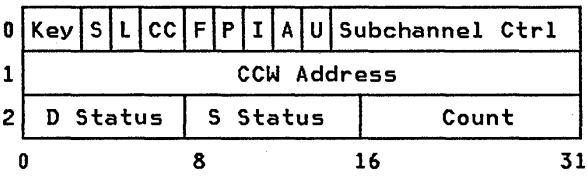
LzLabs GmbH. Ex. 1016-402

The length of the subchannel-status and extended-status words is 12 bytes and four bytes, respectively. The length of the extended-control word is 48 bytes. When the extended-control bit (bit 14, word 0) of the SCSW is zero, words 8-15 of the extended-control word may or may not be stored.

Subchannel-Status Word (SCSW)

The subchannel-status word (SCSW) provides to the program indications describing the status of a subchannel and its associated device. If execution 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 issued to an operational subchannel. The SCSW is placed in words 0-2 of the IRB that is specified as the TEST SUBCHANNEL operand. When STORE SUBCHANNEL is issued, the SCSW is stored in words 7-9 of the subchannel-information block (described in the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions."). The SCSW is described in detail in the section "Subchannel-Status-Word Contents." The format of the SCSW is as follows:



The SCSW fields are described briefly below. For a detailed description of the contents of each field, see the section "Subchannel-Status-Word Contents" later in this chapter.

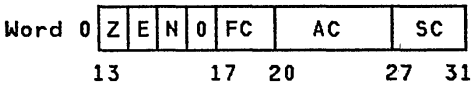
The following fields, when valid, contain the same information that was previously passed to the subchannel in the ORB when the start function was accepted:

Bits	Designation
0-3	Subchannel key
4	Suspend control (S)
8	Format (F)
9	Prefetch (P)
10	Initial-status interruption request (I)
11	Address-limit-checking control (A)
12	Suppress-suspended interruption (U)

Extended-Status-Word Format (L): Bit 5 of word 0 indicates whether logout information has been stored in the extended-status word.

Deferred Condition Code (CC): When the start function is specified, bits 6-7 of word 0 indicate the general reason that the subchannel was status-pending when TEST SUBCHANNEL was issued.

SUBCHANNEL CONTROL: Bits 13-31 of word 0 contain information of a supervisory nature relating to conditions at the subchannel. The subchannel-control field has the following format:



Zero Condition Code (Z): Bit 13 of word 0 may indicate whether the channel program has been initiated or resumed at the device.

Extended Control (E): Bit 14 of word 0 may indicate that additional model-dependent information has been stored in the IRB.

Path Not-Operational (N): Bit 15 of word 0 indicates whether the PNOM contains nonzero bits.

Function Control (FC): Bits 17-19 of word 0 identify the I/O function in progress at the subchannel.

Bit	Designation
17	Start function
18	Halt function
19	Clear function

Activity Control (AC): Bits 20-26 of word 0 identify the current activity associated with an I/O function.

Bit	Designation
20	Resume-pending
21	Start-pending
22	Halt-pending
23	Clear-pending
24	Subchannel-active
25	Device-active
26	Suspended

Status Control (SC): Bits 27-31 of word 0 identify and explain the existence of combinations of subchannel control and status bits described by the zero-condition-code bit (word 0, bit 13), the suspended bit (word 0, bit 26), the device-status field (word 2, bits 0-7), and the subchannel-status field (word 2, bits 8-15).

LzLabs GmbH. Ex. 1016-403

Bit	Designation
27	Alert status
28	Intermediate status
29	Primary status
30	Secondary status
31	Status pending

CCW Address: 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 by TEST SUBCHANNEL.

Device (D) Status: Bits 0-7 of word 2 identify the conditions in the device that caused the subchannel to become status-pending. Each of the eight bits represents one condition, as follows:

Bit	Designation
0	Attention
1	Status modifier
2	Control-unit end
3	Busy
4	Channel end
5	Device end
6	Unit check
7	Unit exception

Subchannel (S) Status: Bits 8-15 of word 2 identify the conditions in the subchannel that caused the subchannel to become status-pending. Each of the eight bits represents one condition, as follows:

Bit	Designation
8	Program-controlled interruption
9	Incorrect length
10	Program check
11	Protection check
12	Channel-data check
13	Channel-control check
14	Interface-control check
15	Chaining check

Count: Bits 16-31 of word 2 contain the residual count.

Extended-Status Word

The extended-status word (ESW) provides additional information to the program about the subchannel and its associated device. See the section "Extended-Status-Word Contents" later in this chapter for a detailed description of the contents of the ESW.

Extended-Control Word

The extended-control word (ECW) provides additional information to the program describing conditions that may exist at the channel subsystem, subchannel, or device. The setting of the extended-status-word-format (L) bit and the extended-control (E) bit, bits 5 and 14 of word 0, respectively, specify the contents of the extended-control word. See the section "Extended-Control-Word Contents" later in this chapter for a detailed description of the contents of the ECW.

SUBCHANNEL-STATUS-WORD CONTENTS

The contents of the subchannel-status word (SCSW) depend on the state of the subchannel when the SCSW was stored. The fields of the SCSW may contain information pertaining to the last operation, may contain information unrelated to the execution of an operation, may contain zeros, or may contain a value of no meaning, depending on the state of the subchannel and device. The following descriptions indicate when an SCSW field contains valid information.

Subchannel Key

Bits 0-3 of word 0 form the access key used during execution of the associated start function. These bits are identical with the key specified in the ORB (bits 0-3, word 1). The subchannel key is valid when the start-function bit (bit 17 of word 0) is one.

Suspend Control (S)

Bit 4 of word 0 controls execution of the suspend function for the channel program identified by the ORB. The setting of bit 4 applies to all channel-command words (CCWs) of a channel program. When bit 4 is one in the ORB, suspension of the channel-program execution occurs when a valid S flag is recognized in a CCW by the channel subsystem. When bit 4 is zero, the suspend-control function is not specified, and the presence of an S flag in any CCW causes the channel subsystem to recognize a program-check condition.

The suspend-control bit is valid when the start-function bit (bit 17 of word 0) is one.

Extended-Status-Word Format (L)

Bit 5 of word 0, when one, indicates that logout information has been stored

LzLabs GmbH. Ex. 1016-404

in the extended-status word. Logout information is provided whenever the following conditions are detected:

Channel-data check
Channel-control check
Interface-control check
Measurement-block-program check
Measurement-block-data check
Measurement-block-protection check

The extended-status-word bit is valid whenever the subchannel is status-pending alone or is status--pending with any combination of primary, secondary, and alert status. This bit remains valid until the interruption condition is cleared by TEST SUBCHANNEL.

DEFERRED-CONDITION-CODE CONTENTS

When valid, bits 6-7 of word 0 indicate the general reason that the subchannel was status-pending when TEST SUBCHANNEL was issued. The deferred-condition-code field is only valid when a start function is specified (bit 17 of word 0 is one) and the subchannel is either status-pending alone or status-pending with any combination of primary, secondary, and alert status. The meaning of these bits for each possible setting when the subchannel is status-pending is given in the figure "Deferred-Condition-Code Meaning for Status-Pending Subchannel."

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. The deferred condition code is valid 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.

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 I/O 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 channel program implied by the preceding START SUBCHANNEL or RESUME SUBCHANNEL. If the intermediate-status, primary-status, or primary-and-secondary-status bits are set to one, the status is related to execution of the channel program implied by the preceding START SUBCHANNEL or RESUME SUBCHANNEL. (See the section "Immediate Conclusion of I/O Operations" in Chapter 15, "Basic I/O Functions.") If the secondary-status bit is one and the primary-status bit is zero, the status present is related to the channel program implied by the START SUBCHANNEL or RESUME SUBCHANNEL preceding the last executed START SUBCHANNEL.

Deferred Condition Code 2: This code is not set and is reserved for future use.

Deferred Condition Code 3: An attempted selection of a device has occurred, and a path-not-operational condition has been detected on all of the channel 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 the section "I/O Addressing" in Chapter 13, "I/O Overview.")

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 the figure "Deferred-Condition-Code Meaning for Status-Pending Subchannel.")

Programming Notes

1. If, during execution of a start function, the 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 device being selected as one of its devices (for example, access mechanism 7 on the IBM 3830 Storage Control that has only access mechanisms 0-3 installed), the control unit, depending upon the model, either fails to recognize the address of the device or considers the 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 the discussion of path-operational mask in the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions.") In the latter case, the not-ready condition is indicated when the control unit

LzLabs GmbH. Ex. 1016-405

responds to the selection and indicates unit check whenever the not-ready state precludes successful initiation of the operation at the device. In this case, unit-check status is indicated in the SCSW, the subchannel becomes primary-, secondary-, and alert-status-pending, and deferred condition code 1 is set. (See the section "Unit Check" in this chapter.) 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 indicates intermediate status pending or intermediate and alert status pending when HALT SUBCHANNEL has been issued to a subchannel that is suspended and intermediate-status-pending. The alert-status condition, if any, is set when logout information is generated as a result of attempting to issue the halt signal to the device.

LzLabs GmbH. Ex. 1016-406

Bit 6	Bit 7	Status Control[1]	Meaning
0	0	A I P S X A I P - X A - P S X A - P - X - I P S X - I P - X - - P S X - - P - X	Normal I/O interruption.
0	1	A I P S X A I P - X A I - - X A - P S X A - P - X A - - S X A - - - X - I P S X - I P - X - I - - X[3] - - P S X - - P - X - - - S X[2] - - - - X[2-3]	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 - I P S X	The device is not operational via any available path or, if a dedicated-allegiance condition exists, the device is not operational via the path to which dedicated allegiance is owed.

Explanation:

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 start-pending.
3 The condition is encountered after the execution of HALT SUBCHANNEL when the subchannel is currently suspended.
A Alert status.
I Intermediate status.
P Primary status.
S Secondary status.
X Status pending.
- Bit is zero.

Deferred-Condition-Code Meaning for Status-Pending Subchannel

Format (F)

Bit 8 of word 0 specifies the format of the CCWs associated with an I/O operation. The format bit is valid when the start-function bit (bit 17 of word 0) is one. If bit 8 of word 0 is zero, format-0 CCWs are specified. If it is one, format-1 CCWs are specified. (See the section "Channel-Command Word" in Chapter 15, "Basic I/O Functions" for the description of the two CCW formats.)

Prefetch (P)

Bit 9 of word 0 indicates whether or not unlimited prefetching of CCWs is allowed. The prefetch bit is valid when the start-function bit (bit 17 of word 0) is one. If bit 9 is zero, prefetching of one CCW is allowed during output-data-chaining operations and is not allowed during any other operations. If bit 9 is one, unlimited prefetching of CCWs is allowed.

LzLabs GmbH. Ex. 1016-407

Initial-Status-Interruption Request (I)

Bit 10 of word 0, when one, indicates that the channel subsystem has been requested by the program to verify device acceptance of the first command associated with a channel program (see the description of bit 10, word 1, in the section "Operation-Request Block" in Chapter 15, "Basic I/O Functions"). This may occur as a result of initiating execution of either a start or resume function. The initial-status-interruption-request bit is valid when the start-function bit (bit 17 of word 0) is one.

Address-Limit-Checking Control (A)

Bit 11 of word 0, when one, indicates the channel subsystem has been requested by the program to perform address-limit checking, subject to the setting of the limit-mode bits at the subchannel (see the description of bit 11, word 1, in the section "Operation-Request Block" in Chapter 15, "Basic I/O Functions"). The address-limit-checking-control bit is valid when the start-function bit (bit 17 of word 0) is one.

Suppress-Suspended Interruption (U)

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 the description of bit 12, word 1, in the section "Operation-Request Block" in Chapter 15, "Basic I/O Functions"). When bit 12 is zero, the channel subsystem generates an intermediate-interruption condition whenever the subchannel is suspended during execution of the associated channel program. The suppress-suspended-interruption bit is valid when the start-function bit (bit 17 of word 0) is one.

SUBCHANNEL-CONTROL-FIELD CONTENTS

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 conditions have been encountered, subsequent to the setting of a condition code 0 for a START SUBCHANNEL or RESUME SUBCHANNEL, which confirm that the channel program has been initiated at the device. The I/O operation is considered to be initiated when the subchannel-active bit is set to one. (See the section "Activity Control" later in this chapter.) The setting of the Z bit is under the control of the I bit (bit 10, word 1) of the ORB that was the operand of the initiating START SUBCHANNEL execution (see the section "Operation-Request Block" in Chapter 15, "Basic I/O Functions"). If the I bit is one in the ORB, at the time the ORB is passed to the subchannel by execution of START SUBCHANNEL, then when the subchannel becomes active, the Z bit is set to one, and the subchannel is made intermediate-status-pending. The validity of the Z bit is maintained from the time the subchannel becomes active until the intermediate-interruption condition is cleared. If the I bit is zero in the ORB, the Z bit remains zero, and the subchannel is not made status-pending with intermediate status when the subchannel-active bit is set to one.

Extended Control (E)

Bit 14 of word 0, in conjunction with bit 5 of word 0, describes the contents of the extended-control word (ECW). When bit 17 is zero, regardless of the setting of bit 5, words 0-3 of the ECW are stored as zeros and words 4-11 of the ECW if stored, are undefined. The setting of bits 14 and 5 to one and zero, respectively, is currently reserved; therefore, words 0-11 of the ECW are currently undefined. If both bits 14 and 5 are one, zeros are stored in words 0-3 of the ECW, and model-dependent information is stored in words 4-11 of the ECW (see the figure "Information in Extended-Control Word"). The E bit is valid whenever the subchannel is status-pending or is status-pending with any combination of primary, secondary, or alert status.

Programming Note

During execution of TEST SUBCHANNEL, the storing of words 4-11 of the ECW is a model-dependent function subject to the settings of bits 5 and 14 as described above. Therefore, the program should always provide sufficient storage to accommodate the storing of a 64-byte IRB.

LzLabs GmbH. Ex. 1016-408

Path Not Operational (N)

Bit 15 of word 0, when one, indicates that the channel subsystem encountered one or more not-operational conditions, on paths whose corresponding POM bits were ones, while attempting to select the device in order to perform a clear, halt, resume, or start function. The path or paths over which the device was found to be not operational are indicated by corresponding bits set to ones in the PNOM. The presence of nonzero bits in the PNOM is signaled to the program by storing the path-not-operational (N) bit as one in the SCSW.

When the channel subsystem performs an alternate-path selection because of a not-operational-path condition, and the corresponding POM bit is one; the corresponding bit in the PNOM is set to one. The channel subsystem then repeats the selection-attempt process by using another available path, if any.

If device selection results in a not-operational-path condition and the corresponding POM bit is zero, then the corresponding PNOM bit is not set to one, the N condition is not indicated, and the channel subsystem attempts selection by another available path, if any.

Notes

1. A not-operational-path condition does not imply a malfunctioning path. A malfunctioning 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 continues to be recognized until the subchannel becomes status-pending with primary status or becomes suspended and 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

The function-control field indicates the basic I/O functions that are specified for 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 valid at an installed subchannel whenever the device-number-valid bit is one (see the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions"). 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 is specified for the subchannel. A start function has been requested by issuing START SUBCHANNEL to the subchannel. This bit is set to one when condition code 0 is set for START SUBCHANNEL. Bit 17 is set to zero at the subchannel when any combination of alert, primary, or secondary status-pending conditions or the status-pending condition alone is cleared by TEST SUBCHANNEL. Bit 17 is also set to zero during the execution of CLEAR SUBCHANNEL.

Halt Function (Bit 18): When one, bit 18 indicates that a halt function is specified for the subchannel. A halt function has been requested by issuing HALT SUBCHANNEL to the subchannel. In normal operations, this function is indicated together with bit 17; that is, there is an I/O operation either pending or in progress which is to be halted. This bit is set to one when condition code 0 is set for HALT SUBCHANNEL. Bit 18 is reset to zero when the next status-pending condition occurs and is cleared by TEST SUBCHANNEL. Depending on the state of the subchannel when HALT SUBCHANNEL is executed, the next status-pending condition can be either status-pending alone or status-pending with at least primary and/or secondary status indicated. Bit 18 is also set to zero during the execution of CLEAR SUBCHANNEL.

Clear Function (Bit 19): When one, bit 19 indicates that a clear function is specified for the subchannel. A clear function has been requested by issuing CLEAR SUBCHANNEL to the subchannel. This bit is set to one when condition code 0 is set for CLEAR SUBCHANNEL (see the section "CLEAR SUBCHANNEL" in Chapter 14, "I/O Instructions"). Bit 19 is set to zero when the resulting status-pending condition is cleared by TEST SUBCHANNEL.

LzLabs GmbH. Ex. 1016-409

Activity Control

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 by 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 valid at an installed subchannel whenever the device-number-valid bit is one (see the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions"). The activity-control bits are defined as follows:

Bit	Designation
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 bits in the function-control field and the resume-, start-, halt-, clear-pending, subchannel-active, and suspended bits in the activity-control field are set to zeros at the subchannel when primary status, alert status, or the status-pending condition (in the cases of CLEAR SUBCHANNEL and HALT SUBCHANNEL) is cleared by TEST SUBCHANNEL.

Resume-Pending (Bit 20): When one, bit 20 indicates that a resume function is currently pending at the subchannel. The subchannel can be resume-pending only when a start function is in progress at the subchannel. The start function previously specified by a START SUBCHANNEL is start-pending, is currently being executed, is currently suspended, or is primary-status-pending. The resume-pending bit is set to one when condition code 0 is set for RESUME SUBCHANNEL. The point at which bit 20 is set to zero 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 (the suspended bit of the activity-control field is one) when the resume-pending condition is recognized, the CCW that caused the suspension is refetched, and the setting of the suspend flag is examined.

1. If the CCW suspend flag is one, the device is not selected, the subchannel resume-pending bit is reset to zero, and the subchannel 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 execute the path-management function as if a new start function is 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 previously caused suspension of channel-program execution. In this situation, the resume-pending bit is reset to zero when any of the following events occurs:

   a. The subchannel becomes subchannel-and-device-active (bits 24 and 25 are set to one) with the start function.

   b. An initial-status byte is accepted from the device for the first command which contains channel end in the absence of busy.

   c. CLEAR SUBCHANNEL is executed.

   d. TEST SUBCHANNEL clears any combination of alert and primary status or clears the status-pending condition alone.

If the subchannel is not in the suspended state (that is, the suspended bit of the activity-control field is zero) when the resume-pending condition is recognized, the CCW suspend flag of the most recently fetched CCW (if any) is examined and the following actions are taken:

1. If a CCW has not been fetched or the suspend flag of the most recently fetched CCW is zero the subchannel resume-pending bit is reset to zero and the resume function is not performed.

2. If the suspend flag of the most recently fetched CCW is one, the resume-pending bit is reset to zero, 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.

LzLabs GmbH. Ex. 1016-410

Some models recognize a resume-pending condition only after a CCW having a valid S flag set to one is fetched. Therefore, if a subchannel is resume-pending and, during execution of the channel program, no CCW is fetched that has a valid S flag set to one, the resume-pending bit remains one until the primary-interruption condition is cleared by TEST SUBCHANNEL.

Start-Pending (Bit 21): When one, bit 21 indicates that the I/O function specified by a START SUBCHANNEL is pending at the subchannel, awaiting execution. Bit 21 is set to one when condition code 0 is set for START SUBCHANNEL.

Bit 21 is set to zero at the subchannel when any of the following occurs:

1. The subchannel becomes subchannel-and-device-active (bits 24 and 25 are set to ones) with the start function.
2. An initial status byte is accepted from the device for the first command which contains channel end in the absence of busy.
3. The subchannel becomes suspended (bit 26 is set to one) because of a valid suspend flag in the first CCW.
4. CLEAR SUBCHANNEL is executed.
5. TEST SUBCHANNEL clears any combination of alert and primary status, clears the status-pending condition alone, or clears the secondary status after the execution of a halt function.

Halt-Pending (Bit 22): When one, bit 22 indicates that the halt function specified by HALT SUBCHANNEL is pending at the subchannel, awaiting execution. Bit 22 is set to one when condition code 0 is set for HALT SUBCHANNEL. Bit 22 is set to zero when the halt signal has been issued to the device or when CLEAR SUBCHANNEL is executed. The halt-pending bit remains one (unless CLEAR SUBCHANNEL is executed) if the channel subsystem determines that the halt signal cannot be issued to the device (see the discussion of path management in the section "Halt-Function Execution" in Chapter 15, "Basic I/O Functions") and is set to zero at the subchannel when TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.

Clear-Pending (Bit 23): When one, bit 23 indicates that the clear function specified by CLEAR SUBCHANNEL is pending at the subchannel, awaiting execution. Bit 23 is set to one when condition code 0 is set for CLEAR SUBCHANNEL and is set to zero when the clear signal has been issued to the device. The clear-pending bit remains one if the channel subsystem determines that the clear signal cannot be issued to the device (see the discussion of path management in the section "Clear-Function Execution" in Chapter 15, "Basic I/O Functions") and is set to zero at the subchannel when TEST SUBCHANNEL clears the status-pending condition alone.

Subchannel Active (Bit 24): When one, bit 24 indicates that an I/O operation is currently in execution at the subchannel. Bit 24 is set to one only when the device has accepted the first command describing an I/O operation and the start function or resume function is not immediately concluded at the subchannel as a result of solicited device status received as initial status for the command (see the section "Immediate Conclusion of I/O Operations" in Chapter 15, "Basic I/O Functions"). The above conditions are satisfied when any of the following initial-device-status combinations are accepted from the device:

1. All zeros
2. Unit check, status modifier, and channel end (command retry, delayed)
3. Unit check, status modifier, channel end, and device end (command retry, immediate)
4. Channel end when the chain-command flag is one in the CCW
5. Channel end and device end when the chain-command flag is one in the CCW
6. Channel end, device end, and status modifier when the chain-command flag is one in the CCW.

(See the section "Command Retry" in Chapter 15, "Basic I/O Functions," for the description of the command-retry procedure.)

The subchannel-active bit is not set to one during execution of the function specified by either a HALT SUBCHANNEL or a CLEAR SUBCHANNEL instruction. Subchannel active (bit 24) is set to zero when the channel subsystem is no longer actively participating in the execution of the function. The end of channel subsystem participation is signaled by the subchannel when either the primary-status bit or the suspended bit is set to one. The subchannel-active bit is also set to zero during the execution of CLEAR SUBCHANNEL and when the device appears not operational during execution of a halt function.

LzLabs GmbH. Ex. 1016-411

Device Active (Bit 25): When one, bit 25 indicates that an I/O operation is currently in progress at the associated device. Bit 25 is set to one (1) when the device has accepted the first command describing an I/O operation, as described under "Subchannel Active" above, and (2) when status containing channel end but neither busy nor device end is presented by the device as initial status for the first command transferred to the device as a result of a start or resume function, and command chaining is not specified in the CCW. (See the section "Immediate Conclusion of I/O Operations" in Chapter 15, "Basic I/O Functions.") The device-active bit is set to zero (1) when status is received from the device indicating that the previous I/O operation has been completed at the device, (2) when the device appears not operational during execution of a halt function, or (3) when the subchannel becomes suspended. If the subchannel is not start-pending or if the status received from the device also describes an alert condition, the subchannel becomes status-pending with the secondary-interruption condition. After the secondary-interruption condition has been placed in the subchannel, the device is capable of accepting a command for executing a new I/O operation. If the subchannel is start-pending and if the secondary-device status is device end or device end with control-unit end, then the channel subsystem discards the secondary status and initiates the new I/O operation by sending the first command to the device by taking the normal path-management actions for path selection (see the discussion of path management in the section "Start-Function Execution" in Chapter 15, "Basic I/O Functions"). 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 device-active bit is not set to one during execution of the functions specified by either a HALT SUBCHANNEL or a CLEAR SUBCHANNEL instruction. The device-active bit is set to zero during the execution of CLEAR SUBCHANNEL.

Suspended (Bit 26): When one, bit 26 indicates that channel-program execution is currently suspended. Bit 26 is set to one at the subchannel as part of the suspend function (see the section "Suspension of Channel-Program Execution" in Chapter 15, "Basic I/O Functions").

Bit 26 is reset to zero at the subchannel under any of the following conditions:

1. As part of the resume function following the execution of RESUME SUBCHANNEL when the subchannel-active bit is set to one.
2. When the resume function has been initiated at the device and the solicited device status received as initial status for the first command contains channel end.
3. As part of the clear function following the execution of CLEAR SUBCHANNEL.
4. When TEST SUBCHANNEL clears any combination of primary, secondary, or alert status or clears the status-pending condition when it appears alone.

Status Control

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 the description of bit 8, word 1, in the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions") 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 always valid; bits 27-30 are valid 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) in an SCSW stored by TEST SUBCHANNEL, bit 27 indicates an alert-interruption condition exists. In such a case, the subchannel is said to be status-pending with alert status. The status-pending bit and the alert-status bit are set to ones when an alert-interruption condition exists at the subchannel. An alert-interruption condition is recognized when alert status is stored 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

LzLabs GmbH. Ex. 1016-412

condition has suppressed command chaining or would have suppressed command chaining if chaining had been specified (see the section "Channel-Command Word" in Chapter 15, "Basic I/O Functions").

- The subchannel is subchannel-and-device-active, command chaining is not specified, 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 section "Status Modifier" later in this chapter).

- 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 execution of the pending I/O operation is not initiated.

When TEST SUBCHANNEL is executed, thus storing an SCSW with the alert-status bit as one in the IRB, the alert-status bit and the status-pending bit are reset to zeros at the subchannel. These bits are also reset to zeros at the subchannel during execution of CLEAR SUBCHANNEL.

Whenever an alert-status condition exists, it is brought to the attention of the program. Examples of alert status include incorrect length, program check, unit check, a device end signaling a not-ready-to-ready transition, and attention.

Intermediate Status (Bit 28): When one (and when the status-pending bit is also one) in an SCSW stored by TEST SUBCHANNEL, 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. The intermediate-status bit and the status-pending bit are set to ones when an intermediate-interruption condition exists at the subchannel.

When the initial-status interruption-request bit is one in an ORB, the subchannel becomes status-pending with intermediate status (the Z bit indicated) only after initial status is received for the first CCW of the channel program and the subchannel is subchannel-active (bit 24 is one at the subchannel). If the subchannel does not become subchannel-active, the Z condition is not generated.

When the suspend flag is indicated in a CCW, suspend control is indicated in the ORB, and the subchannel-suspended interruption condition is not suppressed in the ORB, then the intermediate-status bit and the status-pending bit are set. When the suspended flag is indicated in the first CCW of a channel program, the subchannel is suspended and 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 indicated in a CCW fetched during command chaining, the subchannel is suspended and becomes status-pending with intermediate status (the suspended bit indicated) only after execution of the preceding CCW is complete.

When the PCI flag is indicated in a CCW, the setting of the intermediate-status bit and the status-pending bit is dependent on whether the CCW is the first CCW of the channel program. When the PCI flag is indicated 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 indicated in a CCW fetched while chaining, the subchannel becomes status-pending with intermediate status (the PCI bit indicated) only after 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 carryover 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-status condition remains pending at the subchannel, but the interruption request, if any, is withdrawn, and the status-pending bit is reset to zero. The status-pending bit remains zero until execution 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, thus storing an SCSW with the intermediate-status bit as one in the IRB, the intermediate-status bit and the status-pending bit are reset to zeros at the subchannel. These bits are also reset

LzLabs GmbH. Ex. 1016-413

to zeros at the subchannel during the execution of CLEAR SUBCHANNEL.

Primary Status (Bit 29): When one (and when the status-pending bit is also one) in an SCSW stored by TEST SUBCHANNEL, 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 termination of the start function at the subchannel. The primary interruption condition is described by the SCSW stored as a result of executing TEST SUBCHANNEL. 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 primary-status-pending (and secondary-status-pending) with both the subchannel-status field and the device-status field set to zero.

When TEST SUBCHANNEL is executed, thus storing an SCSW with the primary-status bit as one in the IRB, the primary-status bit and the status-pending bit are reset to zeros at the subchannel. These bits are also reset to zeros at the subchannel during the execution of CLEAR SUBCHANNEL.

Secondary Status (Bit 30): When one (and when the status-pending bit is also one) in an SCSW stored by TEST SUBCHANNEL, 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 termination of the I/O operation at the device. The secondary-interruption condition is described by the SCSW stored as a result of executing TEST SUBCHANNEL.

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 secondary-status-pending (and primary-status-pending if the subchannel is also subchannel-active) with zeros for subchannel and device status.

When TEST SUBCHANNEL is executed, thus storing an SCSW with the secondary-status bit as one in the IRB, the secondary-status bit and the status-pending bit are reset to zeros at the subchannel. These bits are also reset to zeros at the subchannel during execution of CLEAR SUBCHANNEL.

Status-Pending (Bit 31): Whenever an interruption condition is generated at the subchannel, bit 31 is set to one, indicating 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 issued to a subchannel that is idle, the subchannel becomes status-pending subsequent to execution of the halt function to notify the program that the halt function has been completed. When CLEAR SUBCHANNEL is issued to an operational subchannel, the subchannel becomes status-pending subsequent to execution 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, specifies the type of status condition. For example, if bits 29 and 31 are ones, the subchannel is primary-status-pending. 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- and device-status fields are unpredictable. If bit 31 is not one, then the remaining bits of the status-control field have no meaning.

DEVICE-STATUS CONDITIONS

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 valid whenever the subchannel is status-pending with any combination of primary, secondary, intermediate, or alert status. Whenever the subchannel is intermediate-status-pending 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.

If, within a system, the I/O device is accessible from more than one channel path, status related to channel-subsystem-initiated operations in single path mode (solicited status) is signaled over the initiating channel path. Devices operating in 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 conci-

LzLabs GmbH. Ex. 1016-414

tion 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.

Attention

Attention is generated when the device detects an asynchronous condition that is significant to the program. The condition may also be described by other status indications that accompany attention. Attention is interpreted by the program and is not associated with the initiation, execution, or conclusion of an I/O operation.

The device can signal the attention condition to the channel subsystem when no operation is in progress at the I/O device. Attention can be indicated with device end upon completion of an operation, and it can be presented to the channel subsystem during the initiation of a new I/O operation.

When the device signals attention during the initiation of an operation, the operation is not initiated. Attention accompanying device end causes command chaining and command retry to be suppressed.

An I/O device may present attention accompanied by device end and unit exception when a not-ready-to-ready-state transition is signaled (see the section "Device End" later in this chapter).

Status Modifier

Status modifier is generated by the device when the device cannot provide its current status in response to interrogation by the channel subsystem, when the control unit is busy, when the normal sequence of commands has to be modified, or when command retry is to be initiated.

When the device is interrogated and the status-modifier condition signaled, in the absence of any other status bit, this indicates that the device cannot provide its current status. The interruption condition, which may be pending at the device, is not cleared. The 2702 Transmission Control is an example of a type of device that cannot provide its current status as a result of channel-subsystem interrogation.

Presence of status modifier and device end means that the normal sequence of commands must be modified. The handling of this set of bits by the channel subsystem depends on the operation. If command chaining is specified in the current CCW and no unusual conditions have been detected, presence of status modifier and device end causes the channel subsystem to fetch and chain to the CCW whose main-storage address is 16 higher than that of the current CCW. If the I/O device signals the status-modifier condition at a time when no command chaining is specified, or when any unusual conditions have been detected, no action is taken by the channel subsystem, and the status-modifier bit is placed in the SCSW.

Status modifier is presented in combination with unit check and channel end to initiate the command-retry procedure.

Control units that recognize special conditions that must be brought to the attention of the program present status modifier along with other status indications in order to modify the meaning of the status. The status presented is unrelated to the execution of an I/O operation.

When status modifier is generated together with the busy status bit, it indicates that the busy condition pertains to the control unit associated with the addressed I/O device. The control unit appears busy when it is executing a type of operation that precludes the acceptance and execution of any command and may appear busy when it contains status or sense information for a device other than the one addressed. The status may be control-unit end or channel end following the execution of the halt function. The busy state occurs for operations such as backspace tape file, in which case the control unit remains busy after providing channel end for operations concluded by HALT SUBCHANNEL. The busy state temporarily occurs on the IBM 3705 Communication Controller after initiation of an operation on a device accommodated by the control unit. A control unit accessible from two or more channel paths appears busy to the other channel paths when it is communicating with any of the channel paths.

Control-Unit End

Control-unit end indicates that the control unit has become available for use for another operation.

The control-unit-end condition is provided only by control units shared by I/O devices or control units accessible by two or more channel paths, and only when one or both of the following conditions have occurred:

LzLabs GmbH. Ex. 1016-415

1. The channel subsystem had previously caused the control unit to be interrogated while the control unit was busy. The control unit is considered to have been interrogated in the busy state when a command has been issued to a device on the control unit, and the control unit had responded with busy and status modifier in the device status byte.

2. The control unit detected an unusual condition during the portion of the operation after channel end had been signaled to the channel subsystem. The indication of the unusual condition accompanies control-unit end. However, the signaling of control-unit end and device end does not necessarily describe an unusual condition.

The two conditions described above are reset by the reset signal and the clear signal. Therefore, if one of these signals occurs before control-unit end is generated, no control-unit end is generated. If control-unit end has been generated but not presented to the channel subsystem by the time one of the signals occurs, the pending control-unit end is reset.

If the control unit remains busy with the execution of an operation after signaling channel end but has not detected any unusual conditions and has not been interrogated by the channel subsystem, control-unit end is not generated. Similarly, control-unit end is not provided when the control unit has been interrogated and could perform the indicated function. The latter case is indicated by the absence of busy and status modifier in the response to the interrogation.

When the busy condition of the control unit is temporary, control-unit end may be included with busy and status modifier in response to the interrogation even though the control unit has not yet been freed. The busy condition is considered to be temporary if its duration is 2 milliseconds or less. If a temporary busy condition is indicated, the channel subsystem assumes the responsibility to periodically reinterrogate the control unit until it is no longer busy. The IBM 3705 Communications Controller is an example of a device in which the control unit may be busy temporarily and which includes control-unit end with busy and status modifier.

The control-unit end condition can be signaled with channel end, with device end, or between the two.

Control-unit end may be signaled at other times and may be accompanied by other status bits. When control-unit end is signaled in the absence of any other status, the status may be identified with any device recognized by the control unit. For control units attaching more than a single I/O device, a pending control-unit end for one I/O device does not necessarily preclude initiation of new operations with other attached devices. Whether the control unit allows initiation of other operations is at the option of the control unit.

When control-unit end is presented to the channel subsystem subsequent to the acceptance of channel end and is accompanied by other status indications, command chaining is suppressed, if indicated, and an interruption condition may be generated indicating one or more of the following conditions:

1. A secondary-interruption condition, in the following cases:

   a. Control-unit end accompanied by device end and other status indications, or

   b. Control-unit end accompanied by only device end while the subchannel is not start-pending.

2. An alert-interruption condition, in the following cases:

   a. Control-unit end accompanied by device end while the subchannel is subchannel-active, or

   b. Control-unit end accompanied by status other than device end.

3. A primary-interruption condition if the subchannel is subchannel-active.

When control-unit end alone is presented to the channel subsystem, the channel subsystem resets internal indications of control-unit busy and discards the control-unit-end status without recognizing an interruption condition, unless all of the following conditions are met:

1. The control-unit end is presented on the channel path with which the channel subsystem is maintaining a working allegiance for this subchannel.

2. The device is not operating in multipath mode (see the discussion of multipath mode in the section "Path-Management-Control Word" in Chapter 15, "Basic I/O Functions").

3. The subchannel is subchannel-and-device-active.

4. Channel-end status has been previously presented, and command chaining is indicated.

LzLabs GmbH. Ex. 1016-416

If all of the above conditions are met, the channel subsystem suppresses command chaining and recognizes an interruption condition indicating primary, secondary, and alert status.

Control-unit end presented with channel end is unusual status and causes the channel subsystem to suppress command chaining, if indicated, and recognize an interruption condition for the subchannel with primary and alert status indicated.

Busy

Busy indicates that the device cannot execute the command because (1) it is executing a previously initiated operation, (2) it has pending status which must be presented to the channel subsystem, (3) the device is currently inaccessible because of a busy shared facility existing between the control unit and device, as in the case of the string-switch feature on the IBM 3830 Model 2, or (4) a self-initiated function is being performed. The pending status for the addressed device, if any, accompanies the busy indication. If the busy condition applies to the control unit, busy is accompanied by status modifier.

Whenever the device indicates that a busy condition exists and it is unable to execute an operation, the device responds to the channel subsystem when it becomes no longer busy (see the section "Device End" later in this chapter).

Channel End

Channel end is caused by the completion of the portion of an I/O operation involving transfer of data or control information between the I/O device and the channel subsystem.

Each I/O operation initiated at the I/O device causes a channel-end condition to be generated, and there is only one channel end for an operation. The channel-end condition is not generated when programming errors or equipment malfunctions are detected during initiation of the operation. When command chaining takes place, only the channel end of the last operation of the chain is made available to the program. The channel-end condition is not made available to the program when a chain of commands is prematurely concluded because of an unusual condition indicated with device end or during the initiation of a chained command.

The instant within an I/O operation when channel end is generated depends on the operation and the type of device. For operations such as writing on magnetic tape, the channel-end condition occurs when the block has been written. On devices that verify the writing, channel end may or may not be delayed until verification is performed, depending on the device. When magnetic tape is being read, the channel-end condition occurs when the gap on tape reaches the read-write head. On devices equipped with buffers, such as the IBM 3211 Printer Model 1, the channel-end condition occurs upon completion of data transfer between the channel subsystem and the buffer. During control operations, channel end is generated when the control information has been transferred to the devices, although for short operations the condition may be delayed until completion of the operation. Operations that do not cause any data to be transferred can provide the channel-end condition during the initiation sequence.

Channel end is presented in combination with status modifier and unit check with a channel-path-type-dependent signal sequence to initiate the command-retry procedure.

Device End

Device end is indicated (1) when the completion of an I/O operation occurs at the I/O device, (2) when the device signals that a change from the not-ready to ready state has occurred, (3) when the termination of an activity has occurred which previously caused a response of busy to the channel subsystem, and (4) when the I/O device signals that an asynchronous condition has been recognized. Device end normally indicates that the I/O device has become available for use for another operation.

Each I/O operation initiated at the I/O device causes one and only one device end for an I/O operation. The device-end condition is not generated when any programming or equipment malfunction is detected during initiation of the operation. When command chaining is specified and the suspend flag is zero in the next CCW, receipt of the device-end signal, in the absence of any unusual conditions, causes the channel subsystem to initiate transfer of the next command. When command chaining takes place, the only device end made available to the program is that of the last operation of the chain, unless an unusual condition is detected during the initiation of a chained command. If an unusual condition is detected during the initiation of a chained command, the subchannel becomes primary-and-secondary

LzLabs GmbH. Ex. 1016-417

status-pending, with the SCSW indicating the unusual condition without including the device-end indication.

The device-end condition associated with an I/O operation is generated either simultaneously with the channel-end condition or later. For data transfer on some I/O devices, the I/O operation is completed at the time channel end is generated, and both device end and channel end occur together. The time at which device end is presented depends upon the I/O-device type and the kind of command executed. For most I/O devices, device end is presented when the the I/O operation is completed at the I/O device. In some cases, for reasons of performance, device end is presented before the I/O operation has actually been completed at the I/O device. However, in all cases, when device end is presented, the I/O device is available for execution of an immediately following CCW if command chaining was specified in the previous CCW.

On buffered devices, such as an IBM 3211 Printer Model 1, the device-end condition occurs upon completion of the mechanical operation. When device end is generated later than channel end for the last I/O operation of a channel program, the program may elect to request the initiation of another start function prior to receiving the device-end indication. If the device-end indication is solicited and the subchannel is start-pending for a new start function, the device-end indication is discarded by the channel subsystem, and the pending I/O operation is initiated.

For control operations, device end is generated at the completion of the operation at the device. The operation may be completed at the time channel end is generated or later.

When the state of a device is changed from not ready to ready, either device end or device end, attention, and unit exception are indicated. Refer to the System Library publication for the device to determine which indication is given.

Unit Check

Unit check indicates that the I/O device has detected an unusual condition that is detailed by the information available to a sense command. Unit check may indicate that a programming or an equipment error has been detected, that the not-ready state of the device has affected the execution of the command, or that an exceptional condition other than the one identified by unit exception has occurred. The unit-check bit provides a summary indication of the conditions identified by sense data.

An error condition causes the unit-check indication when it occurs during the execution of a command, during some activity associated with an I/O operation, or when an unusual condition is detected that is unrelated to execution of an I/O operation. Unless the error condition pertains to the activity initiated by a command or is of immediate significance to the program, the condition does not cause the program to be alerted after device end has been signaled to the channel subsystem; a malfunction may, however, cause the device to become not ready. If an error condition of immediate significance to the program occurs while there is no I/O operation in progress, unit check is presented together with attention, control-unit end, or device end as unsolicited alert status.

Unit check is indicated when the existence of the not-ready state precludes a satisfactory execution of the command, or when the command, by its nature, tests the state of the device. When no status condition is pending for the addressed device at the control unit, the control unit signals unit check when a command is issued to a not-ready device. In the case of no-operation, the command is rejected, and channel end and device end do not accompany unit check.

Unless the command is designed to cause unit check, such as rewind and unload on magnetic tape, unit check is not indicated if the command is properly executed, even though the device has become not ready during or as a result of the operation. Similarly, unit check is not indicated if the command can be executed with the device not ready. Selection of a device in the not-ready state does not cause a unit-check indication when the sense command is issued, and whenever a status condition is pending at the addressed device.

If the device detects during the initiation sequence that the command cannot be executed, unit check is presented to the channel subsystem and appears without channel end, or device end. Such device status indicates that no action has been taken at the device in response to the command. If the condition precluding proper execution of the operation occurs after the command has been accepted, unit check is accompanied by channel end, or device end, depending on when the condition was detected. Any errors associated with an operation, but detected after device end has been signaled to the channel subsystem, are indicated by signaling unit check with attention.

LzLabs GmbH. Ex. 1016-418

Errors, such as invalid command code or invalid command-code parity, do not cause unit check when the device is active or contains a pending-status condition at the time of selection. Under these circumstances, the device responds by providing the busy bit and indicating the pending-status condition, if any. The command-code invalidity is not indicated.

Conclusion of an operation with the unit-check indication causes command chaining and command retry to be suppressed.

Unit check is presented in combination with channel end and status modifier to initiate the command-retry procedure.

Programming Notes

1. Unit-check status presented either in the absence of or accompanied by other status indicates only that sense information is available to the basic sense command. Presentation of either channel end and unit check or channel end, device end, and unit check does not provide any indication as to the kind of conditions encountered by the control unit, the state of the I/O device, or whether execution of the I/O operation ever was initiated even though the command may have been accepted. Descriptions of these conditions are provided in the sense information.

2. START SUBCHANNEL, HALT SUBCHANNEL, or CLEAR SUBCHANNEL may be issued to a subchannel whose associated device is attached to the same control unit that is currently holding sense data relative to a unit-check condition signaled by another attached device. The channel subsystem ensures that no sense data is lost. The execution of the function of the instruction may be delayed, however, until the sense data has been cleared from the control unit, or it may not take place at all, as in the case of CLEAR SUBCHANNEL. The sense data may be retrieved (or reset) by issuing START SUBCHANNEL to the subchannel that reported the unit check. Sense information is also reset when CLEAR SUBCHANNEL is issued to the subchannel, resulting in a clear signal on the channel path which presented unit check, and when RESET CHANNEL PATH is issued to the channel path which presented unit check.

Unit Exception

Unit exception is caused when the I/O device detects a condition that usually does not occur. Unit exception includes a condition such as recognition of a tape mark and does not necessarily indicate an error. During execution of an I/O operation unit exception has only one meaning for any particular command and type of device.

The unit-exception condition can be generated only when the device is executing an I/O operation, or when the device is involved with some activity associated with an I/O operation and the condition is of immediate significance to the program. If the device detects during the initiation sequence that the operation cannot be executed, unit exception is presented and appears without channel end or device end. Such unit status indicates that no action has been taken at the device in response to the command. If the condition precluding normal execution of the operation occurs after the command has been accepted, unit exception is accompanied by channel end, or device end, depending on when the condition was detected. Any unusual conditions associated with an operation, but detected after device end has been cleared, are indicated by signaling unit exception with attention.

If the I/O device responds with busy status to a command, the generation of unit exception is suppressed even when execution of that command usually causes unit exception to be indicated.

Concluding an operation with the unit-exception indication causes command chaining and command retry to be suppressed.

Some I/O devices present unit exception accompanied by device end and attention whenever the device changes from the not-ready state to the ready state (see the section "Device End" earlier in this chapter).

SUBCHANNEL-STATUS CONDITIONS

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 execution of a halt, resume, or start function. The subchannel-status field is valid whenever the subchannel is status-pending with any combination of primary, secondary, intermediate, or alert status.

LzLabs GmbH. Ex. 1016-419

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 the section "Program-Controlled Interruption" in Chapter 15, "Basic I/O Functions.")

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 storage after the assigned storage areas were filled. 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. The count in the SCSW is zero.

Short Block on Input: The number of bytes transferred during a read, read-backward, or sense operation is insufficient to fill the 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 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 specified by format-0 CCWs. The indication is unpredictable for immediate operations specified by format-1 CCWs when the count field does not specify zero.

Presence of the incorrect-length condition suppresses command chaining unless the SLI flag in the CCW is on or unless the condition occurs in an immediate operation specified by a format-0 CCW.

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 TIC (Format 1) is not zero.

Invalid CCW Address: The channel subsystem has attempted to fetch a CCW from a main-storage location which is not available. An invalid CCW address can occur because the program has specified an invalid address in the CPA 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 either because the absolute address does not correspond to a physical location or because format 0 has been specified in the ORB and the absolute address specifies a location greater than 16,777,215.

Invalid Command Code: The command code in the first CCW designated by the CPA or in a CCW fetched on command chaining has four low-order zeros. 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 the contents of the data-address field in the CCW do not designate the first IDAW on an integral word boundary.

Invalid IDAW Address: The channel subsystem has attempted to fetch an IDAW from a main-storage location which is not available. An invalid IDAW address can occur because the program has specified 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

LzLabs GmbH. Ex. 1016-420

either because the absolute address does not correspond to a physical location or because format 0 has been specified in the ORB and the absolute address specifies a location greater than 16,777,215.

Invalid Data-Address Specification: Bit 32 of a format-1 CCW is not zero.

Invalid Data Address: When one 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 or from a storage location which is either not available or is outside the addressing range specified by SET ADDRESS LIMIT and the limit mode in the subchannel.

An invalid data address can occur because the program has specified an invalid address in the CCW or in an IDAW, or because a limit-mode violation is detected when the address exceeds the boundary address specified by SET ADDRESS LIMIT, or because the channel subsystem, on sequentially accessing storage, attempted to access an unavailable location. A main-storage location is unavailable either because the absolute address does not correspond to a physical location or because format 0 has been specified in the ORB, indirect data addressing has not been specified, and the absolute address obtained by sequentially accessing storage specifies a location greater than 16,777,215. The boundary condition specified by SET ADDRESS LIMIT is under the control of the limit mode in the subchannel.

Note: The maximum storage address is determined as a function of whether 24-bit or 31-bit addressing is used. If format 0 is specified in the ORB, the maximum storage address recognized by the channel subsystem is 16,777,215 unless indirect data addressing is specified. Otherwise, the maximum storage address is 2,147,483,647. If format 1 is specified in the ORB, the maximum storage address recognized by the channel subsystem is 2,147,483,647.

Invalid IDAW Specification: Bit 0 of the IDAW is not zero, or the second or a subsequent IDAW does not specify the beginning or, for read-backward operations, the ending byte of a 2K-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 zeros in bit position 15, or a CCW specifying transfer in channel does not contain zeros in bit positions 0-3 and 8-31.

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 in the ORB (bit 4 of word 1).

Invalid ORB Format: The ORB does not contain zeros in bit positions 5-7, 13-15, and 24-31 of word 1.

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. Storage accesses associated with each I/O operation are performed using the key provided in the ORB associated with that operation.

LzLabs GmbH. Ex. 1016-421

When the protection-check condition occurs during the fetching of the first CCW or IDAW, the operation is not initiated. When protection check is detected after the I/O operation has been initiated at the device, the device is signaled to conclude the operation the next time it requests or offers a byte of data. However, if the device signals the channel-end condition before transferring any data designated by the CCW or IDAW, the status is accepted, and the subchannel becomes status-pending with protection check indicated. Other indications may accompany the protection-check indication as a function of the operation specified by the CCW 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 involving data, contained in main storage, that is currently used in the execution of an I/O operation. If the condition is detected but the data is not used, for example, when prefetching, the condition is not indicated. 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.

When, on an input operation, 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, 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 by a channel path 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 storage-access code and the CCW address, specifies 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 immediately. 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 extended-status-word bit is one, and subchannel-logout information is stored in the ESW when TEST SUBCHANNEL is executed.

Whenever channel-data check is indicated, no measurement parameters for the subchannel are stored.

Channel-Control Check

Channel-control check is caused by any machine malfunction affecting channel-subsystem controls. The condition includes invalid CBC on CCW and data addresses and invalid CBC on the associated keys or on the contents of CCWs and IDAWs. If an invalid CBC condition is detected on a CCW, data address, IDAW, or associated key that is not used, for example, when prefetching, the condition is not indicated.

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 immediately concluded. 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. The extended-status-word bit is one and subchannel-logout information is stored in the ESW when TEST SUBCHANNEL is executed.

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 parameters for the subchannel are stored.

Programming Note

If the SCSW indicates alert-status-pending but the field-validity flag indicates invalid device status, the program should assume that the channel-control-check condition occurred while the channel subsystem was accepting alert status from the device.

LzLabs GmbH. Ex. 1016-422

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 the following reasons:

1. The data or status byte received from a device while the subchannel is subchannel-and-device-active or device-active has invalid parity.
2. The status byte received from a device while the subchannel is idle, start-pending, suspended, or halt-pending has invalid parity,
3. A device responded with an address other than the address specified 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 the discussion of I/O-error alert in the section "Extended-Status Format 0" later in this chapter).
7. The device responded with a status byte, which, although valid, was presented at an improper time.

Detection of the interface-control-check condition causes the current operation, if any, to be immediately concluded, and the subchannel is made status-pending with alert status, primary 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. The extended-status-word-format bit is one and subchannel-logout information is stored in the ESW when TEST SUBCHANNEL is executed.

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 parameters for the subchannel are stored.

Programming Notes

If the SCSW indicates alert-status-pending but the field-validity flag indicates invalid device status, the program should assume that the interface control-check condition occurred while the channel subsystem was accepting alert status from the device and 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.

LzLabs GmbH. Ex. 1016-423

CCW-ADDRESS-FIELD CONTENTS

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 by TEST SUBCHANNEL, as indicated in the figure "CCW Address as Function of Subchannel State." 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 CPA (see the section "Operation-Request Block" in Chapter 15, "Basic I/O Functions") or generated during chaining, would cause the channel subsystem to fetch a CCW from a location greater than 16,777,215 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 2,147,483,647 while in 31-bit addressing mode, the rightmost 31 bits of the invalid address are stored in the CCW-address field.

LzLabs GmbH. Ex. 1016-424

Subchannel State[1]	CCW Address[2]
Start-pending[5] (UUUU0/AIPSX)	Unpredictable
Start-pending[5] and device-active (UUUU0/AIPSX)	Unpredictable
Subchannel-and-device-active[5] (UUUU0/AIPSX)	Unpredictable
Device-active only (UUUU0/AIPSX)	Unpredictable
Suspended[5] (YYYYY/AIPSX)	See note 1.
Status-pending (10001/AIPSX) because of unsolicited alert status from the device while the subchannel was start-pending[5]	Channel-program address (CPA) + 8
Status-pending (00111/AIPSX) because the device appeared not operational on all paths[5]	CPA + 8
Status-pending (10011/AIPSX) because of solicited alert status from the device while the subchannel was start-pending and device-active[5]	CPA + 8
Status-pending (10111/AIPSX) because of solicited alert status generated by the channel subsystem while the subchannel was start-pending[5] or start-pending and device-active[5]	See note 2.
Status-pending (01001/AIPSX) while the subchannel was subchannel-and-device-active[5]	CCW + 8 of the CCW that contained the last recognized PCI, or 8 higher than a CCW which has subsequently become current
Status-pending[5] (1Y1Y1/AIPSX). Termination occurred because of program check	
Unused bits in ORB not set to zeros	CPA + 8
Invalid CCW-address specification in transfer in channel (TIC)	Address of TIC + 8
Invalid CCW-address specification in the channel-program address (CPA)	Invalid CPA address + 8
Invalid CCW address in TIC	Address of TIC + 8
Invalid CCW address in the CPA	Invalid CPA + 8
Invalid CCW address while chaining	Invalid CCW address + 8

CCW Address as Function of Subchannel State (Part 1 of 4)

LzLabs GmbH. Ex. 1016-425

Subchannel State[1]	CCW Address[2]
Invalid command code	Address of invalid CCW + 8[3]
Invalid count	Address of invalid CCW + 8[3]
Invalid IDAW-address specification	Address of invalid CCW + 8[3]
Invalid IDAW address in a CCW	Address of invalid CCW + 8[3]
Invalid IDAW address while sequentially fetching IDAWs	Address of current CCW + 8
Invalid data-address specification, format 1	Address of invalid CCW + 8[3]
Invalid data address in a CCW	Address of invalid CCW + 8[3]
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[3]
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[5] (1Y1Y1/AIPSX). Termination occurred because of protection check	
On a CCW access	Address of the protected CCW + 8[3]
On data or an IDAW access	Address of current CCW + 8
Status-pending[5] (1Y1Y1/AIPSX). Termination occurred because of chaining check	Address of current CCW + 8
Status-pending[5] (YY1Y1/AIPSX). Termination occurred under count control	Address of current CCW + 8[4]
Status-pending[5] (1Y1Y1/AIPSX). Operation prematurely terminated by the device because of alert status	Address of current CCW + 8[4]

CCW Address as Function of Subchannel State (Part 2 of 4)

LzLabs GmbH. Ex. 1016-426

Subchannel State[1]	CCW Address[2]
Status-pending[5] (YYYY1/AIPSX) after termination by HALT SUBCHANNEL and the activity-control-field bits indicated below set to one	
Status-pending alone	Unpredictable
Start pending[5]	Unpredictable
Device active and start pending[5]	Unpredictable
Device active	Unpredictable
Subchannel active and device active[5]	CCW + 8 of the last executed CCW
Suspended	Address of CCW + 8 of CCW causing suspension
Suspended and resume pending	Unpredictable
Status-pending (00001/AIPSX) after termination by CLEAR SUBCHANNEL	Unpredictable
Status-pending[5] (YY1Y1/AIPSX). Operation terminated normally by the device	CCW + 8 of the last executed CCW[4]
Status-pending (00011/AIPSX)	Unpredictable
Status-pending (10001/AIPSX)	Unpredictable
Status-pending (00001/AIPSX)	Unpredictable
Status-pending[5] (1Y111/AIPSX). Command chaining suppressed because of alert status other than channel-control check or interface-control check	Address of current CCW + 8[4]
Status-pending[5] (1YYY1/AIPSX) because of alert status for channel-control check or interface-control check	See note 3.[4]
Status-pending[5] (1Y1Y1/AIPSX) because of channel-data check	Address of current CCW + 8[4]

CCW Address as Function of Subchannel State (Part 3 of 4)

LzLabs GmbH. Ex. 1016-427

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 meaningless 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 stored address is the address of the current CCW + 8 even though it is either invalid or protected.

[4] Incorrect length is indicated as a function of the setting of suppress-length-indication flag in the current CCW (see the section "Channel-Command Word" in Chapter 15, "Basic I/O Functions").

[5] The subchannel may also be resume-pending.

Notes:

1. Unless the subchannel is also resume-pending, the address stored is the address of the CCW that causes 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 CPA + 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 was recognized, then the address stored is the address of the CCW + 8 of the last executed CCW.

CCW Address as Function of Subchannel State (Part 4 of 4)

LzLabs GmbH. Ex. 1016-428

COUNT-FIELD CONTENTS

Bits 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 the figure "CCW Address as Function of Subchannel State"), indicates the number of bytes transferred to or from the area designated by the CCW. The count field is valid whenever the subchannel is primary-status-pending with (1) device status only or (2) subchannel status of incorrect length only and device status.

In the figure "Contents of Count Field in the SCSW," the contents of the count field are listed for all cases where the subchannel is either start-pending, subchannel-and-device-active, device-active, suspended, or status-pending.

LzLabs GmbH. Ex. 1016-429

Subchannel State[1]	Count
Start-pending[2](UUUU0/AIPSX)	Not meaningful[3]
Start-pending and status-pending[2] (10YY1/AIPSX)	Not meaningful[3]
Start-pending and device-active[2] (UUUU0/AIPSX)	Not meaningful[3]
Suspended[2] (YYYYY/AIPSX)	Not meaningful[3]
Subchannel-and-device-active[2] (UUUU0/AIPSX)	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[2] (1Y1Y1/AIPSX). Termination occurred because of:	
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[2] (YY1Y1/AIPSX). Termination occurred under count control.	Correct
Status-pending[2](Y0011/AIPSX)	Not meaningful[3]
Status-pending[2](1Y1Y1/AIPSX)	Correct. Residual count of last CCW used in terminated operation.
Status-pending[2] (1Y111/AIPSX). Command chaining suppressed because of alert status.	Correct. Residual count of last CCW used in terminated operation.
Termination by HALT SUBCHANNEL[2]	Unpredictable
Termination by CLEAR SUB-CHANNEL (00001/AIPSX)	Not meaningful[3]
Status-pending (YY1Y1/AIPSX). Operation terminated normally by the device.[2]	Correct. Indicates the residual 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]

Contents of Count Field in the SCSW (Part 1 of 2)

LzLabs GmbH. Ex. 1016-430

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 meaningless 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] 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 is correct, specifying the residual count of the last CCW used in the terminated operation.

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 many 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 specifies a byte location within the page in which the channel-data-check condition was recognized.

Contents of Count Field in the SCSW (Part 2 of 2)

EXTENDED-STATUS-WORD CONTENTS

The extended-status word (ESW) provides additional information to the program about the subchannel and its associated device. The ESW is stored when TEST SUBCHANNEL is issued to an operational subchannel and is placed in word 3 of the IRB that is specified as the TEST SUBCHANNEL operand. 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 contain one of the following types of extended-status formats:

Format 0: Subchannel logout stored in bytes 0-3

Format 1: Zeros stored in bytes 0 and 2-3, and the LPUM stored in byte 1

LzLabs GmbH. Ex. 1016-431

Format 2: Zeros stored in byte 0, the LPUM stored in byte 1, and the device-connect time stored in bytes 2-3

Format 3: Zeros stored in byte 0, and the LPUM stored in byte 1, with bytes 2-3 containing unpredictable values

Bytes 0-3 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 intermediate-status-pending alone for other than the suspended-interruption condition.

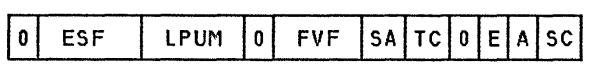
The type of extended-status format stored depends upon conditions existing at the subchannel at the time TEST SUBCHANNEL is executed. The conditions under which each of the types of formats is stored are described in the remainder of this section.

EXTENDED-STATUS FORMAT 0

A format-0 ESW is stored by TEST SUBCHANNEL when the extended-status-word-format bit (bit 5, word 0 of the SCSW) is one and the subchannel is status-pending with any combination of status as defined in the figure "Relationship between Logout Data and SCSW Bits." In this case, subchannel-logout information is 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 logout information.

Subchannel logout is not provided (and the extended-status bit is not valid) when the subchannel is status-pending with intermediate status alone.

The ESW has the following format:



Extended-Status Flags (ESF): Any of the bits 1-7, when one, specify that an error-check condition has been detected by the channel subsystem. The following indications are provided in the ESF field:

1. Key Check. Bit 1, when one, indicates either that the channel subsystem, when accessing data, when attempting to update the measurement block, or when attempting to fetch either a 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, or the measurement-block data-check bit, CCW-check bit, or IDAW-check bit (bits 3, 5, and 6 of the ESW) identifies the source of the key error.

   Note: If an invalid checking-block code on a key is detected but the data, CCW, or IDAW is not used, for example, when prefetching, the condition is not indicated to the program.

2. Measurement-Block Program Check. Bit 2, when one, indicates that the channel subsystem, in attempting to update the measurement block, has detected an invalid absolute address when combining the measurement-block origin with the measurement-block index for this subchannel.

3. Measurement-Block Data Check. Bit 3, when one, indicates that a machine error has been detected involving the parameters of the measurement block in main storage. (See the section "Measurement Block" in Chapter 17, "I/O Support Functions.") Measurement-block data check is indicated when the parameters or the associated key have an invalid checking-block code (CBC) in main storage when the measurement block is updated by the channel subsystem. When invalid CBC on the associated key is detected, the key-check bit, bit 1 of the ESF field, is also stored as one.

4. Measurement-Block Protection Check. Bit 4, when one, indicates that the channel subsystem, when attempting to update the measurement parameters, has been prohibited from accessing the measurement block because the storage key does not match the measurement-block key (see the section "Measurement Block" in Chapter 17, "I/O Support Functions"). Measurement-block storage accesses associated with each update operation are performed

LzLabs GmbH. Ex. 1016-432

by using a key provided by SET CHANNEL MONITOR (see the section "SET CHANNEL MONITOR" in Chapter 14, "I/O Instructions").

Note: Whenever any of the measurement-check conditions, bits 2-4, is indicated, the channel subsystem sets the subchannel measurement-block-update-mode-enable bit to zero, disabling the storing of measurement parameters for the subchannel (see the section "Path-Management-Control Word" in Chapter 15, "Basic I/O Functions").

5. 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 secondary-status-pending as a function of 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.

6. 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 secondary-status-pending as a function of 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: Detection of a channel-data-check condition does not cause the CCW and IDAW check bits to be stored as ones.

7. 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 path in use is set whenever one of the following occurs:

1. The first command of a start-subchannel function is accepted by the device (see the section "Activity Control" earlier in this chapter).

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 the section "Interruption Conditions" earlier in this chapter).

4. The channel subsystem recognizes an interface-control-check condition (see the section "Interface-Control Check" earlier in this chapter), and no logout information is currently present in 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 the section, "Extended-Status-Word Contents," earlier in this chapter) and the SCSW is stored. When logout information is present in the ESW, a zero LPUM-field-validity flag indicates that the LPUM setting is not consistent with the other logout indications.

Field-Validity Flags (FVF): Bits 17-21 specify the validity of the information stored in the designated fields of either the SCSW or the extended-status word. When the validity bit is one, the designated field has been stored and is usable for recovery purposes. When the validity bit is zero, the 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. Further, when these checks are not indicated, the last-path-used mask, device-status, and CCW-address fields are all valid.

The fields designated are:

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

LzLabs GmbH. Ex. 1016-433

subsystem at the time of error. It pertains only to the access of storage for the purpose of fetching or storing data during execution 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

Termination Code (TC): Bits 24-25 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.

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 issued 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).

Secondary Error (E): Bit 27, when one, indicates that a malfunction of a system component which 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 one of the following indications:

Channel-control check

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, selective reset 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. It pertains only to the state of operations initiated by START 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 device status has not yet been analyzed by the channel subsystem. This code is set during initial selection.

010 The command has been accepted by the device, but no data has been transferred. This code is set during initial selection, if the initial status is (1) channel end alone, (2) channel end and device end, (3) channel end, device end, and status modifier, or (4) all zeros.

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 either has not yet been sent to the device, was sent but not accepted by the device, or was sent and accepted but command-retry status was signaled. This code is set when one of the following conditions occurs:

1. When the command address is updated during command chaining or the initiation of a start or resume function at the device.

2. When during initial selection 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).

LzLabs GmbH. Ex. 1016-434

3. When command retry is signaled.

4. When the channel subsystem interrogates the device in the process of clearing an interruption condition.

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 into 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.

The figure "Relationship between Logout Data and SCSW Bits" defines the relationship between indications provided as logout data and the appropriate SCSW bits.

Logout Condition Indicated	Logout Condition for SCSW Indication of[2]		
	CDC	CCC	IFCC
Key check	V	V	0
Measurement-block-program check[1]	0	0	0
Measurement-block-data check[1]	0	0	0
Measurement-block-protection check[1]	0	0	0
CCW check	0	V	0
IDAW check	0	V	0
Last-path-used mask[3]	V	V	V
Field-validity flags	V	V	V
Termination code[3]	V	V	V
Secondary error	0	V	V
I/O-error alert	0	0	V
Sequence code[3]	V	V	V

Explanation:

0 No relationship

V Bit setting valid

CCC Channel-control check

CDC Channel-data check

IFCC Interface-control check

1 Measurement-check indications are mutually exclusive with each other.

2 When more than one SCSW indication is signaled, the logout conditions that are valid are the logical OR for each of the respective SCSW indications.

3 This field has a field-validity flag.

Relationship between Logout Data and SCSW Bits

LzLabs GmbH. Ex. 1016-435

EXTENDED-STATUS FORMAT 1

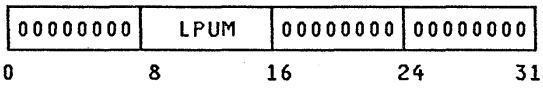
A format-1 ESW is stored by TEST SUBCHANNEL when all the following three conditions are met:

1. The extended-status bit (bit 5, word 0 of the SCSW is zero.
2. The subchannel is status-pending with
   a. Primary status alone, or
   b. Primary status together with other status, or
   c. Intermediate status alone for the suspended-interruption condition.
3. One of the following conditions is indicated:
   a. The device-connect-time-measurement mode is not active,
   b. The timing-facility bit (bit 14, word 1 of the PMCW) is zero, or
   c. The device-connect-time-measurement-enable bit (bit 12, word 1 of the PMCW) is zero.

Zeros are stored in bytes 0 and 2-3, and the LPUM is stored in byte 1.

The device-connect-time-measurement mode is made not active when SET CHANNEL MONITOR is executed and bit 31 of general register 1 is zero.

A format-1 ESW has this format:



Last-Path-Used Mask (LPUM): For a definition of the LPUM, see the description of bits 8-15 in the section "Extended-Status Format 0" earlier in this chapter.

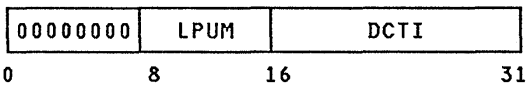
EXTENDED-STATUS FORMAT 2

A format-2 ESW is stored by TEST SUBCHANNEL when:

1. The extended-status bit (bit 5, word 0 of the SCSW) is zero,
2. The timing-facility bit (bit 14, word 1 of the PMCW is one,
3. The device-connect-time-measurement-enable bit (bit 12, word 1 of the PMCW) is one,
4. The device-connect-time-measurement mode is active, and
5. The subchannel is status-pending with either:
   a. Primary status alone,
   b. Primary status together with other status conditions, or
   c. Intermediate status alone for the suspended-interruption condition.

Zeros are stored in byte 0, the LPUM is stored in byte 1, and the device connect time is stored in bytes 2-3.

A format-2 ESW has this format:



Last-Path-Used Mask (LPUM): For a definition of the LPUM, see the description of bits 8-15 in the section "Extended-Status Format 0" earlier in this chapter.

Device-Connect-Time Interval (DCTI): Bits 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 active. The time increment of the DCTI is 128 microseconds.

If the above conditions for the storing of device-connect-time information in the ESW are met but the device-connect-time-measurement mode was made active by SET CHANNEL MONITOR subsequent to 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.

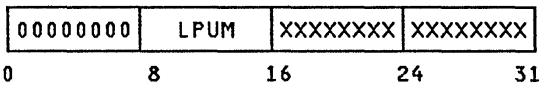
Note: The DCTI value stored in the ESW is the same as that used to update the corresponding parameter of the measurement block for the subchannel if the measurement-block-update mode is in use for the subchannel. If the measurement-block-update mode for the channel subsystem is active and the subchannel is enabled for the device-connect-time-measurement mode but no DCTI value is stored in the ESW (because of the presence of logout information), or if the DCTI is zeros, then nothing is added to the corresponding measurement-block parameter.

LzLabs GmbH. Ex. 1016-436

EXTENDED-STATUS FORMAT 3

A format-3 ESW is stored by TEST SUBCHANNEL when the extended-status bit (bit 5, word 0 of the SCSW) is zero and the subchannel is status-pending with any combination of secondary and alert status. Zeros are stored in byte 0, the LPUM is stored in byte 1, and bytes 2-3 contain unpredictable values.

A format-3 ESW has this format:



Last-Path-Used Mask (LPUM): For a definition of the LPUM, see the description of bits 8-15 in the section "Extended-Status Format 0" earlier in this chapter.

An "X" in the format indicates the bit may be zero or one.

The figure "Information Stored in ESW" summarizes the conditions at the subchannel under which each type of information is stored in the ESW.

LzLabs GmbH. Ex. 1016-437

Subchannel Conditions Under Which ESW is Stored by TSCH					Extended-Status Word(ESW)	
SCSW		PMCW				
Status-Control Field AIPSX	L Bit	Timing-Facility Bit	Device-Connect Time Measmnt Mode Enable Bit	Device-Connect Time Measmnt Mode Active	Contents Byte 0123	Format
10001	1	*	*	Y/N	RRRR	0
11001	1	*	*	Y/N	RRRR	0
10011	1	*	*	Y/N	RRRR	0
**1*1	1	*	*	Y/N	RRRR	0
0*001	1	*	*	Y/N	RRRR	0
**1*1	0	0	*	Y/N	ZMZZ	1
**1*1	0	*	0	Y/N	ZMZZ	1
**1*1	0	*	*	N	ZMZZ	1
01001[1]	-	0	*	Y/N	ZMZZ	1
01001[1]	-	*	0	Y/N	ZMZZ	1
01001[1]	-	*	*	N	ZMZZ	1
**1*1	0	1	1	Y	ZMDD	2
01001[1]	-	1	1	Y	ZMDD	2
*00*1[2]	0	*	*	Y/N	ZM**	3
----0	-	*	*	Y/N	****	U
00001	0	*	*	Y/N	****	U
01001[1]	-	*	*	Y/N	****	U

Explanation:

- \- Not meaningful.
- * Bits may be zeros or ones.
- [1] If the subchannel is intermediate-status-pending alone for the suspended-interruption condition, either a format-1 or a format-2 ESW is stored, depending upon the setting of the other fields shown in this figure. If the subchannel is intermediate-status-pending alone for other than the suspended-interruption condition, bytes 0-3 of the ESW contain unpredictable values.
- [2] Either the alert-status bit, the secondary-status bit, or both must be ones.
- A Alert status.
- D Accumulated device-connect-time-interval (DCTI) value is stored in bytes 2-3.
- I Intermediate status.
- L Extended-status-word format.
- M Last-path-used mask (LPUM) is stored in byte 1.
- N No.
- P Primary status.
- R Bytes 0-3 contain subchannel-logout information.
- S Secondary status.
- U No format defined.
- X Status pending.
- Y Yes.
- Z Bits are stored as zeros.

Information Stored in ESW

LzLabs GmbH. Ex. 1016-438

EXTENDED-CONTROL WORD

The extended-control word provides additional information to the program describing conditions that may exist at the channel subsystem, subchannel, or device. The setting of the extended-status-word-format (L) bit and the extended-control (E) bit, bits 5 and 14 of word 0, respectively, specify the contents of the extended-control word.

The information provided in the extended-control word is given in the figure "Information in Extended-Control Word."

Bits* 5 14	ECW Words 0-3	ECW Words 4-11
0 0	Zeros	Not defined
1 0	Zeros	Not defined
1 1	Zeros	Model-dependent information stored
* The combination 01 is reserved for future use.		

Information in Extended-Control Word

Unused bits in the model-dependent information are stored as zeros.

LzLabs GmbH. Ex. 1016-439

CHAPTER 17. I/O SUPPORT FUNCTIONS

Address-Limit Checking 17-1
Channel-Subsystem-Monitoring Facilities 17-2
  Channel-Subsystem Timing Facility 17-2
    Channel-Subsystem Timer 17-2
  Measurement-Block-Update Facility 17-4
    Measurement Block 17-4
    Measurement-Block Origin 17-5
    Measurement-Block Key 17-5
    Measurement-Block Index 17-6
    Measurement-Block-Update Mode 17-6
    Measurement-Block-Update Enable 17-6
    Time-Interval-Measurement Accuracy 17-6
  Device-Connect-Time-Measurement Facility 17-7
    Device-Connect-Time-Measurement Mode 17-7
    Device-Connect-Time-Measurement Enable 17-7
Signals and Resets 17-8
  Signals 17-8
    Halt Signal 17-8
    Clear Signal 17-8
    Reset Signal 17-8
  Resets 17-9
    Channel-Path Reset 17-9
    I/O-System Reset 17-9
Externally Initiated Functions 17-12
  Initial Program Loading 17-12
  Reconfiguration of the I/O System 17-14
Channel-Subsystem Recovery 17-14
  Channel-Report Word (CRW) 17-15
  Channel Report 17-15
  CRW Contents 17-16

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: address-limit checking, channel-subsystem monitoring, signals and resets, externally initiated functions, and channel-subsystem recovery.

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 program-controlled mode bits in each subchannel. The address-limit constraint operates at a higher priority than key-controlled protection so that I/O data accesses to the protected part of main storage are prevented even when the subchannel key is zero or matches the key in storage.

The address-limit-checking facility consists of the following elements:

- The I/O instruction SET ADDRESS LIMIT.
- The limit-mode bits in each subchannel.
- The address-limit-checking-control bit in the ORB.

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 contain zeros to specify a valid absolute address on a 64K-byte boundary; otherwise, an operand exception is recognized, and execution of the instruction is suppressed.

The setting of the limit-mode bits in each subchannel specifies how the address-limit checking is to be performed. The limit-mode bits are set by establishing the desired value in bits 9-10 of word 1 in the SCHIB and executing MODIFY SUBCHANNEL. The

LzLabs GmbH. Ex. 1016-441

settings of these bits in the subchannel 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. An attempt to set this combination in the subchannel causes an operand exception when MODIFY SUBCHANNEL is executed.

The address-limit-checking-control bit in the ORB (bit 11 of word 1) controls the application of address-limit checking for the start function 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 bits in the subchannel.

During the start-function execution, 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.

CHANNEL-SUBSYSTEM-MONITORING FACILITIES

Monitoring facilities are provided in the channel subsystem which give the program the ability to retrieve measured values for several I/O resource-usage parameters on a subchannel basis. The use of these facilities is under program control by SET CHANNEL MONITOR and program-modifiable enabling bits in each subchannel.

The channel-subsystem-monitoring facilities are the channel-subsystem timing facility, measurement-block-update facility, and device-connect-time-measurement facility. The latter two facilities are logically distinct and operate independent of one another. Each of the three facilities that constitute the channel-subsystem-monitoring facilities is described in this section.

CHANNEL-SUBSYSTEM-TIMING FACILITY

The channel-subsystem-timing facility provides the channel subsystem with the capability of measuring the elapsed time required for executing several different phases in processing 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 the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions.") If the channel-subsystem timing facility is not provided for the subchannel, its 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. Information concerning these restrictions can be found in the System Library publication for the system model and for the device.

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, not accessible to the program. The channel-subsystem timer is incremented by adding a one to the rightmost bit position every 128 microseconds. 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.

LzLabs GmbH. Ex. 1016-442

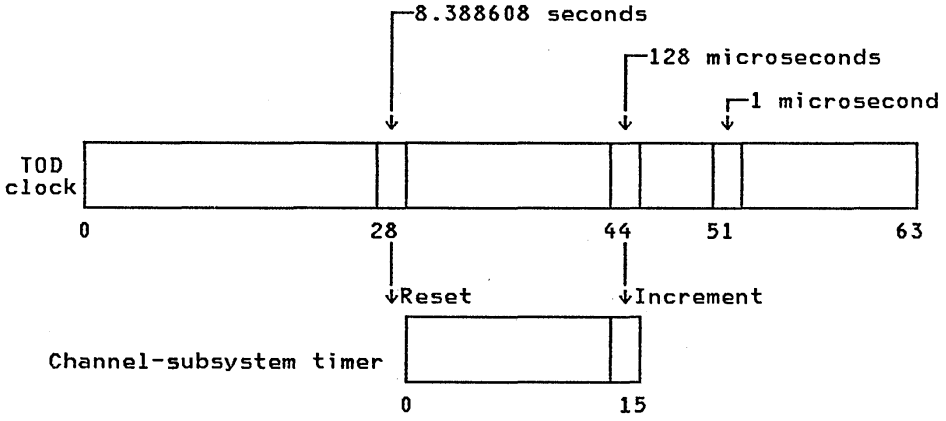
Just as every CPU has access to a TOD clock, every channel subsystem has access to at least one channel-subsystem timer. In configurations where a particular channel subsystem has multiple channel-subsystem timers, they are synchronized with each other if all of the TOD clocks in the configuration are running and synchronized. If the TOD clocks are not synchronized, the elapsed times measured by the channel-subsystem timing facility have unpredictable values for some or all of the subchannels, depending on how the associated devices are physically attached to the system. The values are unpredictable for those devices attached to the system by separately configurable paths whose associated CPU TOD clocks are not synchronized.

TOD-Clock Relationship: The channel-subsystem timer is incremented every 128 microseconds when bit position 44 of the associated TOD clock is incremented. When bit position 28 of the TOD clock is incremented, the channel-subsystem timer is set to the same value as the corresponding bits of the associated TOD clock. When the channel-subsystem timer and the associated TOD clock are synchronized, the channel-subsystem timer contains the same value as TOD-clock bits 29-44 (plus or minus one increment of the channel-subsystem timer). (See the figure "Logical Relationship Between a "Channel-Subsystem Timer and the Associated TOD Clock.")

Synchronization: Each channel-subsystem timer is automatically synchronized with the associated TOD clock every time TOD-clock bit position 28 is incremented. Continued synchronization is checked by the channel-subsystem timing facility within the approximately 8-second resynchronization cycle. If either the measurement-block-update mode or device-connect-time-measurement mode is active and any channel-subsystem timer and the associated TOD clock are 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 the section "Channel-Subsystem Recovery" later in this chapter). If neither of these modes is active, the lack of synchronization with the TOD clock is not recognized.

Since each channel-subsystem timer is automatically synchronized with its associated TOD clock, the synchronization of multiple channel-subsystem timers is dependent upon their multiple associated TOD clocks being synchronized with each other. TOD-clock synchronization is performed by the program using the SET CLOCK instruction. (See the section "Time-of-Day Clock" in Chapter 4, "Control.")

Loss of synchronization among channel-subsystem timers due to a loss of synchronization of their associated TOD clocks is not detected by the channel-subsystem timing facility.



Logical Relationship between a Channel-Subsystem Timer and the Associated TOD Clock

LzLabs GmbH. Ex. 1016-443

MEASUREMENT-BLOCK-UPDATE FACILITY

The measurement-block-update facility provides the program with the capability of accumulating resource-usage information on a subchannel basis. The program designates a contiguous area of absolute storage and subdivides this area into 32-byte blocks, one block for each subchannel for which resource-usage information is to be accumulated. Resource-usage information is accumulated in the block specified by the program each time an I/O operation or chain of I/O operations initiated by START SUBCHANNEL is suspended or completed. Accumulation of performance information by the measurement-block-update facility is under control of the program by use of the SET CHANNEL MONITOR measurement-block-update mode-control bit and the corresponding enable bit in the subchannel. Five parameters are accumulated by the measurement-block-update facility: SSCH+RSCH count, sample count, device-connect time, function-pending time, and device-disconnect time.

Measurement Block

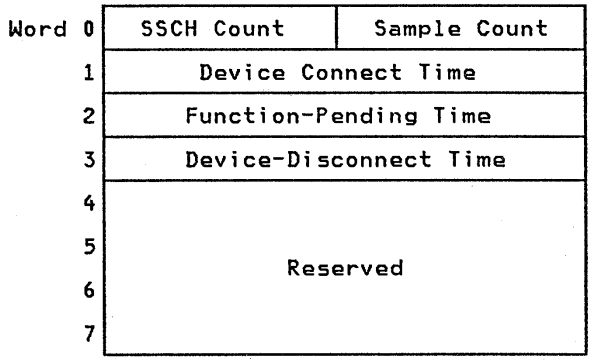
The measurement block is a 32-byte area at the location specified by the program using the measurement-block origin in conjunction with the measurement-block index. The measurement block contains the accumulated values of the measured parameters described below. When the measurement-block-update mode is active and the subchannel is enabled for measuring, the measurement-block-update facility measures the values for the parameters that accrue during the execution 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 at the subchannel 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 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 are not incremented.

If any of the accrued time values is detected to exceed the internal storage provided for accruing these values, none of the accrued values are added to the measurement block for the subchannel, the sample count is not incremented, but the SSCH+RSCH count is incremented.

Accesses to the measurement block by the measurement-block-update facility, in order to increment the parameters 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 by the measurement-block-update facility.

The measurement block has the following format:



SSCH+RSCH Count: Bits 0-15 of word 0 are used as a binary counter. When either the suspend function is performed or the primary-interruption condition is recognized during the execution of a start function, the measurement parameters are stored, and the counter is incremented by adding one in bit position 15. The counter wraps around from the maximum value of 65,535 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 value for an individual start function precludes the updating of the sample count and words 1-3, 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 not active, if the subchannel is not enabled for the measurement-block update, or if logout information has been generated for the start function.

Sample Count: Bits 16-31 of word 0 are used as a binary counter. When words 1, 2, and 3 of the measurement parameters are updated, the counter is incremented by adding one in bit position 31. On some models, certain conditions may preclude the measurement-block-update facility obtaining measured values for the parameters for an individual start function, even when the measurement-block-update mode is active and the subchannel is enabled for that mode. In

LzLabs GmbH. Ex. 1016-444

this situation, the sample-count field is not incremented.

The counter wraps around from the maximum value of 65,535 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 words 1, 2, and 3 of the measurement-block parameters.

Device-Connect Time: Bits 0-31 of word 1 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 for purposes of transferring information between it and the channel subsystem.

The time intervals are measured 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 three 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 provision of this parameter in the ESW are not affected by whether the measurement-block-update mode is active. (See the section "Device-Connect-Time-Measurement Facility" later in this chapter.)

Function-Pending Time: Bits 0-31 of word 2 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 issued to the subchannel while it is in the suspended state.

The function-pending-time interval is measured 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: Bits 0-31 of word 3 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 subsystem while the subchannel is active.

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 (subchannel active).

The device-disconnect-time interval is measured by 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.

Words 4-7 of the measurement block are not updated, but are reserved for future use.

Measurement-Block Origin

The measurement-block origin specifies the absolute address of the beginning of the measurement-block area on a 32-byte boundary in main storage. The measurement-block origin is passed from 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.

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 origin is passed.

LzLabs GmbH. Ex. 1016-445

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 measurement-block parameters for that subchannel. The location of the 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 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 preclude the possibility of unpredictable 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 executing SET CHANNEL MONITOR when bit 30 of general register 1 is 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. Bits 0-3 of general register 1 are also passed to the measurement-block-update facility as the access key to be used when updating the measurement block for each subchannel. When the measurement-block-update mode is active, the measurement-block-update facility accumulates measurements in individual measurement blocks within the measurement-block area for subchannels whose measurement-block-update-enable bit is one. (See the section "Measurement Block" earlier in this chapter 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 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 measurement-block key that are in control are old or current.

Measurement-Block-Update Enable

Bit 11, 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. Similarly, 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.

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.

LzLabs GmbH. Ex. 1016-446

Programming Notes

1. Excessive delays may be encountered by the channel subsystem when attempting to update measurement parameters if the program is concurrently accessing the same measurement-block area. A programming convention is desirable that ensures the storage block designated by SET CHANNEL MONITOR is made read-only for the duration of time that 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 unused words (words 4-7) 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 FACILITY

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). 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" in Chapter 16, "I/O Interruptions"). 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" in Chapter 16, "I/O Interruptions," 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 executing 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 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, 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

LzLabs GmbH. Ex. 1016-447

SUBCHANNEL issued to the subchannel. In this situation, the DCTI values are provided in the ESW (see the section "Extended-Status Format 2" in Chapter 16, "I/O Interruptions"). Similarly, if MODIFY SUBCHANNEL is executed with bit 12, 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.

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 and/or perform a reset function. 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:

1. The program executing the CLEAR SUBCHANNEL, HALT SUBCHANNEL, or RESET CHANNEL PATH instruction,
2. The I/O device signaling I/O error alert, or
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. 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.

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. 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.

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 path, the device immediately disconnects from that channel path. 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. 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

LzLabs GmbH. Ex. 1016-448

Interface Channel to Control Unit OEMI, GA22-6974.

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:

1. The program executing the RESET CHANNEL PATH instruction,
2. The operator activating a system-reset operator control or an initial-program-load operator control, or
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

Channel-path reset occurs when the channel subsystem performs the channel-path-reset function initiated by RESET CHANNEL PATH. (See the section "RESET CHANNEL PATH" in Chapter 14, "I/O Instructions.") All internal indications of dedicated allegiance, control-unit busy, and device busy that pertain to the specified 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 the section "I/O-System Reset" in this chapter.)

The results of the channel-path-reset function on the specified channel path are communicated to the program by means of a subsequent machine-check interruption condition generated by the channel (see the section "Channel-Subsystem Recovery" in this chapter).

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 the section "Initial Program Loading" later in this chapter), 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 is performed as part of subsystem reset, which also resets all floating interruption requests, including pending I/O interruptions. (See the section "Subsystem Reset" in Chapter 4, "Control.") 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 load (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 in the check-stop state (PAM is 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 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.

LzLabs GmbH. Ex. 1016-449

The term "operation in progress" used in the preceding paragraph has a slightly different meaning for devices that have the dynamic-reconnection facility and are operating in multipath mode than it does for devices that do not have that facility. For devices that are operating in single-path mode, an operation can be in progress for, at most, one channel path. Devices that are operating in 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 group are configured to the same channel subsystem. Therefore, if an operation is in progress in single path mode and the reset signal is received on that channel path, the operation in progress is reset. If, on the other hand, an operation is in progress in multipath mode and the reset signal is received on one of the channel paths of that group, then the operation in progress is reset for the resetting channel path only. This means that the operation in progress cannot continue on the resetting channel path but can continue on the other channel paths of the 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 multipath mode but the path group consists only of the resetting path, then the operation is reset.

3. Except as noted in item 2 above, 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 continued execution of the operation.

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.

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.

A 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.

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. The channel subsystem uses 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, to bring the channel paths into the quiescent state.

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 have their initial values. The initial

LzLabs GmbH. Ex. 1016-450

values of the following subchannel fields are zeros:

- Interruption parameter
- Interruption-subclass code
- Enabled bit
- Limit mode
- Multipath-mode bit
- Measurement-mode bits
- Path-not-operational mask
- Last-path-used mask
- Measurement-block index

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 the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions"; the section "Subchannel-Status Word" in Chapter 16, "I/O Interruptions"; and in the section "Extended-Status Word" in Chapter 16, "I/O Interruptions."

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 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 execution of I/O-system reset. See the section "Subsystem Reset" in Chapter 4, "Control."

LzLabs GmbH. Ex. 1016-451

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]
Interruption-subclass code	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]
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.

Summary of I/O-System-Reset Actions

EXTERNALLY INITIATED FUNCTIONS

I/O-system reset, which is an externally initiated function, is described in the section "I/O-System Reset" earlier in this chapter.

INITIAL PROGRAM LOADING

Initial-program-loading (IPL) provides a manual means for causing a program to be read from a specified 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 designating an input device with the load-unit-address controls and subsequently activating the load-normal or load-clear key. The load-normal key causes an initial-CPU-reset and a subsystem-reset operation to be performed, and the load-clear key causes a clear-reset operation to be performed. Other CPUs in the configuration perform CPU reset and clear reset, respectively.

LzLabs GmbH. Ex. 1016-452

Subsequently, a read operation is initiated from the selected input device and associated subchannel. The CPU does not necessarily enter the stopped state during execution of the reset operation. The load indicator is on while the CPU is in the load state.

The read operation is performed as if a START SUBCHANNEL instruction were executed that specified (1) the subchannel corresponding to the device number designated 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 request: zero (no request)

Address-limit-checking control: zero (no checking)

Suppress-suspended interruption: zero (suppression not allowed)

Logical-path mask: ones (all paths logically available)

Channel-program address: absolute address 0

The first CCW to be executed may be either an actual CCW stored at absolute location 0 or implied. In either case, the effect is as if a format-0 CCW were executed that has the following format:

Loc.

00	00000010	00000000	0000000000000000
04	01100000	////////	0000000000011000
	0	8	16 31

The above 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 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 is all zeros and the device-status field 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 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 if no machine malfunctions are detected, 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 is unsuccessful when any of the following occurs:

- No subchannel contains a valid device number equal to the IPL device number designated by the load-unit-address controls.
- A machine malfunction is detected in the CPU, main storage, or channel subsystem during the IPL operation.
- Unsolicited alert status is presented by the device between the

LzLabs GmbH. Ex. 1016-453

time the subchannel is made start-pending for the IPL read and 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 specifying 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 absolute 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. When the load-clear key is activated, the IPL program starts with a cleared machine in a known state, except that 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 reflected in the setting of the PAM bits in the affected subchannels (see the section "Subchannel-Information Block" in Chapter 15, "Basic I/O Functions").

CHANNEL-SUBSYSTEM RECOVERY

The channel-subsystem-recovery mechanism provides for extensive detection of machine malfunctions to ensure the integrity of channel-subsystem operation and to achieve automatic recovery of some malfunctions. Various reporting methods are used by the channel-subsystem recovery mechanism to assist in program recovery, maintenance and repair.

LzLabs GmbH. Ex. 1016-454

The method used to report a particular channel-subsystem malfunction is dependent upon the severity of the malfunction and the degree to which the malfunction can be isolated. Malfunctions in the channel subsystem are indicated to the 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 issued in response to the occurrence of an I/O interruption. (See Chapter 16, "I/O Interruptions," for a definition of the information stored, as well as Chapter 6, "Interruptions."

2. Information is provided in a channel report describing a machine malfunction affecting a specific facility within the channel subsystem. This information is made available to the program by the execution of STORE CHANNEL REPORT WORD, which is usually issued in response to the occurrence of a machine-check interruption. (See Chapter 11, "Machine-Check Handling," 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 machine malfunction 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 machine malfunction affecting the continued operational integrity of the channel subsystem. (See the section "Channel-Subsystem Damage" in Chapter 11, "Machine-Check Handling."

5. Information is provided in the MCIC describing a machine malfunction affecting continued operational integrity of a process or of the system (see the sections "Instruction-Processing Damage" and "System Damage" in Chapter 11, "Machine-Check Handling."

The channel-report facility is intended to be used to report malfunctions 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 WORD (CRW)

The channel-report word (CRW) provides error-related 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. Each CRW has the CRW format described in the section "CRW Contents" later in this chapter. Execution of recovery actions by the program or by external means may be required to gain recovery from the error condition. The CRW specifies the error environment and the severity of the error to the extent necessary for selecting the proper recovery action.

CHANNEL REPORT

When a malfunction or other condition affecting elements of the channel subsystem has been recognized, a channel report is generated. 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 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 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 if it is either the first channel report to be generated after the initial-program-load procedure 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. In the event that the program clears the first CRW of a 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. The ability of the channel subsystem to

LzLabs GmbH. Ex. 1016-455

cause a machine-check interruption indicating that a channel report is pending is subject to the setting of the machine-check mask, PSW bit 13, and the channel-report-pending subclass mask, bit 3 of control register 14. Both of these bits must be one for the interruption to take place.

If the channel report that is made pending is not an initial channel report, the program is not notified that the channel report was generated.

The 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 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 and as the 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, then 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 designates 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 designates 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 point in time.

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.

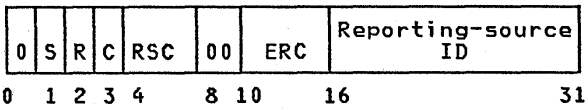
3. After a machine-check interruption has occurred with bit 9 of the machine-check-interruption code set to one, successive executions of STORE CHANNEL REPORT WORD should be performed until all of the pending channel reports have been cleared and condition code 1 has been set.

4. A CRW-overflow condition can occur if the program does not execute 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. While the overflow condition exists, CRWs that would have otherwise been made pending are lost and are never presented to the program.

CRW CONTENTS

The format of the CRW is as follows. Bits 0 and 8-9 are reserved and are always stored as zeros.



Solicited CRW (S): Bit 1, when one, specifies a solicited CRW. A CRW is considered by the channel subsystem to

LzLabs GmbH. Ex. 1016-456

be solicited if it is made pending as the direct result of some action that is taken by the program. For example, the solicited CRW bit is stored as one, indicating a channel-path-initialized condition when the CRW is made pending as the result of RESET CHANNEL PATH being executed and the channel-path-reset function being performed successfully. 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. For example, the solicited CRW bit is zero when a CRW is made pending as a result of the channel subsystem recognizing a channel-path-terminal condition.

Overflow (R): Bit 2, when one, specifies 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, specifies chaining of associated CRWs. Chaining of CRWs is specified whenever a machine malfunction 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 meaningless if the overflow bit, bit 2, is one.

Reporting-Source Code (RSC): Bits 4-7 specify the channel-subsystem facility that has been associated with the malfunction. 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	Designation
0 0 1 0	Monitoring facility
0 0 1 1	Subchannel
0 1 0 0	Channel path
1 0 0 1	Configuration-alert facility

All other bit combinations in the reporting-source-code field are reserved.

Error-Recovery Code (ERC): Bits 10-15 contain the error-recovery code which 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 reporting-source facility has already been recovered and is available for use or whether recovery actions are still required. The following error-recovery codes are possible:

Bits 10 11 12 13 14 15	Error State
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

All other bit combinations in the error-recovery-code field are reserved.

The specific meaning of each error-recovery code is dependent on the particular reporting-source code that accompanies it in a CRW. The error-recovery codes are defined as follows:

Available: The designated facility is in the same state that the program would expect if the CRW had not been generated.

Initialized: The designated 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: The designated 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 designated facility is in such a state that an operation that was in process can neither be completed nor terminated in the normal manner.

LzLabs GmbH. Ex. 1016-457

Permanent Error With Facility Not Initialized: The designated 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 designated 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.

Reporting-Source ID (RSID): Bits 16-31 contain the reporting-source ID which may, depending upon the machine malfunction and the reporting-source code, either further identify the channel-subsystem facility affected by the malfunction or provide additional information describing the malfunction. 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
4 5 6 7	Bits 16-31
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

Notes:

X = Subchannel number
Y = Channel-path ID (CHPID)

LzLabs GmbH. Ex. 1016-458

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-4
  Decimal Integers A-5
  Floating-Point Numbers A-5
  Conversion Example A-7
Instruction-Use Examples A-7
  Machine Format A-7
  Assembler-Language Format A-7
    Addressing Mode in Examples A-8
General Instructions A-8
  ADD HALFWORD (AH) A-8
  AND (N, NR, NI, NC) A-8
    And (NI) A-8
  BRANCH AND LINK (BAL, BALR) A-9
  BRANCH ON CONDITION (BC, BCR) A-9
  BRANCH ON COUNT (BCT, BCTR) A-10
  BRANCH ON INDEX HIGH (BXH) A-10
    BXH Example 1 ....A-10
    BXH Example 2 ....A-10
  BRANCH ON INDEX LOW OR EQUAL (BXLE) A-11
  COMPARE HALFWORD (CH) A-11
  COMPARE LOGICAL (CL, CLC, CLI, CLR) A-11
    Compare Logical (CLC) A-11
    Compare Logical (CLI) A-12
    Compare Logical (CLR) A-12
  COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) A-12
  COMPARE LOGICAL LONG (CLCL) A-13
  CONVERT TO BINARY (CVB) A-14
  CONVERT TO DECIMAL (CVD) A-14
  DIVIDE (D, DR) A-15
  EXCLUSIVE OR (X, XC, XI, XR) A-15
    Exclusive Or (XC) A-15
    Exclusive Or (XI) A-16
  EXECUTE (EX) A-17
  INSERT CHARACTERS UNDER MASK (ICM) A-17
  LOAD (L, LR) A-18
  LOAD ADDRESS (LA) A-18
  LOAD HALFWORD (LH) A-19
  MOVE (MVC, MVI) A-19
    Move (MVC) A-19
    Move (MVI) A-20
  MOVE LONG (MVCL) A-20
  MOVE NUMERICS (MVN) A-21
  MOVE WITH OFFSET (MVO) A-21
  MOVE ZONES (MVZ) A-22
  MULTIPLY (M, MR) A-22
  MULTIPLY HALFWORD (MH) A-23
  OR (O, OR, OI, OC) A-23
    Or (OI) A-23
  PACK (PACK) A-23
  SHIFT LEFT DOUBLE (SLDA) A-24
  SHIFT LEFT SINGLE (SLA) A-24
  STORE CHARACTERS UNDER MASK (STCM) A-24
  STORE MULTIPLE (STM) A-25
  TEST UNDER MASK (TM) A-25
  TRANSLATE (TR) A-26
  TRANSLATE AND TEST (TRT) A-26
  UNPACK (UNPK) A-28
Decimal Instructions A-28
  ADD DECIMAL (AP) A-28
  COMPARE DECIMAL (CP) A-29
  DIVIDE DECIMAL (DP) A-29
  EDIT (ED) A-29
  EDIT AND MARK (EDMK) A-31
  MULTIPLY DECIMAL (MP) A-32
  SHIFT AND ROUND DECIMAL (SRP) A-32

LzLabs GmbH. Ex. 1016-459

Decimal Left Shift A-32
Decimal Right Shift A-33
Decimal Right Shift and Round A-33
Multiplying by a Variable Power of 10 .....A-33
ZERO AND ADD (ZAP) A-34
Floating-Point Instructions A-34
ADD NORMALIZED (AD, ADR, AE, AER, AXR) A-34
ADD UNNORMALIZED (AU, AUR, AW, AWR) A-34
COMPARE (CD, CDR, CE, CER) A-35
DIVIDE (DD, DDR, DE, DER) A-35
HALVE (HDR, HER) A-36
MULTIPLY (MD, MDR, ME, MER, MXD, MXDR, MXR) A-36
Floating-Point-Number Conversion A-36
Fixed Point to Floating Point A-37
Floating Point to Fixed Point A-37
Multiprogramming and Multiprocessing Examples A-38
Example of a Program Failure Using OR Immediate A-38
COMPARE AND SWAP (CS, CDS) A-38
Setting a Single Bit A-39
Updating Counters A-39
Bypassing POST AND WAIT A-40
BYPASS POST Routine A-40
BYPASS WAIT Routine A-40
LOCK/UNLOCK A-40
LOCK/UNLOCK with LIFO Queuing for Contentions A-41
LOCK/UNLOCK with FIFO Queuing for Contentions A-42
Free-Pool Manipulation A-44

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 designate 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
                              complement
                              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 posi-
                   tion and carry out.
```


LzLabs GmbH. Ex. 1016-460

```
3. +35 = 0010 0011
   -57 = 1100 0111
   ---------------
   -22 = 1110 1010  Sign change only --
                    no carry into left-
                    most position and no
                    carry out.

4. -57 = 1100 0111
   -35 = 1101 1101
   ---------------
   -92 = 1010 0100  No overflow -- carry
                    into leftmost posi-
                    tion and carry out.

5. +57 = 0011 1001
   +92 = 0101 1100
   ---------------
  +149 =*1001 0101  *Overflow -- carry
                    into leftmost posi-
                    tion, no carry out.

6. -57 = 1100 0111
   -92 = 1010 0100
   ---------------
  -149 =*0110 1011  *Overflow -- no carry
                    into leftmost posi-
                    tion but carry out.
```

The presence or absence of an overflow condition may be recognized from the carries:

- There is no overflow:
  a. If there is no carry into the leftmost bit position and no carry out (examples 1 and 3).
  b. If there is a carry into the leftmost position and also a carry out (examples 2 and 4).
- There is an overflow:
  a. If there is a carry into the leftmost position but no carry out (example 5).
  b. 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).

```
 2^15-1 =  32,767 = 0 111 1111 1111 1111
 2^0    =       1 = 0 000 0000 0000 0001
 0      =       0 = 0 000 0000 0000 0000
-2^0    =      -1 = 1 111 1111 1111 1111
-2^15   = -32,768 = 1 000 0000 0000 0000
```

The following figure 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.

```
  2^31-1 =  2 147 483 647 = 0 111 1111 1111 1111 1111 1111 1111 1111
  2^16   =     65 536     = 0 000 0000 0000 0001 0000 0000 0000 0000
  2^0    =          1     = 0 000 0000 0000 0000 0000 0000 0000 0001
  0      =          0     = 0 000 0000 0000 0000 0000 0000 0000 0000
 -2^0    =         -1     = 1 111 1111 1111 1111 1111 1111 1111 1111
 -2^1    =         -2     = 1 111 1111 1111 1111 1111 1111 1111 1110
 -2^16   =    -65 536     = 1 111 1111 1111 1111 0000 0000 0000 0000
 -2^31+1 = -2 147 483 647 = 1 000 0000 0000 0000 0000 0000 0000 0001
 -2^31   = -2 147 483 648 = 1 000 0000 0000 0000 0000 0000 0000 0000
```

32-Bit Signed Binary Integers

LzLabs GmbH. Ex. 1016-461

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


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.

The following figure illustrates several 32-bit unsigned binary integers arranged in descending order.

$2^{32}-1$	=	4 294 967 295	= 1111 1111 1111 1111 1111 1111 1111 1111
2^{31}	=	2 147 483 648	= 1000 0000 0000 0000 0000 0000 0000 0000
$2^{31}-1$	=	2 147 483 647	= 0111 1111 1111 1111 1111 1111 1111 1111
2^{16}	=	65 536	= 0000 0000 0000 0001 0000 0000 0000 0000
2^{0}	=	1	= 0000 0000 0000 0000 0000 0000 0000 0001
0	=	0	= 0000 0000 0000 0000 0000 0000 0000 0000

32-Bit Unsigned Binary Integers

LzLabs GmbH. Ex. 1016-462

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 or 12 3F	F1 F2 C3 or F1 F2 F3
-4321	04 32 1D	F4 F3 F2 D1
+000050	00 00 05 0C or 00 00 05 0F	F0 F0 F0 F0 F5 C0 or F0 F0 F0 F0 F5 F0
-7	7D	D7
00000	00 00 0C or 00 00 0F	F0 F0 F0 F0 C0 or F0 F0 F0 F0 F0

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)
```

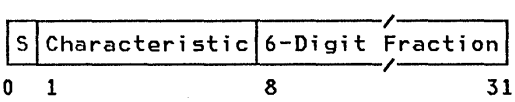
because the product sign follows the algebraic rule of signs even when the value is zero. A negative zero, however, is equivalent to a positive zero in that they compare equal in a decimal comparison.

FLOATING-POINT NUMBERS

A floating-point 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.

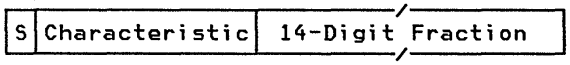
The part of a floating-point number which represents the significant digits of the number is called the fraction. A second part specifies the power (exponent) to which 16 is raised and indicates the location of the radix point of the number. The fraction and exponent may be represented by 32 bits (short format), 64 bits (long format), or 128 bits (extended format).

Short Floating-Point Number



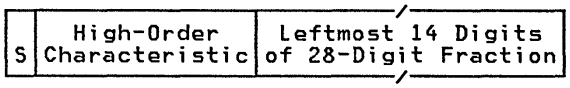
LzLabs GmbH. Ex. 1016-463

Long Floating-Point Number

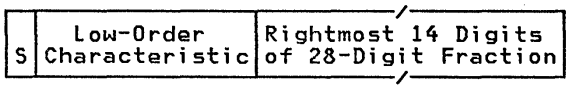


Extended Floating-Point Number

High-Order Part



Low-Order Part



A floating-point 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:

$$16^{-65} \leq M < 16^{63}$$

In decimal terms:

16^{-65} is approximately 5.4×10^{-79}

16^{63} is approximately 7.2×10^{75}

More precisely,

In the short format:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \leq M \leq (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \leq M \leq (1 - 16^{-28}) \times 16^{63}$$

Within a given fraction length (6, 14, or 28 digits), a floating-point operation will provide the greatest precision 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 floating-point 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.

The following figure illustrates sample normalized short floating-point numbers. The last two numbers represent the smallest and the largest positive normalized numbers.

1.0	$= +1/16 \times 16^{1}$	=	0 100 0001 0001 0000 0000 0000 0000 0000 0000{2}
0.5	$= +8/16 \times 16^{0}$	=	0 100 0000 1000 0000 0000 0000 0000 0000 0000{2}
1/64	$= +4/16 \times 16^{-1}$	=	0 011 1111 0100 0000 0000 0000 0000 0000 0000{2}
0.0	$= +0 \times 16^{-64}$	=	0 000 0000 0000 0000 0000 0000 0000 0000 0000{2}
-15.0	$= -15/16 \times 16^{1}$	=	1 100 0001 1111 0000 0000 0000 0000 0000 0000{2}
5.4×10^{-79}	$\sim +1/16 \times 16^{-64}$	=	0 000 0000 0001 0000 0000 0000 0000 0000 0000{2}
7.2×10^{75}	$\sim (1-16^{-6}) \times 16^{63}$	=	0 111 1111 1111 1111 1111 1111 1111 1111 1111{2}

[The symbol ~ means "approximately equal."]

Normalized Short Floating-Point Numbers

LzLabs GmbH. Ex. 1016-464

CONVERSION EXAMPLE

Convert the decimal number 59.25 to a short floating-point 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 floating-point numbers are shown in the section "Floating-Point-Number Conversion" later in this appendix.

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 manuals.

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).

In the instruction-use examples, the notation {2}, {10}, or {16} may be used, indicating that the preceding number is binary, decimal, or hexadecimal, respectively.

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.

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

LzLabs GmbH. Ex. 1016-465

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. For the 31-bit addressing mode, the use of the BRANCH AND SAVE (BAS, BASR) instructions is recommended in place of the BRANCH AND LINK (BAL, BALR) instructions illustrated in this appendix.

GENERAL INSTRUCTIONS

(See Chapter 7 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	R_1,D_2(X_2,B_2)
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, NR, NI, NC)

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 0101{2}
Second-operand byte:	0101 1100{2}
Result byte:	0001 0100{2}

And (NI)

A frequent use of the AND instruction is to set a particular bit to zero. For example, assume that storage location 4891 contains 0100 0011{2}. 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	D_1(B_1),I_2
NI	1(8),X'FE'

When this instruction is executed, the byte in storage is ANDed with the imme-

LzLabs GmbH. Ex. 1016-466

diate byte (the I_2 field of the instructions):

```
Location 4891:    0100 0011(2)
Immediate byte:   1111 1110(2)
-------------------------------
Result:           0100 0010(2)
```

The resulting byte, with bit 7 set to zero, is stored back in location 4891. Condition code 1 is set.

BRANCH AND LINK (BAL, BALR)

The BRANCH AND LINK instructions are commonly used to branch to a subroutine with the option of later returning to the main instruction sequence. For example, assume that a branch to a subroutine at storage address 1160 is required. Also assume:

The contents of register 2 are not significant.
Register 5 contains 00 00 11 50.
Address 00 00 C6 contains the BAL instruction, so that 00 00 CA is the address of the next sequential instruction.

The format of the BAL instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
45	2	0	5	010

Assembler Format

```
Op Code  R1,D2(X2,B2)
-------------------------
   BAL   2,X'10'(0,5)
```

After the instruction is executed:

Register 2 (bits 8-31) contains 00 00 CA.
PSW bits 40-63 contain 00 11 60.

Register 2 also contains other information from the PSW in bits 0-7.

The program can return to the main instruction sequence at any time with a BRANCH ON CONDITION (BCR) instruction that specifies register 2 and a mask of 15(10), provided that bits 8-31 of register 2 have not meanwhile been disturbed.

The BALR 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 sequence of statements:

```
BALR    15,0
USING   *,15
```

tells the assembler program that, when the program is executed, the address of the next sequential instruction following the BALR will be placed in register 15, which is to be used as a base register from here on. (The USING statement is an "assembler instruction" and is thus not a part of the object program.)

As another example, BALR 6,0 may be used to preserve the current condition code in bits 2 and 3 of register 6 for future inspection.

Note that, in all three examples, a value of zero in the X_2 or R_2 field indicates that the indexing or branching function, respectively, is not to be performed; it does not refer to register 0. Register 0 can be designated by the R_1 field, however.

BRANCH ON CONDITION (BC, BCR)

The BRANCH ON CONDITION instructions test the condition code to see whether a branch should or should not be taken. The branch is taken 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)
```


LzLabs GmbH. Ex. 1016-467

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.

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).

BRANCH ON COUNT (BCT, BCTR)

The BRANCH ON COUNT instructions are 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	R_1,D_2(X_2,B_2)
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 instruction is:

Machine Format

Op Code	R_1	R_3	B_2	D_2
86	4	6	A	000

Assembler Format

Op Code	R_1,R_3,D_2(B_2)
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. 1016-468

Table	
2 Bytes	2 Bytes
ARG1	FUNCT1
ARG2	FUNCT2
ARG3	FUNCT3
ARG4	FUNCT4
ARG5	FUNCT5
ARG6	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 FFFFFFFC = -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)

This instruction is similar to BRANCH ON INDEX HIGH except that the branch is successful when the sum is low or equal compared to the compare value.

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 instructions differ from the signed-binary comparison instructions (C, CH, CR) in that all quantities are handled as unsigned binary integers or as unstructured data.

Compare Logical (CLC)

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. 1016-469

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	$D_1(L,B_1),D_2(B_2)$
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 A.B.JOHNSON should precede A.C.JOHNSON for the names to be in alphabetic sequence.

Compare Logical (CLI)

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	$D_1(B_1),I_2$
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.

Compare Logical (CLR)

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	R_1,R_2
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 four-bit mask

LzLabs GmbH. Ex. 1016-470

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 10000.
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	R_1,M_3,D_2(B_2)
CLM	6,B'1101',X'200'(12)

causes the following comparison:

```
Register 6:  F0   BC   5C   7B
   Mask M3:   1    1    0    1
              --   --        --

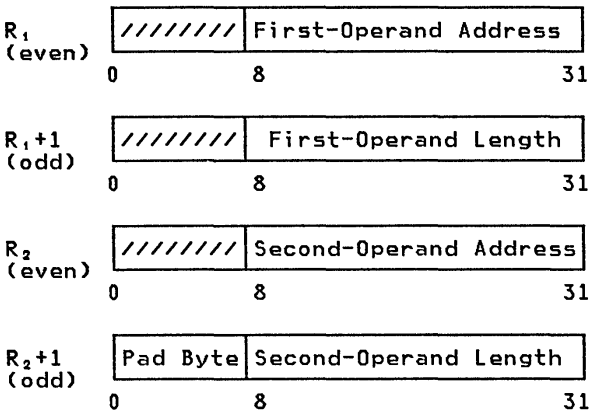
             F0   BC        7B
Storage
locations
10200-10202:      F0   BC   7B
```

Because the selected bytes are equal, condition code 0 is set.

COMPARE LOGICAL LONG (CLCL)

The COMPARE LOGICAL LONG (CLCL) 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:



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:

```
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 regis-
                      ter 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.

LzLabs GmbH. Ex. 1016-471

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	R_1,R_2
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 X'28' and X'48', 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 reissued 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 X'16', 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.

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	$R_1,D_2(X_2,B_2)$
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.

The format of the instruction is:

LzLabs GmbH. Ex. 1016-472

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.

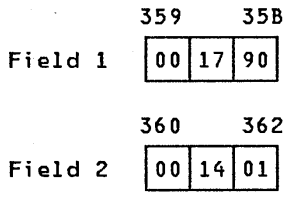
EXCLUSIVE OR (X, XC, XI, XR)

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}

Exclusive Or (XC)

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:



Execution of the instruction (assume that register 7 contains 00 00 03 58):

LzLabs GmbH. Ex. 1016-473

Machine Format

Op Code	L	B_1	D_1	B_1	D_2
D7	02	7	001	7	008

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
XC	1(3,7),8(7)

Field 1 is EXCLUSIVE ORed with field 2 as follows:

```
Field 1:  00000000 00010111 10010000{2}
        = 00 17 90{16}
Field 2:  00000000 00010100 00000001{2}
        = 00 14 01{16}
-----------------------------------------
Result:   00000000 00000011 10010001{2}
        = 00 03 91{16}
```

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	$D_1(L,B_1),D_2(B_2)$
XC	8(3,7),1(7)

produces the following result:

```
Field 1: 00000000 00000011 10010001{2}
       = 00 03 91{16}
Field 2: 00000000 00010100 00000001{2}
       = 00 14 01{16}
-----------------------------------------
Result:  00000000 00010111 10010000{2}
       = 00 17 90{16}
```

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	$D_1(L,B_1),D_2(B_2)$
XC	1(3,7),8(7)

produces the following result:

```
Field 1: 00000000 00000011 10010001{2}
       = 00 03 91{16}
Field 2: 00000000 00010111 10010000{2}
       = 00 17 90{16}
-----------------------------------------
Result:  00000000 00010100 00000001{2}
       = 00 14 01{16}
```

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.

Exclusive Or (XI)

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	$D_1(B_1),I_2$
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 1001{2}
Immediate byte:   1000 0001{2}
-----------------------------
Result:           1110 1000{2}
```


LzLabs GmbH. Ex. 1016-474

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 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 the section "Floating-Point-Number Conversion" later in this appendix.

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	$D_1(L,B_1),D_2(B_2)$
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	$R_1,D_2(X_2,B_2)$
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 0000(2) = 00
Register byte:      0000 0011(2) = 03
-------------------------------------
Result:             0000 0011(2) = 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	$D_1(L,B_1),D_2(B_2)$
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

LzLabs GmbH. Ex. 1016-475

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	* * * *

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 instructions take 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

LzLabs GmbH. Ex. 1016-476

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:

Machine Format

Op Code	R_1	X_2	B_2	D_2
41	5	0	5	00A

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
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- 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	$R_1,D_2(X_2,B_2)$
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)

Move (MVC)

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:

```
        2048                                    2052
Field   +--+--+--+--+--+--+--+--+--+--+--+
  1     |C1|C2|C3|C4|C5|C6|C7|C8|C9|CA|CB|
        +--+--+--+--+--+--+--+--+--+--+--+

        3840                          3848
Field   +--+--+--+--+--+--+--+--+--+
  2     |F1|F2|F3|F4|F5|F6|F7|F8|F9|
        +--+--+--+--+--+--+--+--+--+
```

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	$D_1(L,B_1),D_2(B_2)$
MVC	0(8,1),0(2)

After the instruction is executed, field 1 becomes:

LzLabs GmbH. Ex. 1016-477

	2048										2052
Field 1	F1	F2	F3	F4	F5	F6	F7	F8	C9	CA	CB

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:

358								360
00	F1	F2	F3	F4	F5	F6	F7	F8

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

Op Code	L	B_1	D_1	B_2	D_2
D2	07	B	001	B	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
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.

Move (MVI)

The MOVE (MVI) instruction places one byte of information from the instruction stream into storage. For example, the instruction:

Machine Format

Op Code	I_2	B_1	D_1
92	5B	1	000

Assembler Format

Op Code	$D_1(B_1),I_2$
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 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. 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.

LzLabs GmbH. Ex. 1016-478

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 registers 2 and 8 would have been incremented by 400{16}, and bits 0-7 of registers 2 and 8 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.

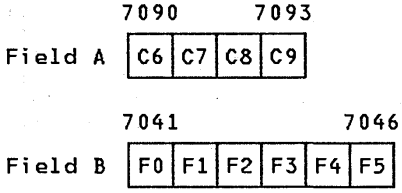
The technique for setting a field to zeros that is illustrated in the second example of MVC cannot be used with MVCL. If the registers were set up to attempt such an operation with MVCL, no data movement would take place and condition code 3 would indicate destructive overlap.

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 larger 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:

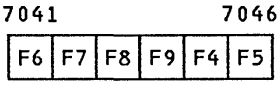
Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D1	03	F	001	E	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
MVN	1(4,15),0(14)

is executed, field B becomes:



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 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:

LzLabs GmbH. Ex. 1016-479

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	$D_1(L_1,B_1),D_2(L_2,B_2)$
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.

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.

LzLabs GmbH. Ex. 1016-480

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.
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, OR, OI, OC)

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 0101{2}
Second-operand byte:	0101 1100{2}
Result byte:	0111 1101{2}

Or (OI)

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 0010{2}
Immediate byte:	0000 0001{2}
Result:	0100 0011{2}

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:

LzLabs GmbH. Ex. 1016-481

	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.

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 10011000{2}
```

The instruction:

Machine Format

Op Code	R_1		B_2	D_2
8F	2	////	0	01F

Assembler Format

Op Code	R_1,D_2(B_2)
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 00000000{2}
```

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
01110010{2}
```

The instruction:

Machine Format

Op Code	R_1		B_2	D_2
8B	2	////	0	008

Assembler Format

Op Code	R_1,D_2(B_2)
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
00000000{2}
```

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:

LzLabs GmbH. Ex. 1016-482

Machine Format

Op Code	R_1	M_3	S_2
BE	8	7	* * * *

Register Format

Op Code	R_1,M_3,S_2
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	R_1,M_3,S_2
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

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 1011{2}
Mask = 1100 0011{2}
-------------------
Test = 11xx xx11{2}
```

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:

LzLabs GmbH. Ex. 1016-483

```
B9   = 1011 1001{2}
Mask = 1100 0011{2}
-------------------
Test = 10xx xx01{2}
```

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 1100{2}
Mask = 1100 0011{2}
-------------------
Test = 00xx xx00{2}
```

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

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
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.

The figure "Translate-and-Test Table" that follows 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.

LzLabs GmbH. Ex. 1016-484

	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.

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.

```
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	$D_1(L,B_1),D_2(B_2)$
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:

LzLabs GmbH. Ex. 1016-485

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:

1. The address of the source byte, 00CA84, is placed in the rightmost 24 bits of register 1.
2. The table entry, 04, is placed in the rightmost eight bits of register 2, which now contains 00 00 30 04.
3. 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 R1 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	$D_1(L_1,B_1),D_2(L_2,B_2)$
UNPK	0(5,12),1(2,13)

is executed, the storage locations 1000-1004 contain F0 F0 F1 F2 D3.

DECIMAL INSTRUCTIONS

(See Chapter 8 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:

LzLabs GmbH. Ex. 1016-486

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	$D_1(L_1,B_1),D_2(L_2,B_2)$
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	$D_1(L_1,B_1),D_2(L_2,B_2)$
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

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:

	2000				2004
Locations 2000-2004	38	46	0D	01	8C
	quotient		remainder		

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

LzLabs GmbH. Ex. 1016-487

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

1200 1203

02	57	42	6C

↑ +

Pattern

1000 100C

40	20	20	6B	20	21	20	4B	20	20	40	C3	D9
b	d	d	,	d	(	d	.	d	d	b	C	R

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:

LzLabs GmbH. Ex. 1016-488

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.

Thus, after the instruction is executed, the pattern field contains the result as follows:

Pattern

1000 100C

40	40	F2	6B	F5	F7	F4	4B	F2	F6	40	40	40
b	b	2	,	5	7	4	.	2	6	b	b	b

When printed, the new pattern field appears 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

1000 100C

40	40	40	40	40	40	F0	4B	F2	F6	40	C3	D9
b	b	b	b	b	b	0	.	2	6	b	C	R

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 signif-
                        icant digit in GR1.
BCTR 1,0                Subtract 1 from
                        address in GR1.
MVI  0(1),C'$'          Store dollar sign
                        at address in GR1.
```


LzLabs GmbH. Ex. 1016-489

produce the following results for the two examples under EDIT:

Pattern

1000												100C
40	5B	F2	6B	F5	F7	F4	4B	F2	F6	40	40	40
b	$	2	,	5	7	4	.	2	6	b	b	b

This pattern field prints as:

$2,574.26

Condition code 2 is set to indicate that the number edited was greater than zero.

Pattern

1000												100C
40	40	40	40	40	5B	F0	4B	F2	F6	40	C3	D9
b	b	b	b	b	$	0	.	2	6	b	C	R

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).

	1202	1204
Multiplicand	38	46 0D

	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	$D_1(L_1,B_1),D_2(L_2,B_2)$
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	$S_1(L_1),S_2,I_3$
SRP	FIELD1(6),3,0

FIELD1 (before): 00 01 23 45 67 8C

FIELD1 (after): 12 34 56 78 00 0C

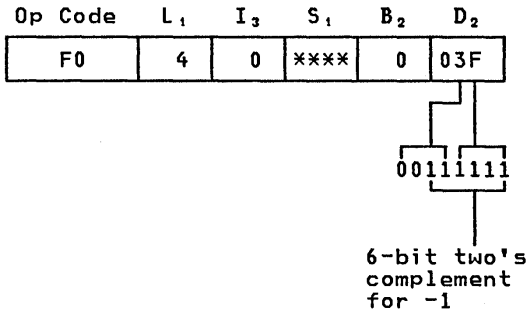
The second-operand address in this instruction specifies the shift amount (three places). The rounding digit, I_3, is not used in a left shift, but it must be a valid decimal digit. After execution, condition code 2 is set to show that the result is greater than zero.

LzLabs GmbH. Ex. 1016-490

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



Assembler Format

Op Code	$S_1(L_1),S_2,I_3$
SRP	FIELD2(5),64-1,0

FIELD 2 (before): 01 23 45 67 8C

FIELD 2 (after): 00 12 34 56 7C

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.

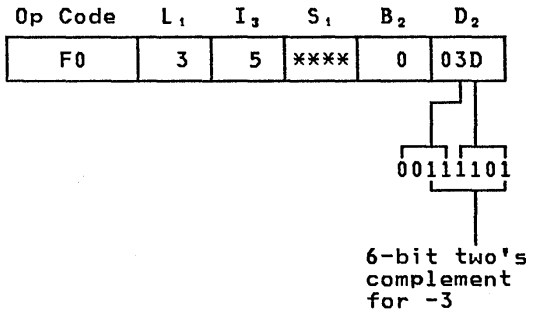
The six-bit two's complement of a number, n, can be specified as 64 - n. In this example, a right shift of one is represented as 64 - 1.

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	$S_1(L_1),S_2,I_3$
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 designated 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.

LzLabs GmbH. Ex. 1016-491

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

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F8	4	2	9	000	9	500

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
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.

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.

FLOATING-POINT INSTRUCTIONS

(See Chapter 9 for a complete description of the 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 instructions perform the addition of two floating-point numbers and place 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 floating-point 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 instructions operate the same as the ADD NORMALIZED instructions, 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:

LzLabs GmbH. Ex. 1016-492

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:

	Fraction	GD[1]
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	R_1	R_2
29	4	6

Assembler Format

Op Code	R_1,R_2
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 floating-point numbers with different characteristics or fractions may compare equal if the numbers are unnormalized or zero.

As another example of comparing unnormalized floating-point 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 floating-point 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	R_1	R_2
3D	2	0

Assembler Format

Op Code	R_1,R_2
DER	2,0

Several examples of short-precision floating-point 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. 1016-493

```
                  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 floating-point 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, ME, MER, 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

FLOATING-POINT-NUMBER CONVERSION

The following examples illustrate one method of converting between binary fixed-point numbers (32-bit signed binary integers) and normalized floating-point numbers. Conversion must provide for the different representations used

LzLabs GmbH. Ex. 1016-494

with negative numbers: the two's-complement form for signed binary integers, and the signed-absolute-value form for the fractions of floating-point numbers.

Fixed Point to 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 floating-point 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 floating-point 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:
                      7FFF FFC5

ST  9,TEMP+4          TEMP:
                      xxxx xxxx 7FFF FFC5

MVC TEMP(4),TWO31     TEMP:
                      4E00 0000 7FFF FFC5

LD  2,TEMP            FPR2:
                      4E00 0000 7FFF FFC5

SD  2,TWO31           FPR2:
                      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 floating-point 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 floating-point form and normalizing the result.

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 floating-point 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 floating-point 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):

```
                          Result
SD  6,TWO31R          FPR6:
                      C87F FFFF C500 0000
BC  11,OVERFLOW       Branch to overflow
                        routine if result
                        is greater than or
                        equal to zero
AW  6,TWO32           FPR6:
                      4E00 0000 8000 003B
BC  4,OVERFLOW        Branch to overflow
                        routine if result
                        is less than zero
STD 6,TEMP            TEMP:
                      4E00 0000 8000 003B
XI  TEMP+4,X'80'      TEMP:
                      4E00 0000 0000 003B
L   8,TEMP+4          GR8:
                      0000 003B
```


LzLabs GmbH. Ex. 1016-495

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 running 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.

COMPARE AND SWAP (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 running enabled for interruption (multiprogramming) or by programs that are running on a multiprocessing configuration. That is, the routine allows a program to modify the contents of a storage location while running 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 CS 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 regis-

LzLabs GmbH. Ex. 1016-496

ter, in preparation for the program to loop back and try again. During the execution of CS, no other CPU can access the specified location.

Setting a Single Bit

The following instruction sequence shows how the CS 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 un-
                         changed, or re-
                         fetch current
                         flag values if
                         changed
      BC   4,RETRY     If new flags are not
                         stored, try again
```

The format of the CS instruction is:

Machine Format

Op Code	R_1	R_3	S_2
BA	7	8	****

Assembler Format

Op Code	R_1,R_3,S_2
CS	7,8,WORD

The CS 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 CS instruction is permitted to access 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 CS instruction indicated an unsuccessful comparison (condition code 1). When the CS 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 CS 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 CS 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 CS instruction, the CS instruction would have loaded the new counter value into general register 7 and set the condition code to 1, indicating an unsuccessful update. The program then must update the new counter value in general register 7 and retry the CS instruction, retesting the condition code, and retrying until a successful update is completed.

The following instruction sequence performs the above procedure:

LzLabs GmbH. Ex. 1016-497

```
     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 GR7	CPU A GR8	CNTR	CPU B GR7	CPU B GR8	Comments
		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 OS/VS to be bypassed whenever the corresponding WAIT has not yet been issued, provided that the supervisor WAIT and POST routines use COMPARE AND SWAP to manipulate event control blocks (ECBs).

Initial Conditions:

GR1 contains the address of the ECB.
GR0 contains the POST code.

```
HSPOST  L    3,0(1)      GR3 = contents
                           of ECB
        LTR  3,3         ECB marked
                           'waiting'?
        BC   4,PSVC      Yes, issue 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 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 al-
                           ready 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 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.

LzLabs GmbH. Ex. 1016-498

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.

LzLabs GmbH. Ex. 1016-499

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.

The following routines allow enabled code to perform the actions described in the previous chart.

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 negative 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
                           updated?
        WAIT   ECB=(1)   Yes, wait for the
                           resource;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 neg-
        BC     4,B         ative value?
        L      0,4(1)    No, load the
        CS     1,0,0(2)    pointer from 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
        CS     1,0,0(2)    a negative value;
                           free the header
        BC     7,A         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, either the entire new or the entire old value of the word is obtained, and not a combination of the two. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

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

LzLabs GmbH. Ex. 1016-500

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, 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 the section "Free-Pool Manipulation" in this appendix.

The following chart describes the action taken for FIFO LOCK and FIFO UNLOCK.

Function	Action
FIFO LOCK (the incoming element is at location A)	Store address A into the header. 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 ele-
                              ment 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
        BC      7,TRYAGN      while remember-
                              ing old contents
                              of header into
                              GR1; GR1 now
                              contains address
                              of removed
                              element
        LR      4,1         Removed element
                              becomes new cur-
                              rently owned
                              element
        HSWAIT              Perform bypass-
                              wait routine; if
                              ECB already
                              posted, con-
                              tinue; if not,
                              wait; GR1 con-
                              tains 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.

LzLabs GmbH. Ex. 1016-501

```
FUNLK     LR    1,2   Place address of en-
                      tered element in
                      GR1; GR1 = address
                      of ECB to be posted
          SR    0,0   GR0 = 0; GR0 has a
                      post code of zero
          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 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 CS 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 CS 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 $2^{32}-1$ 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.

LzLabs GmbH. Ex. 1016-502

Function	Action	
	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 ele-
                       ment 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 = con-
                           tents 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
                           element
       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 removed
                           element is in
                           GR2
```

Note that the LM instructions at locations ADDQ and DELETQ would have to be CDS instructions if it were not for the rule concerning storage-operand consistency. This rule requires the LM instructions to fetch an eight-byte operand aligned on a doubleword boundary such that, if another CPU changes the doubleword being fetched, either the entire new or the entire old value of the doubleword is obtained, and not a combination of the two. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

LzLabs GmbH. Ex. 1016-503

APPENDIX B. LISTS OF INSTRUCTIONS

The following three 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 code 00 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" 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
- $ 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 Access exceptions and ASN-translation-specification exception; see instruction description for details
- AT ASN-translation exceptions (which include addressing, ASN-translation specification, AFX translation, and ASX translation)
- B PER branch event
- C Condition code is set
- D 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
- EO Exponent-overflow exception
- EU Exponent-underflow exception
- EX Execute exception
- FK Floating-point-divide exception
- 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
- GM Instruction execution includes the implied use of multiple general registers
- GS Instruction execution includes the implied use of general register 1 as the SID
- IF Fixed-point-overflow exception
- II Interruptible instruction
- IK Fixed-point-divide exception
- L New condition code is loaded
- LS Significance exception
- MO Monitor event
- OP Operand exception
- P Privileged-operation exception
- Q Privileged-operation exception for semiprivileged instructions
- R PER general-register alteration event
- RR RR instruction format
- RRE RRE instruction format
- RS RS instruction format
- RX RX instruction format
- S S instruction format
- SI SI instruction format
- SO Special-operation exception
- SP Specification exception
- SS SS instruction format
- SSE SSE instruction format
- ST PER storage-alteration event
- T Trace exceptions (which include trace table, addressing, and low-address protection)
- Z1 Additional exceptions and events for PROGRAM CALL (which include addressing, EX translation, LX translation, PC-translation specification, and special-operation exceptions and space-switch event)
- Z2 Additional exceptions and events for PROGRAM TRANSFER (which include addressing, primary-authority, and special-operation exceptions and space-switch event)
- Z3 Additional exceptions for SET SECONDARY ASN (which include addressing, secondary authority, and special operation)

LzLabs GmbH. Ex. 1016-505

Name	Mnemonic	Characteristics					Op Code	Page No.
ADD	AR	RR C		IF		R	1A	7-8
ADD	A	RX C	A	IF		R	5A	7-8
ADD DECIMAL	AP	SS C	A	D DF		ST	FA	8-5
ADD HALFWORD	AH	RX C	A	IF		R	4A	7-8
ADD LOGICAL	ALR	RR C				R	1E	7-9
ADD LOGICAL	AL	RX C	A			R	5E	7-9
ADD NORMALIZED (extended)	AXR	RR C	SP	EU EO	LS		36	9-6
ADD NORMALIZED (long)	ADR	RR C	SP	EU EO	LS		2A	9-6
ADD NORMALIZED (long)	AD	RX C	A SP	EU EO	LS		6A	9-6
ADD NORMALIZED (short)	AER	RR C	SP	EU EO	LS		3A	9-6
ADD NORMALIZED (short)	AE	RX C	A SP	EU EO	LS		7A	9-6
ADD UNNORMALIZED (long)	AWR	RR C	SP	EO	LS		2E	9-7
ADD UNNORMALIZED (long)	AW	RX C	A SP	EO	LS		6E	9-7
ADD UNNORMALIZED (short)	AUR	RR C	SP	EO	LS		3E	9-7
ADD UNNORMALIZED (short)	AU	RX C	A SP	EO	LS		7E	9-7
AND	NR	RR C				R	14	7-9
AND	N	RX C	A			R	54	7-9
AND (character)	NC	SS C	A			ST	D4	7-9
AND (immediate)	NI	SI C	A			ST	94	7-9
BRANCH AND LINK	BALR	RR		T		B R	05	7-10
BRANCH AND LINK	BAL	RX				B R	45	7-10
BRANCH AND SAVE	BASR	RR		T		B R	0D	7-10
BRANCH AND SAVE	BAS	RX				B R	4D	7-10
BRANCH AND SAVE AND SET MODE	BASSM	RR		T		B R	0C	7-11
BRANCH AND SET MODE	BSM	RR				B R	0B	7-11
BRANCH ON CONDITION	BCR	RR		¢[1]		B	07	7-12
BRANCH ON CONDITION	BC	RX				B	47	7-12
BRANCH ON COUNT	BCTR	RR				B R	06	7-13
BRANCH ON COUNT	BCT	RX				B R	46	7-13
BRANCH ON INDEX HIGH	BXH	RS				B R	86	7-13
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS				B R	87	7-13
CLEAR SUBCHANNEL	CSCH	S C	P	OP ¢	GS		B230	14-3
COMPARE	CR	RR C					19	7-14
COMPARE	C	RX C	A				59	7-14
COMPARE (long)	CDR	RR C	SP				29	9-8
COMPARE (long)	CD	RX C	A SP				69	9-8
COMPARE (short)	CER	RR C	SP				39	9-8
COMPARE (short)	CE	RX C	A SP				79	9-8
COMPARE AND SWAP	CS	RS C	A SP	$		R ST	BA	7-14
COMPARE DECIMAL	CP	SS C	A	D			F9	8-5
COMPARE DOUBLE AND SWAP	CDS	RS C	A SP	$		R ST	BB	7-14
COMPARE HALFWORD	CH	RX C	A				49	7-16
COMPARE LOGICAL	CLR	RR C					15	7-16
COMPARE LOGICAL	CL	RX C	A				55	7-16
COMPARE LOGICAL (character)	CLC	SS C	A				D5	7-16
COMPARE LOGICAL (immediate)	CLI	SI C	A				95	7-16
COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS C	A				BD	7-17
COMPARE LOGICAL LONG	CLCL	RR C	A SP	II		R	0F	7-17
CONVERT TO BINARY	CVB	RX	A	D	IK	R	4F	7-19
CONVERT TO DECIMAL	CVD	RX	A			ST	4E	7-20
DIAGNOSE		DM	P DM				83	10-4
DIVIDE	DR	RR	SP		IK	R	1D	7-20
DIVIDE	D	RX	A SP		IK	R	5D	7-20
DIVIDE (extended)	DXR	RRE	SP	EU EO	FK		B22D	9-8
DIVIDE (long)	DDR	RR	SP	EU EO	FK		2D	9-8

Instructions Arranged by Name (Part 1 of 4)

LzLabs GmbH. Ex. 1016-506

Name	Mnemonic	Characteristics											Op Code	Page No.
DIVIDE (long)	DD	RX			A		SP	EU	EO	FK			6D	9-8
DIVIDE (short)	DER	RR					SP	EU	EO	FK			3D	9-8
DIVIDE (short)	DE	RX			A		SP	EU	EO	FK			7D	9-8
DIVIDE DECIMAL	DP	SS			A		SP	D		DK		ST	FD	8-5
EDIT	ED	SS	C		A			D				ST	DE	8-6
EDIT AND MARK	EDMK	SS	C		A			D			G1	R ST	DF	8-9
EXCLUSIVE OR	XR	RR	C									R	17	7-21
EXCLUSIVE OR	X	RX	C		A							R	57	7-21
EXCLUSIVE OR (character)	XC	SS	C		A							ST	D7	7-21
EXCLUSIVE OR (immediate)	XI	SI	C		A							ST	97	7-21
EXECUTE	EX	RX			AI		SP				EX		44	7-22
EXTRACT PRIMARY ASN	EPAR	RRE		Q				SO				R	B226	10-5
EXTRACT SECONDARY ASN	ESAR	RRE		Q				SO				R	B227	10-5
HALT SUBCHANNEL	HSCH	S	C	P				OP		♦	GS		B231	14-4
HALVE (long)	HDR	RR					SP	EU					24	9-10
HALVE (short)	HER	RR					SP	EU					34	9-10
INSERT ADDRESS SPACE CONTROL	IAC	RRE	C	Q				SO				R	B224	10-6
INSERT CHARACTER	IC	RX			A							R	43	7-23
INSERT CHARACTERS UNDER MASK	ICM	RS	C		A							R	BF	7-23
INSERT PROGRAM MASK	IPM	RRE										R	B222	7-23
INSERT PSW KEY	IPK	S		Q							G2	R	B20B	10-7
INSERT STORAGE KEY EXTENDED	ISKE	RRE		P									B229	10-7
INSERT VIRTUAL STORAGE KEY	IVSK	RRE		Q	A[1]			SO				R	B223	10-7
INVALIDATE PAGE TABLE ENTRY	IPTE	RRE		P	A[1]					$			B221	10-8
LOAD	LR	RR										R	18	7-24
LOAD	L	RX			A							R	58	7-24
LOAD (long)	LDR	RR					SP						28	9-10
LOAD (long)	LD	RX			A		SP						68	9-10
LOAD (short)	LER	RR					SP						38	9-10
LOAD (short)	LE	RX			A		SP						78	9-10
LOAD ADDRESS	LA	RX										R	41	7-24
LOAD ADDRESS SPACE PARAMETERS	LASP	SSE	C	P		AS	SP	SO					E500	10-10
LOAD AND TEST	LTR	RR	C									R	12	7-24
LOAD AND TEST (long)	LTDR	RR	C				SP						22	9-11
LOAD AND TEST (short)	LTER	RR	C				SP						32	9-11
LOAD COMPLEMENT	LCR	RR	C						IF			R	13	7-24
LOAD COMPLEMENT (long)	LCDR	RR	C				SP						23	9-11
LOAD COMPLEMENT (short)	LCER	RR	C				SP						33	9-11
LOAD CONTROL	LCTL	RS		P	A		SP						B7	10-17
LOAD HALFWORD	LH	RX			A							R	48	7-25
LOAD MULTIPLE	LM	RS			A							R	98	7-25
LOAD NEGATIVE	LNR	RR	C									R	11	7-25
LOAD NEGATIVE (long)	LNDR	RR	C				SP						21	9-11
LOAD NEGATIVE (short)	LNER	RR	C				SP						31	9-11
LOAD POSITIVE	LPR	RR	C						IF			R	10	7-26
LOAD POSITIVE (long)	LPDR	RR	C				SP						20	9-12
LOAD POSITIVE (short)	LPER	RR	C				SP						30	9-12
LOAD PSW	LPSW	S	L	P	A		SP			♦			82	10-17
LOAD REAL ADDRESS	LRA	RX	C	P	A[1]							R	B1	10-18
LOAD ROUNDED (extended to long)	LRDR	RR					SP		EO				25	9-12
LOAD ROUNDED (long to short)	LRER	RR					SP		EO				35	9-12
MODIFY SUBCHANNEL	MSCH	S	C	P	A		SP	OP		♦	GS		B232	14-6
MONITOR CALL	MC	SI					SP				MO		AF	7-26
MOVE (character)	MVC	SS			A							ST	D2	7-27
MOVE (immediate)	MVI	SI			A							ST	92	7-27

Instructions Arranged by Name (Part 2 of 4)

LzLabs GmbH. Ex. 1016-507

Name	Mnemonic	Characteristics				Op Code	Page No.
MOVE LONG	MVCL	RR C	A SP	II	R ST	0E	7-27
MOVE NUMERICS	MVN	SS	A		ST	D1	7-30
MOVE TO PRIMARY	MVCP	SS C	Q A[1]	SO ϕ	ST	DA	10-19
MOVE TO SECONDARY	MVCS	SS C	Q A[1]	SO ϕ	ST	DB	10-19
MOVE WITH KEY	MVCK	SS C	Q A		ST	D9	10-20
MOVE WITH OFFSET	MVO	SS	A		ST	F1	7-31
MOVE ZONES	MVZ	SS	A		ST	D3	7-31
MULTIPLY	MR	RR	SP		R	1C	7-32
MULTIPLY	M	RX	A SP		R	5C	7-32
MULTIPLY (extended)	MXR	RR	SP	EU EO		26	9-13
MULTIPLY (long to extended)	MXDR	RR	SP	EU EO		27	9-13
MULTIPLY (long to extended)	MXD	RX	A SP	EU EO		67	9-13
MULTIPLY (long)	MDR	RR	SP	EU EO		2C	9-13
MULTIPLY (long)	MD	RX	A SP	EU EO		6C	9-13
MULTIPLY (short to long)	MER	RR	SP	EU EO		3C	9-13
MULTIPLY (short to long)	ME	RX	A SP	EU EO		7C	9-13
MULTIPLY DECIMAL	MP	SS	A SP	D	ST	FC	8-10
MULTIPLY HALFWORD	MH	RX	A		R	4C	7-32
OR	OR	RR C			R	16	7-33
OR	O	RX C	A		R	56	7-33
OR (character)	OC	SS C	A		ST	D6	7-33
OR (immediate)	OI	SI C	A		ST	96	7-33
PACK	PACK	SS	A		ST	F2	7-33
PROGRAM CALL	PC	S	Q AT	Z[1] T ϕ GM	B R	B218	10-22
PROGRAM TRANSFER	PT	RRE	Q AT SP	Z[2] T ϕ	B	B228	10-28
PURGE TLB	PTLB	S	P	$		B20D	10-33
RESET CHANNEL PATH	RCHP	S C	P	OP ϕ G1		B23B	14-6
RESET REFERENCE BIT EXTENDED	RRBE	RRE C	P			B22A	10-33
RESUME SUBCHANNEL	RSCH	S C	P	OP ϕ GS		B238	14-8
SET ADDRESS LIMIT	SAL	S	P	OP ϕ G1		B237	14-9
SET ADDRESS SPACE CONTROL	SAC	S	SP	SO ϕ		B219	10-33
SET CHANNEL MONITOR	SCHM	S	P	OP ϕ GM		B23C	14-10
SET CLOCK	SCK	S C	P A SP			B204	10-34
SET CLOCK COMPARATOR	SCKC	S	P A SP			B206	10-35
SET CPU TIMER	SPT	S	P A SP			B208	10-35
SET PREFIX	SPX	S	P A SP	$		B210	10-36
SET PROGRAM MASK	SPM	RR L				04	7-34
SET PSW KEY FROM ADDRESS	SPKA	S	Q			B20A	10-36
SET SECONDARY ASN	SSAR	RRE	AT	Z[3] T ϕ		B225	10-37
SET STORAGE KEY EXTENDED	SSKE	RRE	P	ϕ		B22B	10-40
SET SYSTEM MASK	SSM	S	P A SP	SO		80	10-40
SHIFT AND ROUND DECIMAL	SRP	SS C	A	D DF	ST	F0	8-10
SHIFT LEFT DOUBLE	SLDA	RS C	SP	IF	R	8F	7-34
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS	SP		R	8D	7-35
SHIFT LEFT SINGLE	SLA	RS C		IF	R	8B	7-35
SHIFT LEFT SINGLE LOGICAL	SLL	RS			R	89	7-36
SHIFT RIGHT DOUBLE	SRDA	RS C	SP		R	8E	7-36
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS	SP		R	8C	7-36
SHIFT RIGHT SINGLE	SRA	RS C			R	8A	7-37
SHIFT RIGHT SINGLE LOGICAL	SRL	RS			R	88	7-37
SIGNAL PROCESSOR	SIGP	RS C	P	$	R	AE	10-41
START SUBCHANNEL	SSCH	S C	P A SP	OP ϕ GS		B233	14-11
STORE	ST	RX	A		ST	50	7-37
STORE (long)	STD	RX	A SP		ST	60	9-14
STORE (short)	STE	RX	A SP		ST	70	9-14

Instructions Arranged by Name (Part 3 of 4)

LzLabs GmbH. Ex. 1016-508

Name	Mnemonic	Characteristics									Op Code	Page No.
STORE CHANNEL PATH STATUS	STCPS	S		P	A	SP		¢		ST	B23A	14-13
STORE CHANNEL REPORT WORD	STCRW	S	C	P	A	SP		¢		ST	B239	14-13
STORE CHARACTER	STC	RX			A					ST	42	7-38
STORE CHARACTERS UNDER MASK	STCM	RS			A					ST	BE	7-38
STORE CLOCK	STCK	S	C		A			$		ST	B205	7-38
STORE CLOCK COMPARATOR	STCKC	S		P	A	SP				ST	B207	10-42
STORE CONTROL	STCTL	RS		P	A	SP				ST	B6	10-42
STORE CPU ADDRESS	STAP	S		P	A	SP				ST	B212	10-43
STORE CPU ID	STIDP	S		P	A	SP				ST	B202	10-43
STORE CPU TIMER	STPT	S		P	A	SP				ST	B209	10-43
STORE HALFWORD	STH	RX			A					ST	40	7-39
STORE MULTIPLE	STM	RS			A					ST	90	7-39
STORE PREFIX	STPX	S		P	A	SP				ST	B211	10-44
STORE SUBCHANNEL	STSCH	S	C	P	A	SP	OP	¢	GS	ST	B234	14-14
STORE THEN AND SYSTEM MASK	STNSM	SI		P	A					ST	AC	10-44
STORE THEN OR SYSTEM MASK	STOSM	SI		P	A	SP				ST	AD	10-44
SUBTRACT	SR	RR	C					IF		R	1B	7-40
SUBTRACT	S	RX	C		A			IF		R	5B	7-40
SUBTRACT DECIMAL	SP	SS	C		A		D	DF		ST	FB	8-11
SUBTRACT HALFWORD	SH	RX	C		A			IF		R	4B	7-40
SUBTRACT LOGICAL	SLR	RR	C							R	1F	7-40
SUBTRACT LOGICAL	SL	RX	C		A					R	5F	7-40
SUBTRACT NORMALIZED (extended)	SXR	RR	C			SP	EU	EO	LS		37	9-14
SUBTRACT NORMALIZED (long)	SDR	RR	C			SP	EU	EO	LS		2B	9-14
SUBTRACT NORMALIZED (long)	SD	RX	C		A	SP	EU	EO	LS		6B	9-14
SUBTRACT NORMALIZED (short)	SER	RR	C			SP	EU	EO	LS		3B	9-14
SUBTRACT NORMALIZED (short)	SE	RX	C		A	SP	EU	EO	LS		7B	9-14
SUBTRACT UNNORMALIZED (long)	SWR	RR	C			SP		EO	LS		2F	9-15
SUBTRACT UNNORMALIZED (long)	SW	RX	C		A	SP		EO	LS		6F	9-15
SUBTRACT UNNORMALIZED (short)	SUR	RR	C			SP		EO	LS		3F	9-15
SUBTRACT UNNORMALIZED (short)	SU	RX	C		A	SP		EO	LS		7F	9-15
SUPERVISOR CALL	SVC	RR							¢		0A	7-41
TEST AND SET	TS	S	C		A				$	ST	93	7-41
TEST BLOCK	TB	RRE	C	P	A[1]		II		$ GO	R	B22C	10-45
TEST PENDING INTERRUPTION	TPI	S	C	P	A[1]	SP			¢	ST	B236	14-15
TEST PROTECTION	TPROT	SSE	C	P	A[1]						E501	10-47
TEST SUBCHANNEL	TSCH	S	C	P	A	SP	OP		¢ GS	ST	B235	14-16
TEST UNDER MASK	TM	SI	C		A						91	7-42
TRACE	TRACE	RS		P	A	SP		T	¢		99	10-48
TRANSLATE	TR	SS			A					ST	DC	7-42
TRANSLATE AND TEST	TRT	SS	C		A				GM	R	DD	7-43
UNPACK	UNPK	SS			A					ST	F3	7-44
ZERO AND ADD	ZAP	SS	C		A		D	DF		ST	F8	8-11

Instructions Arranged by Name (Part 4 of 4)

LzLabs GmbH. Ex. 1016-509

Mnemonic	Name	Characteristics												Op Code	Page No.
	DIAGNOSE			DM	P	DM								83	10-4
A	ADD	RX	C			A			IF			R		5A	7-8
AD	ADD NORMALIZED (long)	RX	C			A	SP	EU	EO		LS			6A	9-6
ADR	ADD NORMALIZED (long)	RR	C				SP	EU	EO		LS			2A	9-6
AE	ADD NORMALIZED (short)	RX	C			A	SP	EU	EO		LS			7A	9-6
AER	ADD NORMALIZED (short)	RR	C				SP	EU	EO		LS			3A	9-6
AH	ADD HALFWORD	RX	C			A			IF			R		4A	7-8
AL	ADD LOGICAL	RX	C			A						R		5E	7-9
ALR	ADD LOGICAL	RR	C									R		1E	7-9
AP	ADD DECIMAL	SS	C			A		D	DF				ST	FA	8-5
AR	ADD	RR	C						IF			R		1A	7-8
AU	ADD UNNORMALIZED (short)	RX	C			A	SP		EO		LS			7E	9-7
AUR	ADD UNNORMALIZED (short)	RR	C				SP		EO		LS			3E	9-7
AW	ADD UNNORMALIZED (long)	RX	C			A	SP		EO		LS			6E	9-7
AWR	ADD UNNORMALIZED (long)	RR	C				SP		EO		LS			2E	9-7
AXR	ADD NORMALIZED (extended)	RR	C				SP	EU	EO		LS			36	9-6
BAL	BRANCH AND LINK	RX										B	R	45	7-10
BALR	BRANCH AND LINK	RR							T			B	R	05	7-10
BAS	BRANCH AND SAVE	RX										B	R	4D	7-10
BASR	BRANCH AND SAVE	RR							T			B	R	0D	7-10
BASSM	BRANCH AND SAVE AND SET MODE	RR							T			B	R	0C	7-11
BC	BRANCH ON CONDITION	RX										B		47	7-12
BCR	BRANCH ON CONDITION	RR								¢[1]		B		07	7-12
BCT	BRANCH ON COUNT	RX										B	R	46	7-13
BCTR	BRANCH ON COUNT	RR										B	R	06	7-13
BSM	BRANCH AND SET MODE	RR										B	R	0B	7-11
BXH	BRANCH ON INDEX HIGH	RS										B	R	86	7-13
BXLE	BRANCH ON INDEX LOW OR EQUAL	RS										B	R	87	7-13
C	COMPARE	RX	C			A								59	7-14
CD	COMPARE (long)	RX	C			A	SP							69	9-8
CDR	COMPARE (long)	RR	C				SP							29	9-8
CDS	COMPARE DOUBLE AND SWAP	RS	C			A	SP			$		R	ST	BB	7-14
CE	COMPARE (short)	RX	C			A	SP							79	9-8
CER	COMPARE (short)	RR	C				SP							39	9-8
CH	COMPARE HALFWORD	RX	C			A								49	7-16
CL	COMPARE LOGICAL	RX	C			A								55	7-16
CLC	COMPARE LOGICAL (character)	SS	C			A								D5	7-16
CLCL	COMPARE LOGICAL LONG	RR	C			A	SP	II				R		0F	7-17
CLI	COMPARE LOGICAL (immediate)	SI	C			A								95	7-16
CLM	COMPARE LOGICAL CHARACTERS UNDER MASK	RS	C			A								BD	7-17
CLR	COMPARE LOGICAL	RR	C											15	7-16
CP	COMPARE DECIMAL	SS	C			A		D						F9	8-5
CR	COMPARE	RR	C											19	7-14
CS	COMPARE AND SWAP	RS	C			A	SP			$		R	ST	BA	7-14
CSCH	CLEAR SUBCHANNEL	S	C		P			OP		¢	GS			B230	14-3
CVB	CONVERT TO BINARY	RX				A		D		IK		R		4F	7-19
CVD	CONVERT TO DECIMAL	RX				A							ST	4E	7-20
D	DIVIDE	RX				A	SP			IK		R		5D	7-20
DD	DIVIDE (long)	RX				A	SP	EU	EO	FK				6D	9-8
DDR	DIVIDE (long)	RR					SP	EU	EO	FK				2D	9-8
DE	DIVIDE (short)	RX				A	SP	EU	EO	FK				7D	9-8
DER	DIVIDE (short)	RR					SP	EU	EO	FK				3D	9-8
DP	DIVIDE DECIMAL	SS				A	SP	D		DK			ST	FD	8-5
DR	DIVIDE	RR					SP			IK		R		1D	7-20
DXR	DIVIDE (extended)	RRE					SP	EU	EO	FK				B22D	9-8

Instructions Arranged by Mnemonic (Part 1 of 4)

LzLabs GmbH. Ex. 1016-510

Mnemonic	Name	Characteristics								Op Code	Page No.
ED	EDIT	SS	C		A		D		ST	DE	8-6
EDMK	EDIT AND MARK	SS	C		A		D	G1	R ST	DF	8-9
EPAR	EXTRACT PRIMARY ASN	RRE		Q			SO		R	B226	10-5
ESAR	EXTRACT SECONDARY ASN	RRE		Q			SO		R	B227	10-5
EX	EXECUTE	RX			AI	SP		EX		44	7-22
HDR	HALVE (long)	RR				SP	EU			24	9-10
HER	HALVE (short)	RR				SP	EU			34	9-10
HSCH	HALT SUBCHANNEL	S	C	P			OP	♦ GS		B231	14-4
IAC	INSERT ADDRESS SPACE CONTROL	RRE	C	Q			SO		R	B224	10-6
IC	INSERT CHARACTER	RX			A				R	43	7-23
ICM	INSERT CHARACTERS UNDER MASK	RS	C		A				R	BF	7-23
IPK	INSERT PSW KEY	S		Q				G2	R	B20B	10-7
IPM	INSERT PROGRAM MASK	RRE							R	B222	7-23
IPTE	INVALIDATE PAGE TABLE ENTRY	RRE		P	A[1]			$		B221	10-8
ISKE	INSERT STORAGE KEY EXTENDED	RRE		P						B229	10-7
IVSK	INSERT VIRTUAL STORAGE KEY	RRE		Q	A[1]		SO		R	B223	10-7
L	LOAD	RX			A				R	58	7-24
LA	LOAD ADDRESS	RX							R	41	7-24
LASP	LOAD ADDRESS SPACE PARAMETERS	SSE	C	P	AS	SP	SO			E500	10-10
LCDR	LOAD COMPLEMENT (long)	RR	C			SP				23	9-11
LCER	LOAD COMPLEMENT (short)	RR	C			SP				33	9-11
LCR	LOAD COMPLEMENT	RR	C				IF		R	13	7-24
LCTL	LOAD CONTROL	RS		P	A	SP				B7	10-17
LD	LOAD (long)	RX			A	SP				68	9-10
LDR	LOAD (long)	RR				SP				28	9-10
LE	LOAD (short)	RX			A	SP				78	9-10
LER	LOAD (short)	RR				SP				38	9-10
LH	LOAD HALFWORD	RX			A				R	48	7-25
LM	LOAD MULTIPLE	RS			A				R	98	7-25
LNDR	LOAD NEGATIVE (long)	RR	C			SP				21	9-11
LNER	LOAD NEGATIVE (short)	RR	C			SP				31	9-11
LNR	LOAD NEGATIVE	RR	C						R	11	7-25
LPDR	LOAD POSITIVE (long)	RR	C			SP				20	9-12
LPER	LOAD POSITIVE (short)	RR	C			SP				30	9-12
LPR	LOAD POSITIVE	RR	C				IF		R	10	7-26
LPSW	LOAD PSW	S	L	P	A	SP		♦		82	10-17
LR	LOAD	RR							R	18	7-24
LRA	LOAD REAL ADDRESS	RX	C	P	A[1]				R	B1	10-18
LRDR	LOAD ROUNDED (extended to long)	RR				SP	EO			25	9-12
LRER	LOAD ROUNDED (long to short)	RR				SP	EO			35	9-12
LTDR	LOAD AND TEST (long)	RR	C			SP				22	9-11
LTER	LOAD AND TEST (short)	RR	C			SP				32	9-11
LTR	LOAD AND TEST	RR	C						R	12	7-24
M	MULTIPLY	RX			A	SP			R	5C	7-32
MC	MONITOR CALL	SI				SP		MO		AF	7-26
MD	MULTIPLY (long)	RX			A	SP	EU EO			6C	9-13
MDR	MULTIPLY (long)	RR				SP	EU EO			2C	9-13
ME	MULTIPLY (short to long)	RX			A	SP	EU EO			7C	9-13
MER	MULTIPLY (short to long)	RR				SP	EU EO			3C	9-13
MH	MULTIPLY HALFWORD	RX			A				R	4C	7-32
MP	MULTIPLY DECIMAL	SS			A	SP	D		ST	FC	8-10
MR	MULTIPLY	RR				SP			R	1C	7-32
MSCH	MODIFY SUBCHANNEL	S	C	P	A	SP	OP	♦ GS		B232	14-6
MVC	MOVE (character)	SS			A				ST	D2	7-27
MVCK	MOVE WITH KEY	SS	C	Q	A				ST	D9	10-20

Instructions Arranged by Mnemonic (Part 2 of 4)

LzLabs GmbH. Ex. 1016-511

Mnemonic	Name	Characteristics				Op Code	Page No.
MVCL	MOVE LONG	RR C	A SP	II	R ST	0E	7-27
MVCP	MOVE TO PRIMARY	SS C	Q A[1]	SO ϕ	ST	DA	10-19
MVCS	MOVE TO SECONDARY	SS C	Q A[1]	SO ϕ	ST	DB	10-19
MVI	MOVE (immediate)	SI	A		ST	92	7-27
MVN	MOVE NUMERICS	SS	A		ST	D1	7-30
MVO	MOVE WITH OFFSET	SS	A		ST	F1	7-31
MVZ	MOVE ZONES	SS	A		ST	D3	7-31
MXD	MULTIPLY (long to extended)	RX	A SP	EU EO		67	9-13
MXDR	MULTIPLY (long to extended)	RR	SP	EU EO		27	9-13
MXR	MULTIPLY (extended)	RR	SP	EU EO		26	9-13
N	AND	RX C	A		R	54	7-9
NC	AND (character)	SS C	A		ST	D4	7-9
NI	AND (immediate)	SI C	A		ST	94	7-9
NR	AND	RR C			R	14	7-9
O	OR	RX C	A		R	56	7-33
OC	OR (character)	SS C	A		ST	D6	7-33
OI	OR (immediate)	SI C	A		ST	96	7-33
OR	OR	RR C			R	16	7-33
PACK	PACK	SS	A		ST	F2	7-33
PC	PROGRAM CALL	S	Q AT	Z[1] T ϕ GM	B R	B218	10-22
PT	PROGRAM TRANSFER	RRE	Q AT SP	Z[2] T ϕ	B	B228	10-28
PTLB	PURGE TLB	S	P	$		B20D	10-33
RCHP	RESET CHANNEL PATH	S C	P	OP ϕ G1		B23B	14-6
RRBE	RESET REFERENCE BIT EXTENDED	RRE C	P			B22A	10-33
RSCH	RESUME SUBCHANNEL	S C	P	OP ϕ GS		B238	14-8
S	SUBTRACT	RX C	A	IF	R	5B	7-40
SAC	SET ADDRESS SPACE CONTROL	S	SP	SO ϕ		B219	10-33
SAL	SET ADDRESS LIMIT	S	P	OP ϕ G1		B237	14-9
SCHM	SET CHANNEL MONITOR	S	P	OP ϕ GM		B23C	14-10
SCK	SET CLOCK	S C	P A SP			B204	10-34
SCKC	SET CLOCK COMPARATOR	S	P A SP			B206	10-35
SD	SUBTRACT NORMALIZED (long)	RX C	A SP	EU EO LS		6B	9-14
SDR	SUBTRACT NORMALIZED (long)	RR C	SP	EU EO LS		2B	9-14
SE	SUBTRACT NORMALIZED (short)	RX C	A SP	EU EO LS		7B	9-14
SER	SUBTRACT NORMALIZED (short)	RR C	SP	EU EO LS		3B	9-14
SH	SUBTRACT HALFWORD	RX C	A	IF	R	4B	7-40
SIGP	SIGNAL PROCESSOR	RS C	P	$	R	AE	10-41
SL	SUBTRACT LOGICAL	RX C	A		R	5F	7-40
SLA	SHIFT LEFT SINGLE	RS C		IF	R	8B	7-35
SLDA	SHIFT LEFT DOUBLE	RS C	SP	IF	R	8F	7-34
SLDL	SHIFT LEFT DOUBLE LOGICAL	RS	SP		R	8D	7-35
SLL	SHIFT LEFT SINGLE LOGICAL	RS			R	89	7-36
SLR	SUBTRACT LOGICAL	RR C			R	1F	7-40
SP	SUBTRACT DECIMAL	SS C	A	D DF	ST	FB	8-11
SPKA	SET PSW KEY FROM ADDRESS	S	Q			B20A	10-36
SPM	SET PROGRAM MASK	RR L				04	7-34
SPT	SET CPU TIMER	S	P A SP			B208	10-35
SPX	SET PREFIX	S	P A SP	$		B210	10-36
SR	SUBTRACT	RR C		IF	R	1B	7-40
SRA	SHIFT RIGHT SINGLE	RS C			R	8A	7-37
SRDA	SHIFT RIGHT DOUBLE	RS C	SP		R	8E	7-36
SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS	SP		R	8C	7-36
SRL	SHIFT RIGHT SINGLE LOGICAL	RS			R	88	7-37
SRP	SHIFT AND ROUND DECIMAL	SS C	A	D DF	ST	F0	8-10
SSAR	SET SECONDARY ASN	RRE	AT	Z[3] T ϕ		B225	10-37

Instructions Arranged by Mnemonic (Part 3 of 4)

LzLabs GmbH. Ex. 1016-512

Mnemonic	Name	Characteristics								Op Code	Page No.
SSCH	START SUBCHANNEL	S	C	P	A	SP	OP	◊	GS	B233	14-11
SSKE	SET STORAGE KEY EXTENDED	RRE		P				◊		B22B	10-40
SSM	SET SYSTEM MASK	S		P	A	SP	SO			80	10-40
ST	STORE	RX			A				ST	50	7-37
STAP	STORE CPU ADDRESS	S		P	A	SP			ST	B212	10-43
STC	STORE CHARACTER	RX			A				ST	42	7-38
STCK	STORE CLOCK	S	C		A			$	ST	B205	7-38
STCKC	STORE CLOCK COMPARATOR	S		P	A	SP			ST	B207	10-42
STCM	STORE CHARACTERS UNDER MASK	RS			A				ST	BE	7-38
STCPS	STORE CHANNEL PATH STATUS	S		P	A	SP		◊	ST	B23A	14-13
STCRW	STORE CHANNEL REPORT WORD	S	C	P	A	SP		◊	ST	B239	14-13
STCTL	STORE CONTROL	RS		P	A	SP			ST	B6	10-42
STD	STORE (long)	RX			A	SP			ST	60	9-14
STE	STORE (short)	RX			A	SP			ST	70	9-14
STH	STORE HALFWORD	RX			A				ST	40	7-39
STIDP	STORE CPU ID	S		P	A	SP			ST	B202	10-43
STM	STORE MULTIPLE	RS			A				ST	90	7-39
STNSM	STORE THEN AND SYSTEM MASK	SI		P	A				ST	AC	10-44
STOSM	STORE THEN OR SYSTEM MASK	SI		P	A	SP			ST	AD	10-44
STPT	STORE CPU TIMER	S		P	A	SP			ST	B209	10-43
STPX	STORE PREFIX	S		P	A	SP			ST	B211	10-44
STSCH	STORE SUBCHANNEL	S	C	P	A	SP	OP	◊	GS ST	B234	14-14
SU	SUBTRACT UNNORMALIZED (short)	RX	C		A	SP	EO		LS	7F	9-15
SUR	SUBTRACT UNNORMALIZED (short)	RR	C			SP	EO		LS	3F	9-15
SVC	SUPERVISOR CALL	RR						◊		0A	7-41
SW	SUBTRACT UNNORMALIZED (long)	RX	C		A	SP	EO		LS	6F	9-15
SWR	SUBTRACT UNNORMALIZED (long)	RR	C			SP	EO		LS	2F	9-15
SXR	SUBTRACT NORMALIZED (extended)	RR	C			SP	EU EO		LS	37	9-14
TB	TEST BLOCK	RRE	C	P	A[1]		II	$	G0 R	B22C	10-45
TM	TEST UNDER MASK	SI	C		A					91	7-42
TPI	TEST PENDING INTERRUPTION	S	C	P	A[1]	SP		◊	ST	B236	14-15
TPROT	TEST PROTECTION	SSE	C	P	A[1]					E501	10-47
TR	TRANSLATE	SS			A				ST	DC	7-42
TRACE	TRACE	RS		P	A	SP	T	◊		99	10-48
TRT	TRANSLATE AND TEST	SS	C		A				GM R	DD	7-43
TS	TEST AND SET	S	C		A			$	ST	93	7-41
TSCH	TEST SUBCHANNEL	S	C	P	A	SP	OP	◊	GS ST	B235	14-16
UNPK	UNPACK	SS			A				ST	F3	7-44
X	EXCLUSIVE OR	RX	C		A				R	57	7-21
XC	EXCLUSIVE OR (character)	SS	C		A				ST	D7	7-21
XI	EXCLUSIVE OR (immediate)	SI	C		A				ST	97	7-21
XR	EXCLUSIVE OR	RR	C						R	17	7-21
ZAP	ZERO AND ADD	SS	C		A		D	DF	ST	F8	8-11

Instructions Arranged by Mnemonic (Part 4 of 4)

LzLabs GmbH. Ex. 1016-513

Op Code	Name	Mnemonic	Characteristics				Page No.
04	SET PROGRAM MASK	SPM	RR L				7-34
05	BRANCH AND LINK	BALR	RR		T	B R	7-10
06	BRANCH ON COUNT	BCTR	RR			B R	7-13
07	BRANCH ON CONDITION	BCR	RR		¢[1]	B	7-12
0A	SUPERVISOR CALL	SVC	RR		¢		7-41
0B	BRANCH AND SET MODE	BSM	RR			B R	7-11
0C	BRANCH AND SAVE AND SET MODE	BASSM	RR		T	B R	7-11
0D	BRANCH AND SAVE	BASR	RR		T	B R	7-10
0E	MOVE LONG	MVCL	RR C	A SP	II	R ST	7-27
0F	COMPARE LOGICAL LONG	CLCL	RR C	A SP	II	R	7-17
10	LOAD POSITIVE	LPR	RR C		IF	R	7-26
11	LOAD NEGATIVE	LNR	RR C			R	7-25
12	LOAD AND TEST	LTR	RR C			R	7-24
13	LOAD COMPLEMENT	LCR	RR C		IF	R	7-24
14	AND	NR	RR C			R	7-9
15	COMPARE LOGICAL	CLR	RR C				7-16
16	OR	OR	RR C			R	7-33
17	EXCLUSIVE OR	XR	RR C			R	7-21
18	LOAD	LR	RR			R	7-24
19	COMPARE	CR	RR C				7-14
1A	ADD	AR	RR C		IF	R	7-8
1B	SUBTRACT	SR	RR C		IF	R	7-40
1C	MULTIPLY	MR	RR	SP		R	7-32
1D	DIVIDE	DR	RR	SP	IK	R	7-20
1E	ADD LOGICAL	ALR	RR C			R	7-9
1F	SUBTRACT LOGICAL	SLR	RR C			R	7-40
20	LOAD POSITIVE (long)	LPDR	RR C	SP			9-12
21	LOAD NEGATIVE (long)	LNDR	RR C	SP			9-11
22	LOAD AND TEST (long)	LTDR	RR C	SP			9-11
23	LOAD COMPLEMENT (long)	LCDR	RR C	SP			9-11
24	HALVE (long)	HDR	RR	SP	EU		9-10
25	LOAD ROUNDED (extended to long)	LRDR	RR	SP	EO		9-12
26	MULTIPLY (extended)	MXR	RR	SP	EU EO		9-13
27	MULTIPLY (long to extended)	MXDR	RR	SP	EU EO		9-13
28	LOAD (long)	LDR	RR	SP			9-10
29	COMPARE (long)	CDR	RR C	SP			9-8
2A	ADD NORMALIZED (long)	ADR	RR C	SP	EU EO LS		9-6
2B	SUBTRACT NORMALIZED (long)	SDR	RR C	SP	EU EO LS		9-14
2C	MULTIPLY (long)	MDR	RR	SP	EU EO		9-13
2D	DIVIDE (long)	DDR	RR	SP	EU EO FK		9-8
2E	ADD UNNORMALIZED (long)	AWR	RR C	SP	EO LS		9-7
2F	SUBTRACT UNNORMALIZED (long)	SWR	RR C	SP	EO LS		9-15
30	LOAD POSITIVE (short)	LPER	RR C	SP			9-12
31	LOAD NEGATIVE (short)	LNER	RR C	SP			9-11
32	LOAD AND TEST (short)	LTER	RR C	SP			9-11
33	LOAD COMPLEMENT (short)	LCER	RR C	SP			9-11
34	HALVE (short)	HER	RR	SP	EU		9-10
35	LOAD ROUNDED (long to short)	LRER	RR	SP	EO		9-12
36	ADD NORMALIZED (extended)	AXR	RR C	SP	EU EO LS		9-6
37	SUBTRACT NORMALIZED (extended)	SXR	RR C	SP	EU EO LS		9-14
38	LOAD (short)	LER	RR	SP			9-10
39	COMPARE (short)	CER	RR C	SP			9-8
3A	ADD NORMALIZED (short)	AER	RR C	SP	EU EO LS		9-6
3B	SUBTRACT NORMALIZED (short)	SER	RR C	SP	EU EO LS		9-14
3C	MULTIPLY (short to long)	MER	RR	SP	EU EO		9-13

Instructions Arranged by Operation Code (Part 1 of 4)

LzLabs GmbH. Ex. 1016-514

Op Code	Name	Mne-monic	Characteristics				Page No.
3D	DIVIDE (short)	DER	RR	SP	EU EO FK		9-8
3E	ADD UNNORMALIZED (short)	AUR	RR C	SP	EO LS		9-7
3F	SUBTRACT UNNORMALIZED (short)	SUR	RR C	SP	EO LS		9-15
40	STORE HALFWORD	STH	RX	A		ST	7-39
41	LOAD ADDRESS	LA	RX			R	7-24
42	STORE CHARACTER	STC	RX	A		ST	7-38
43	INSERT CHARACTER	IC	RX	A		R	7-23
44	EXECUTE	EX	RX	AI SP	EX		7-22
45	BRANCH AND LINK	BAL	RX			B R	7-10
46	BRANCH ON COUNT	BCT	RX			B R	7-13
47	BRANCH ON CONDITION	BC	RX			B	7-12
48	LOAD HALFWORD	LH	RX	A		R	7-25
49	COMPARE HALFWORD	CH	RX C	A			7-16
4A	ADD HALFWORD	AH	RX C	A	IF	R	7-8
4B	SUBTRACT HALFWORD	SH	RX C	A	IF	R	7-40
4C	MULTIPLY HALFWORD	MH	RX	A		R	7-32
4D	BRANCH AND SAVE	BAS	RX			B R	7-10
4E	CONVERT TO DECIMAL	CVD	RX	A		ST	7-20
4F	CONVERT TO BINARY	CVB	RX	A	D IK	R	7-19
50	STORE	ST	RX	A		ST	7-37
54	AND	N	RX C	A		R	7-9
55	COMPARE LOGICAL	CL	RX C	A			7-16
56	OR	O	RX C	A		R	7-33
57	EXCLUSIVE OR	X	RX C	A		R	7-21
58	LOAD	L	RX	A		R	7-24
59	COMPARE	C	RX C	A			7-14
5A	ADD	A	RX C	A	IF	R	7-8
5B	SUBTRACT	S	RX C	A	IF	R	7-40
5C	MULTIPLY	M	RX	A SP		R	7-32
5D	DIVIDE	D	RX	A SP	IK	R	7-20
5E	ADD LOGICAL	AL	RX C	A		R	7-9
5F	SUBTRACT LOGICAL	SL	RX C	A		R	7-40
60	STORE (long)	STD	RX	A SP		ST	9-14
67	MULTIPLY (long to extended)	MXD	RX	A SP	EU EO		9-13
68	LOAD (long)	LD	RX	A SP			9-10
69	COMPARE (long)	CD	RX C	A SP			9-8
6A	ADD NORMALIZED (long)	AD	RX C	A SP	EU EO LS		9-6
6B	SUBTRACT NORMALIZED (long)	SD	RX C	A SP	EU EO LS		9-14
6C	MULTIPLY (long)	MD	RX	A SP	EU EO		9-13
6D	DIVIDE (long)	DD	RX	A SP	EU EO FK		9-8
6E	ADD UNNORMALIZED (long)	AW	RX C	A SP	EO LS		9-7
6F	SUBTRACT UNNORMALIZED (long)	SW	RX C	A SP	EO LS		9-15
70	STORE (short)	STE	RX	A SP		ST	9-14
78	LOAD (short)	LE	RX	A SP			9-10
79	COMPARE (short)	CE	RX C	A SP			9-8
7A	ADD NORMALIZED (short)	AE	RX C	A SP	EU EO LS		9-6
7B	SUBTRACT NORMALIZED (short)	SE	RX C	A SP	EU EO LS		9-14
7C	MULTIPLY (short to long)	ME	RX	A SP	EU EO		9-13
7D	DIVIDE (short)	DE	RX	A SP	EU EO FK		9-8
7E	ADD UNNORMALIZED (short)	AU	RX C	A SP	EO LS		9-7
7F	SUBTRACT UNNORMALIZED (short)	SU	RX C	A SP	EO LS		9-15
80	SET SYSTEM MASK	SSM	S	P A SP	SO		10-40
82	LOAD PSW	LPSW	S L	P A SP	ϕ		10-17
83	DIAGNOSE		DM	P DM			10-4
86	BRANCH ON INDEX HIGH	BXH	RS			B R	7-13

Instructions Arranged by Operation Code (Part 2 of 4)

LzLabs GmbH. Ex. 1016-515

Op Code	Name	Mnemonic	Characteristics												Page No.
87	BRANCH ON INDEX LOW OR EQUAL	BXLE	RS									B	R		7-13
88	SHIFT RIGHT SINGLE LOGICAL	SRL	RS										R		7-37
89	SHIFT LEFT SINGLE LOGICAL	SLL	RS										R		7-36
8A	SHIFT RIGHT SINGLE	SRA	RS	C									R		7-37
8B	SHIFT LEFT SINGLE	SLA	RS	C						IF			R		7-35
8C	SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS				SP						R		7-36
8D	SHIFT LEFT DOUBLE LOGICAL	SLDL	RS				SP						R		7-35
8E	SHIFT RIGHT DOUBLE	SRDA	RS	C			SP						R		7-36
8F	SHIFT LEFT DOUBLE	SLDA	RS	C			SP			IF			R		7-34
90	STORE MULTIPLE	STM	RS			A								ST	7-39
91	TEST UNDER MASK	TM	SI	C		A									7-42
92	MOVE (immediate)	MVI	SI			A								ST	7-27
93	TEST AND SET	TS	S	C		A					$			ST	7-41
94	AND (immediate)	NI	SI	C		A								ST	7-9
95	COMPARE LOGICAL (immediate)	CLI	SI	C		A									7-16
96	OR (immediate)	OI	SI	C		A								ST	7-33
97	EXCLUSIVE OR (immediate)	XI	SI	C		A								ST	7-21
98	LOAD MULTIPLE	LM	RS			A							R		7-25
99	TRACE	TRACE	RS		P	A	SP			T	¢				10-48
AC	STORE THEN AND SYSTEM MASK	STNSM	SI		P	A								ST	10-44
AD	STORE THEN OR SYSTEM MASK	STOSM	SI		P	A	SP							ST	10-44
AE	SIGNAL PROCESSOR	SIGP	RS	C	P						$		R		10-41
AF	MONITOR CALL	MC	SI				SP					MO			7-26
B1	LOAD REAL ADDRESS	LRA	RX	C	P	A[1]							R		10-18
B202	STORE CPU ID	STIDP	S		P	A	SP							ST	10-43
B204	SET CLOCK	SCK	S	C	P	A	SP								10-34
B205	STORE CLOCK	STCK	S	C		A					$			ST	7-38
B206	SET CLOCK COMPARATOR	SCKC	S		P	A	SP								10-35
B207	STORE CLOCK COMPARATOR	STCKC	S		P	A	SP							ST	10-42
B208	SET CPU TIMER	SPT	S		P	A	SP								10-35
B209	STORE CPU TIMER	STPT	S		P	A	SP							ST	10-43
B20A	SET PSW KEY FROM ADDRESS	SPKA	S		Q										10-36
B20B	INSERT PSW KEY	IPK	S		Q							G2	R		10-7
B20D	PURGE TLB	PTLB	S		P						$				10-33
B210	SET PREFIX	SPX	S		P	A	SP				$				10-36
B211	STORE PREFIX	STPX	S		P	A	SP							ST	10-44
B212	STORE CPU ADDRESS	STAP	S		P	A	SP							ST	10-43
B218	PROGRAM CALL	PC	S		Q	AT				Z[1]	T	¢ GM	B	R	10-22
B219	SET ADDRESS SPACE CONTROL	SAC	S				SP			SO		¢			10-33
B221	INVALIDATE PAGE TABLE ENTRY	IPTE	RRE		P	A[1]						$			10-8
B222	INSERT PROGRAM MASK	IPM	RRE										R		7-23
B223	INSERT VIRTUAL STORAGE KEY	IVSK	RRE		Q	A[1]				SO			R		10-7
B224	INSERT ADDRESS SPACE CONTROL	IAC	RRE	C	Q					SO			R		10-6
B225	SET SECONDARY ASN	SSAR	RRE			AT				Z[3]	T	¢			10-37
B226	EXTRACT PRIMARY ASN	EPAR	RRE		Q					SO			R		10-5
B227	EXTRACT SECONDARY ASN	ESAR	RRE		Q					SO			R		10-5
B228	PROGRAM TRANSFER	PT	RRE		Q	AT	SP			Z[2]	T	¢	B		10-28
B229	INSERT STORAGE KEY EXTENDED	ISKE	RRE		P										10-7
B22A	RESET REFERENCE BIT EXTENDED	RRBE	RRE	C	P										10-33
B22B	SET STORAGE KEY EXTENDED	SSKE	RRE		P							¢			10-40
B22C	TEST BLOCK	TB	RRE	C	P	A[1]				II		$ GO	R		10-45
B22D	DIVIDE (extended)	DXR	RRE				SP			EU	EO	FK			9-8
B230	CLEAR SUBCHANNEL	CSCH	S	C	P					OP		¢ GS			14-3
B231	HALT SUBCHANNEL	HSCH	S	C	P					OP		¢ GS			14-4
B232	MODIFY SUBCHANNEL	MSCH	S	C	P	A	SP			OP		¢ GS			14-6

Instructions Arranged by Operation Code (Part 3 of 4)

LzLabs GmbH. Ex. 1016-516

Op Code	Name	Mnemonic	Characteristics				Page No.
B233	START SUBCHANNEL	SSCH	S C	P A SP	OP ◊ GS		14-11
B234	STORE SUBCHANNEL	STSCH	S C	P A SP	OP ◊ GS	ST	14-14
B235	TEST SUBCHANNEL	TSCH	S C	P A SP	OP ◊ GS	ST	14-16
B236	TEST PENDING INTERRUPTION	TPI	S C	P A[1] SP	◊	ST	14-15
B237	SET ADDRESS LIMIT	SAL	S	P	OP ◊ G1		14-9
B238	RESUME SUBCHANNEL	RSCH	S C	P	OP ◊ GS		14-8
B239	STORE CHANNEL REPORT WORD	STCRW	S C	P A SP	◊	ST	14-13
B23A	STORE CHANNEL PATH STATUS	STCPS	S	P A SP	◊	ST	14-13
B23B	RESET CHANNEL PATH	RCHP	S C	P	OP ◊ G1		14-6
B23C	SET CHANNEL MONITOR	SCHM	S	P	OP ◊ GM		14-10
B6	STORE CONTROL	STCTL	RS	P A SP		ST	10-42
B7	LOAD CONTROL	LCTL	RS	P A SP			10-17
BA	COMPARE AND SWAP	CS	RS C	A SP	$	R ST	7-14
BB	COMPARE DOUBLE AND SWAP	CDS	RS C	A SP	$	R ST	7-14
BD	COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS C	A			7-17
BE	STORE CHARACTERS UNDER MASK	STCM	RS	A		ST	7-38
BF	INSERT CHARACTERS UNDER MASK	ICM	RS C	A		R	7-23
D1	MOVE NUMERICS	MVN	SS	A		ST	7-30
D2	MOVE (character)	MVC	SS	A		ST	7-27
D3	MOVE ZONES	MVZ	SS	A		ST	7-31
D4	AND (character)	NC	SS C	A		ST	7-9
D5	COMPARE LOGICAL (character)	CLC	SS C	A			7-16
D6	OR (character)	OC	SS C	A		ST	7-33
D7	EXCLUSIVE OR (character)	XC	SS C	A		ST	7-21
D9	MOVE WITH KEY	MVCK	SS C	Q A		ST	10-20
DA	MOVE TO PRIMARY	MVCP	SS C	Q A[1]	SO ◊	ST	10-19
DB	MOVE TO SECONDARY	MVCS	SS C	Q A[1]	SO ◊	ST	10-19
DC	TRANSLATE	TR	SS	A		ST	7-42
DD	TRANSLATE AND TEST	TRT	SS C	A	GM	R	7-43
DE	EDIT	ED	SS C	A	D	ST	8-6
DF	EDIT AND MARK	EDMK	SS C	A	D G1	R ST	8-9
E500	LOAD ADDRESS SPACE PARAMETERS	LASP	SSE C	P AS SP	SO		10-10
E501	TEST PROTECTION	TPROT	SSE C	P A[1]			10-47
F0	SHIFT AND ROUND DECIMAL	SRP	SS C	A	D DF	ST	8-10
F1	MOVE WITH OFFSET	MVO	SS	A		ST	7-31
F2	PACK	PACK	SS	A		ST	7-33
F3	UNPACK	UNPK	SS	A		ST	7-44
F8	ZERO AND ADD	ZAP	SS C	A	D DF	ST	8-11
F9	COMPARE DECIMAL	CP	SS C	A	D		8-5
FA	ADD DECIMAL	AP	SS C	A	D DF	ST	8-5
FB	SUBTRACT DECIMAL	SP	SS C	A	D DF	ST	8-11
FC	MULTIPLY DECIMAL	MP	SS	A SP	D	ST	8-10
FD	DIVIDE DECIMAL	DP	SS	A SP	D DK	ST	8-5

Instructions Arranged by Operation Code (Part 4 of 4)

LzLabs GmbH. Ex. 1016-517

APPENDIX C. CONDITION-CODE SETTINGS

This appendix lists the condition-code setting for all instructions in the 370-XA architecture which set the condition code. In addition to those instructions listed which set the condition code, the condition code may be changed by DIAGNOSE and the target of EXECUTE. The condition code is loaded by LOAD PSW, by 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.

Instruction	Condition Code			
	0	1	2	3
ADD, ADD HALFWORD	zero	< zero	> zero	overflow
ADD DECIMAL	zero	< zero	> zero	overflow
ADD LOGICAL	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	--	--
CLEAR SUBCHANNEL	function accepted	--	--	unsuccessful
COMPARE (gen, fl pt)	equal	low	high	--
COMPARE HALFWORD	equal	low	high	--
COMPARE AND SWAP	equal	not equal	--	--
COMPARE DECIMAL	equal	low	high	--
COMPARE DOUBLE AND SWAP	equal	not equal	--	--
COMPARE LOGICAL	equal	low	high	--
COMPARE LOGICAL CHARACTERS UNDER MASK	equal	low	high	--
COMPARE LOGICAL LONG	equal	low	high	--
EDIT, EDIT AND MARK	zero	< zero	> zero	--
EXCLUSIVE OR	zero	not zero	--	--
HALT SUBCHANNEL	function accepted	status pending, function not executed	busy	unsuccessful
INSERT ADDRESS SPACE CONTROL	zero	one	--	--
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, fl pt)	zero	< zero	> zero	--
LOAD COMPLEMENT (gen)	zero	< zero	> zero	overflow
LOAD COMPLEMENT (fl pt)	zero	< zero	> zero	--
LOAD NEGATIVE (gen, fl pt)	zero	< zero	--	--
LOAD POSITIVE (gen)	zero	--	> zero	overflow
LOAD POSITIVE (fl pt)	zero	--	> zero	--
LOAD REAL ADDRESS	translation available	ST entry invalid	PT entry invalid	length violation
MODIFY SUBCHANNEL	function executed	status pending, function not executed	busy	unsuccessful
MOVE LONG	length equal	length low	length high	destr overlap

Summary of Condition-Code Settings (Part 1 of 2)

LzLabs GmbH. Ex. 1016-519

Instruction	Condition Code			
	0	1	2	3
MOVE TO PRIMARY, MOVE TO SECONDARY	length =< 256	--	--	length > 256
MOVE WITH KEY	length =< 256	--	--	length > 256
OR	zero	not zero	--	--
RESET CHANNEL PATH	function accepted	--	busy	unsuccessful
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	resume initiated	status pending	resume not applicable	not operational
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	ORB accepted	status pending, ORB not accepted	busy	unsuccessful
STORE CHANNEL REPORT WORD	CRW stored	CRW not stored	--	--
STORE CLOCK	set	not set	error	not operational
STORE SUBCHANNEL	SCHIB stored	--	--	unsuccessful
SUBTRACT, SUBTRACT HALFWORD	zero	< zero	> zero	overflow
SUBTRACT DECIMAL	zero	< zero	> zero	overflow
SUBTRACT LOGICAL	--	not zero, no carry	zero, carry	not zero, carry
SUBTRACT NORMALIZED	zero	< zero	> zero	--
SUBTRACT UNNORMALIZED	zero	< zero	> zero	--
TEST AND SET	left bit zero	left bit one	--	--
TEST BLOCK	usable	not usable	--	--
TEST PENDING INTERRUPTION	int code not stored	int code stored	--	--
TEST PROTECTION	can fetch, can store	can fetch, cannot store	cannot fetch, cannot store	translation not available
TEST SUBCHANNEL	status pending, function executed	not status pending, function executed	--	unsuccessful
TEST UNDER MASK	all zeros	mixed	--	all ones
TRANSLATE AND TEST	all zeros	incomplete	complete	--
ZERO AND ADD	zero	< zero	> zero	overflow

Explanation:

> zero	Result is greater than zero
< zero	Result is less than zero
=< 256	Equal to, or less than, 256
> 256	Greater than 256
high	First operand compares high
low	First operand compares low
length	Length of first operand

Summary of Condition-Code Settings (Part 2 of 2)

LzLabs GmbH. Ex. 1016-520

APPENDIX D. COMPARISON BETWEEN SYSTEM/370 AND 370-XA MODES

New Facilities in 370-XA Mode D-1
  Bimodal Addressing D-1
  31-Bit Logical Addressing D-1
  31-Bit Real and Absolute Addressing D-1
  Page Protection D-2
  Tracing D-2
Comparison of Facilities D-2
Summary of Changes D-4
  Changes in Instructions Provided D-4
  Input/Output Comparison D-6
  Comparison of PSW Formats D-7
  Changes in Control-Register Assignment D-7
  Changes in Assigned Storage Locations D-9
  SIGNAL PROCESSOR Changes D-9
  Machine-Check Changes D-9
  Changes to Addressing Wraparound D-10
  Changes to LOAD REAL ADDRESS D-10
  Changes to 31-Bit Real Operand Addresses D-10

This appendix provides (1) a list of the new facilities in the 370-XA mode that are not provided in the System/370 mode, (2) a description of the handling in the 370-XA mode of the facilities available in the System/370 mode, and (3) a list of changes between the System/370 mode and the 370-XA mode.

NEW FACILITIES IN 370-XA MODE

The following facilities are new with the 370-XA mode and are not provided in the System/370 mode.

BIMODAL ADDRESSING

Two modes of operation are provided: a 24-bit addressing mode, for running 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 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

LzLabs GmbH. Ex. 1016-521

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

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-mode segment protection, which is not offered in the 370-XA mode.

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.

COMPARISON OF FACILITIES

The figure "Availability of System/370 Facilities in 370-XA Mode" shows the facilities offered in System/370 and whether or not the facility is provided in the 370-XA mode.

LzLabs GmbH. Ex. 1016-522

System/370 Facility	Availability in 370-XA Mode
Commercial instruction set	P[1]
Block-multiplexer channels	F
Branch and save	S
Byte-multiplexer channels	F
Channel indirect data addressing	S
Channel-set switching	F
Clear I/O	F
Command retry	S
Conditional swapping	S
CPU timer and clock comparator	S
Direct control	-
Dual address space	P[2]
Extended	P[3]
Extended-precision floating point	S
Extended real addressing	R[4]
External signals	-
Fast release	F
Floating point	S
Halt device	F
I/O extended logout	-
Limited channel logout	F
Move inverse	-
Multiprocessing	S[5]
PSW-key handling	S
Recovery extensions	-
Segment protection	R[6]
Selector channels	F
Service signal	S
Start-I/O-fast queuing	F
Storage-key-instruction extensions	S
Storage-key 4K-byte block	P[7]
Suspend and resume	F
Test block	S
Translation	P[8]
31-bit IDAWs	S

Availability of System/370 Facilities in 370-XA Mode (Part 1 of 2)

Explanation:

F Facility is not provided, but a comparable function is provided by the channel subsystem.
P Facility is partially available in the 370-XA mode.
R Facility is replaced with a comparable facility.
S Facility is standard in the 370-XA mode.
- Facility is not provided in the 370-XA mode.

[1] The following items, which are part of the basic computing function in the System/370 mode, are not provided in the 370-XA mode: BC-mode, interval timer, and 2K-byte protection blocks. Also see the following instruction lists for those instructions standard in the System/370 mode which are not provided in the 370-XA mode.

[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 the 370-XA mode.

[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 standard.

[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.

Availability of System/370 Facilities in 370-XA Mode (Part 2 of 2)

LzLabs GmbH. Ex. 1016-523

CHANGES IN INSTRUCTIONS PROVIDED

The following figures show those instructions which are optional or not provided in either the System/370 or the 370-XA mode. Those instructions which are standard in both modes are not shown.

Instruction Name*	Mne-monic	Op Code	System/370 Mode	370-XA Mode
BRANCH AND SAVE	BASR	0D	BS	S
BRANCH AND SAVE	BAS	4D	BS	S
BRANCH AND SAVE AND SET MODE	BASSM	0C	-	S
BRANCH AND SET MODE	BSM	0B	-	S
COMPARE AND SWAP	CS	BA	SW	S
COMPARE DOUBLE AND SWAP	CDS	BB	SW	S
DIVIDE (extended)	DXR	B22D	-	S
INSERT PROGRAM MASK	IPM	B222	-	S
MOVE INVERSE	MVCIN	E8	MI	-

Explanation:

- `-` Instruction is not provided.
- `*` Those instructions which are part of the floating-point and extended-precision floating-point facilities in the System/370 mode are standard in the 370-XA mode and are not shown.
- `BS` Branch-and-save facility.
- `MI` Move-inverse facility.
- `S` Instruction is standard.
- `SW` Conditional-swapping facility.

Unprivileged Instructions Provided

LzLabs GmbH. Ex. 1016-524

Instruction Name	Mne-monic	Op Code	System/370 Mode	370-XA Mode
CONNECT CHANNEL SET	CONCS	B200	CS	-
DISCONNECT CHANNEL SET	DISCS	B201	CS	-
EXTRACT PRIMARY ASN	EPAR	B226	DU	S
EXTRACT SECONDARY ASN	ESAR	B227	DU	S
INSERT ADDRESS SPACE CONTROL	IAC	B224	DU	S
INSERT PSW KEY	IPK	B20B	PK	S
INSERT STORAGE KEY	ISK	09	S	-
INSERT STORAGE KEY EXTENDED	ISKE	B229	EK	S
INSERT VIRTUAL STORAGE KEY	IVSK	B223	DU	S
INVALIDATE PAGE TABLE ENTRY	IPTE	B221	EF	S
LOAD ADDRESS SPACE PARAMETERS	LASP	E500	DU	S
LOAD REAL ADDRESS	LRA	B1	TR	S
MOVE TO PRIMARY	MVCP	DA	DU	S
MOVE TO SECONDARY	MVCS	DB	DU	S
MOVE WITH KEY	MVCK	D9	DU	S
PROGRAM CALL	PC	B218	DU	S
PROGRAM TRANSFER	PT	B228	DU	S
PURGE TLB	PTLB	B20D	TR	S
READ DIRECT	RDD	85	DC	-
RESET REFERENCE BIT	RRB	B213	TR	-
RESET REFERENCE BIT EXTENDED	RRBE	B22A	EK	S
SET ADDRESS SPACE CONTROL	SAC	B219	DU	S
SET CLOCK COMPARATOR	SCKC	B206	CK	S
SET CPU TIMER	SPT	B208	CK	S
SET PREFIX	SPX	B210	MP	S
SET PSW KEY FROM ADDRESS	SPKA	B20A	PK	S
SET SECONDARY ASN	SSAR	B225	DU	S
SET STORAGE KEY	SSK	08	S	-
SET STORAGE KEY EXTENDED	SSKE	B22B	EK	S
SIGNAL PROCESSOR	SIGP	AE	MP	S
STORE CLOCK COMPARATOR	STCKC	B207	CK	S
STORE CPU ADDRESS	STAP	B212	MP	S
STORE CPU TIMER	STPT	B209	CK	S
STORE PREFIX	STPX	B211	MP	S
STORE THEN AND SYSTEM MASK	STNSM	AC	TR	S
STORE THEN OR SYSTEM MASK	STOSM	AD	TR	S
TEST BLOCK	TB	B22C	TB	S
TEST PROTECTION	TPROT	E501	EF	S
TRACE	TRACE	99	-	S
WRITE DIRECT	WRD	84	DC	-

Explanation:

- `-` Instruction is not provided.
- `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.
- `MP` Multiprocessing facility.
- `PK` PSW-key-handling facility.
- `S` Instruction is standard.
- `TB` Test-block facility.
- `TR` Translation facility.

Control Instructions Provided

LzLabs GmbH. Ex. 1016-525

Instruction Name	Mne-monic	Op Code	System/370 Mode	370-XA Mode
CLEAR CHANNEL	CLRCH	9F01	RE	-
CLEAR I/O	CLRIO	9D01	S	-
HALT DEVICE	HDV	9E01	S	-
HALT I/O	HIO	9E00	S	-
RESUME I/O	RIO	9C02	SR	-
START I/O	SIO	9C00	S	-
START I/O FAST RELEASE	SIOF	9C01	S	-
STORE CHANNEL ID	STIDC	B203	S	-
TEST CHANNEL	TCH	9F00	S	-
TEST I/O	TIO	9D00	S	-
CLEAR SUBCHANNEL	CSCH	B230	-	S
HALT SUBCHANNEL	HSCH	B231	-	S
MODIFY SUBCHANNEL	MSCH	B232	-	S
RESET CHANNEL PATH	RCHP	B23B	-	S
RESUME SUBCHANNEL	RSCH	B238	-	S
SET ADDRESS LIMIT	SAL	B237	-	S
SET CHANNEL MONITOR	SCHM	B23C	-	S
START SUBCHANNEL	SSCH	B233	-	S
STORE CHANNEL PATH STATUS	STCPS	B23A	-	S
STORE CHANNEL REPORT WORD	STCRW	B239	-	S
STORE SUBCHANNEL	STSCH	B234	-	S
TEST PENDING INTERRUPTION	TPI	B236	-	S
TEST SUBCHANNEL	TSCH	B235	-	S

Explanation:

- - Instruction is not provided.
- RE - Recovery-extension facility.
- S - Instruction is standard.
- SR - Suspend-and-resume facility.

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 the 370-XA mode 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 the System/370 mode, subchannels are not shared among channels, and each subchannel is associated with only one channel path. In 370-XA mode, 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 the 370-XA mode to detect malfunctions, to recover from these malfunctions if possible, and to report to the program via a channel report.

In the System/370 mode, 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 the 370-XA mode, I/O interruptions can be accepted by any CPU in the configuration. The subsystem ID and I/O-interruption parameter are stored at the doubleword at real location 184.

Associated with the new I/O instructions is a new program-interruption condition called operand exception.

LzLabs GmbH. Ex. 1016-526

COMPARISON OF PSW FORMATS

The figure "Comparison of PSW Formats" shows those bits and fields in the PSW which are different between the System/370 mode and the 370-XA mode.

Name of Bit or Field	PSW Bit	System/ 370 Mode	370-XA Mode
PER Mask	1	TR	S
DAT Mode	5	TR	S
EC Mode	12		
Bit 12 = 0 (BC Mode)		S	NP
Bit 12 = 1 (EC Mode)		TR	S[1]
Address-space control	16	DU	S
Addressing mode	32	NP	S
Instruction address	*	S	S

Explanation:

* The instruction address is in PSW bits 40-63 in the System/370 mode and bits 33-63 in the 370-XA mode.

1 In the 370-XA mode, PSW bit 12 must be one, and the term "EC mode" is not used.

DU Provided as part of the dual-address-space facility.

NP Mode is not provided.

S Standard.

TR Provided as part of the translation facility.

Comparison of PSW Formats

CHANGES IN CONTROL-REGISTER ASSIGNMENT

The figure "Differences in Control-Register Assignments" shows those bits and fields in the control registers which are different between the System/370 mode and the 370-XA mode.

LzLabs GmbH. Ex. 1016-527

Name of Bit or Field	Control-Register Position for	
	System/370 Mode	370-XA Mode
Block-multiplexing control	0.0	NP
Fetch-protection override	NP	0.6
Storage-key-exception control	0.7	NP
Page-fault-assist control	0.13	NP
Interval-timer subclass mask	0.24	NP
External-signal subclass mask	0.26	NP
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	NP
Linkage-table origin	5.8 - 5.24	5.1 - 5.24
I/O-interruption subclass mask	NP	6.0 - 6.7
VM assists	6.0	NP
Virtual problem state	6.1	NP
ISK-SSK inhibit	6.2	NP
360 operations only	6.3	NP
SVC inhibit	6.4	NP
Shadow-table-validation inhibit	6.5	NP
Expanded VM and CP	6.6	NP
Virtual interval timer	6.7	NP
Virtual-machine parameter list	6.8 - 6.28	NP
VM extended facility	6.29	NP
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	NP	12.0
Trace-entry address	NP	12.1 - 12.29
ASN-trace control	NP	12.30
Explicit-trace control	NP	12.31
Check-stop control	14.0	NP
Synchronous-MCEL control	14.1	NP
I/O-extended-logout control	14.2	NP
Channel-status subclass mask	NP	14.3
External-damage subclass mask	14.4	*
Timing-facility-damage subclass mask	*	14.4
Asynchronous-MCEL control	14.8	NP
Asynchronous-fixed-log control	14.9	NP
ASN-first-table origin	14.20 - 14.31	14.13 - 14.31
MCEL address	15.8 - 15.28	NP

Explanation:

NP Bit or field is not provided.

* Bit is provided, but with a different name.

Differences in Control-Register Assignments

LzLabs GmbH. Ex. 1016-528

CHANGES IN ASSIGNED STORAGE LOCATIONS

The figure "Differences in Assigned Storage Locations" shows those assigned-storage locations where changes have been made between the System/370 mode and the 370-XA mode.

Name of Bit or Field	Assigned Storage Location and Length* for System/370 Mode		Assigned Storage Location and Length* for 370-XA Mode	
Channel-status word	64	8	NP	
Channel-address word	72	4	NP	
Interval timer	80	4	NP	
Trace-table designation	84	4	NP	
MAPL address	164	4	NP	
Channel ID	168	4	NP	
IOEL address	172	4	NP	
Limited channel logout	176	4	NP	
Subsystem ID	NP		184	4
Measurement-byte	185	1	NP	
I/O address	186	2	NP	
I/O-interruption parameter	NP		188	4
External-damage code	244	4	NP	
Region code	252	4	NP	
Fixed Logout	256	96	256	16
Store-status model-dependent save area	268	4	NP	
CPU identity	795	1	NP	

Explanation:

* The first number is the address, the second the length.
NP Field is not provided.

Differences in Assigned Storage Locations

SIGNAL PROCESSOR CHANGES

The figures "Signal-Processor Orders" and "Signal-Processor Status Bits" show those SIGNAL PROCESSOR orders and status codes where changes have been made between the System/370 mode and the 370-XA mode. In addition to these changes, a parameter is provided as part of the SIGNAL PROCESSOR instruction in the 370-XA mode. The parameter is used by the store-status-at-address and set-prefix orders.

Name of Order	Order Code System/370 Mode	Order Code 370-XA Mode
Initial program reset	07	NP
Program reset	08	NP
Initial microprogram load	0A	NP
Set prefix	NP	0D
Store status at address	NP	0E

Explanation:

NP Order is not provided.

Signal-Processor Orders

Name of Status Bit	Bit Position System/370 Mode	Bit Position 370-XA Mode
Incorrect state	NP	22
Invalid parameter	NP	23
Not ready	28	NP
Inoperative	29	29

Explanation:

NP Status bit is not provided.

Signal-Processor Status Bits

MACHINE-CHECK CHANGES

The figure "Machine-Check-Interruption-Code Bits" summarizes those bits and fields in the machine-check-interruption code (MCIC) where changes have been made between the System/370 mode and the 370-XA mode. In addition to these changes, the region code, the external damage code, the machine-check-extended logout, and asynchronous fixed logouts have been eliminated in the 370-XA mode.

LzLabs GmbH. Ex. 1016-529

Machine-Check-Interruption Condition or Field	MCIC Bits	
	System/370 Mode	370-XA Mode
Interval-timer damage	3	NP
External damage	5	NP
Channel report pending	NP	9
Channel-subsystem damage	NP	11
Delayed	15	NP
Region-code validity	25	NP
External-damage-code validity	26	NP
Logout validity	30	NP
MCEL length	48-63	NP

Explanation:

NP Condition or field is not provided.

Machine-Check-Interruption-Code Bits

CHANGES TO ADDRESSING WRAPAROUND

In the System/370 mode, addresses wrap from $2^{24} - 1$ to zero (or vice versa). In the 370-XA mode, 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 $2^{31} - 1$.

CHANGES TO 31-BIT REAL OPERAND ADDRESSES

The following instructions operate by using 31-bit real addresses in the System/370 mode. In the 370-XA mode, these instructions operate under control of the addressing mode, bit 32 of the PSW. As a result, in 24-bit 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. 1016-530

APPENDIX E. TABLE OF POWERS OF 2

PLUS		*MINUS*
1	0	1.0
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

Powers of 2 (Part 1 of 2)

LzLabs GmbH. Ex. 1016-531

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

Powers of 2 (Part 2 of 2)

LzLabs GmbH. Ex. 1016-532

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.

	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

LzLabs GmbH. Ex. 1016-533

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

	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

LzLabs GmbH. Ex. 1016-534

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

	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

LzLabs GmbH. Ex. 1016-535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

	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

LzLabs GmbH. Ex. 1016-536

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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. 1016-537

Conversion Table: Hexadecimal and Decimal Integers

HALFWORD								HALFWORD							
BYTE				BYTE				BYTE				BYTE			
BITS: 0123		4567		0123		4567		0123		4567		0123		4567	
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 decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that 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 toble 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 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 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

To convert integer numbers greater than the capacity of 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

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

16 | 3380 → remainder 4
16 | 211 → 3
16 | 13 → D

$3380_{10} = D34_{16}$

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. 1016-538

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 6875

Find .00C in position 3 .0029 2968 7500

.ABC Hex 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 table, use techniques below:

HEXADECIMAL FRACTION TO 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: $.8A7 = .540771_{10}$

$8A7_{16} = 2215_{10}$

$16^3 = 4096$

$$4096\overline{)2215.000000}\quad .540771$$

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

.5408 x16

8 ← [8].6528 x16

A ← [10].4448 x16

7 ← [7].1168

LzLabs GmbH. Ex. 1016-539

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

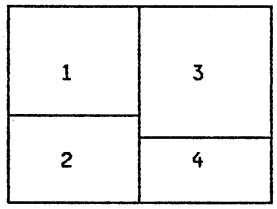
LzLabs GmbH. Ex. 1016-540

APPENDIX G. EBCDIC CHART

EXTENDED BINARY-CODED-DECIMAL INTERCHANGE CODE (EBCDIC)

The 256-position EBCDIC table shows graphic-character, control-character, and formatting-character representations for EBCDIC. The bit-position numbers, bit patterns, hexadecimal representations, and card-hole patterns for these and other possible EBCDIC characters are also shown.

To find the card-hole pattern for most characters, partition the table into four blocks, as follows:



Block 1: Zone punches at top of table; digit punches at left
Block 2: Zone punches at bottom of table; digit punches at left
Block 3: Zone punches at top of table; digit punches at right
Block 4: Zone punches at bottom of table; digit punches at right

Fifteen positions in the table are exceptions to the above arrangement. Each such position is indicated by a circled number in the upper right corner of the box for that position. The card-hole patterns for these positions are shown beneath the table. Bit-position numbers, bit patterns, and hexadecimal representations for these positions are found in the usual manner.

LzLabs GmbH. Ex. 1016-541

The EBCDIC table shows 94 graphic-character positions. Some products have used an 88-character, 63-character, or 62-character subset of these graphic characters.

The 94-character set consists of all graphic characters shown in the EBCDIC table. This character set can be used for interchange with other systems; those systems may use codes, other than EBCDIC, which have 94 graphic characters.

An 88-character set that has been used consists of the 94-character set with the graphic characters at 6A, 79, A1, C0, D0, and E0 hex omitted. This character set has been used for 44-key keyboard applications which require both uppercase and lowercase alphabetic characters.

A 63-character set that has been used consists of the 94-character set with the lowercase alphabetic characters omitted and with the graphic characters at 6A, 79, A1, C0, and D0 hex omitted. This character set has been used for interchange with other systems; those systems may have used codes, other than EBCDIC, which have 63 graphic characters.

A 62-character set that has been used consists of the 63-character set with the graphic character at E0 hex omitted. This character set has been used for 44-key keyboard applications which do not require lowercase alphabetic characters.

Thirteen positions (4A, 4F, 5A, 5B, 5F, 6A, 79, 7B, 7C, A1, C0, D0, and E0 hex) are defined in the table as Data Processing National Use positions. Each such position contains a triangle in the top left corner ( ) of the box for that position. The graphic characters provided in these positions on printing and display devices may differ from one language to another or from one country to another. The characters provided for use in data-processing applications by the English (U.S.) version of EBCDIC are shown in the table.

The other graphic characters shown in the EBCDIC table are provided for data-processing applications in the English (U.S.) version of EBCDIC and in additional versions of EBCDIC in other languages which use a Latin-based alphabet. Products designed for data-processing applications in a language which does not use a Latin-based alphabet support character sets meeting the particular requirements of that language.

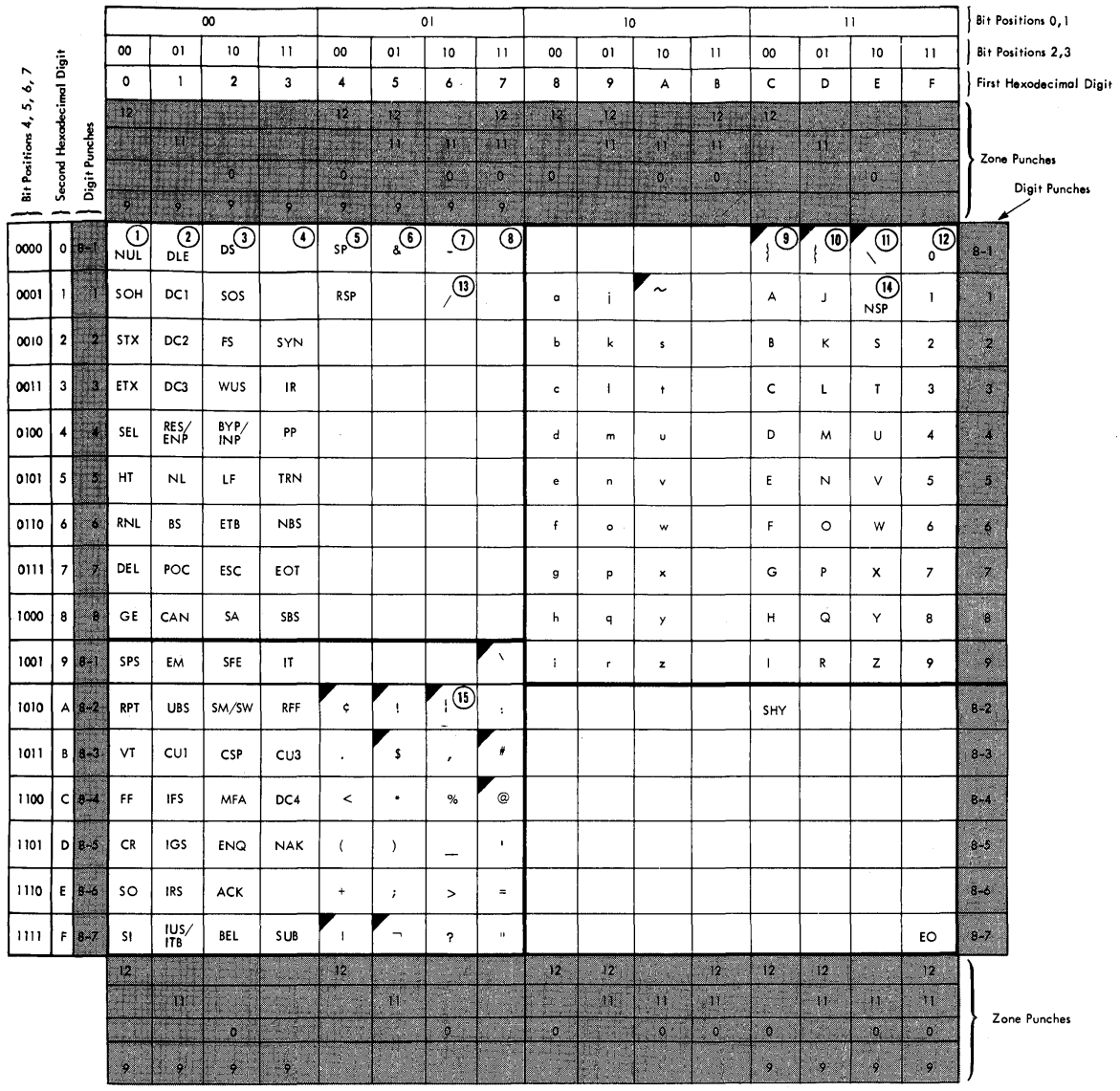
Word-processing products normally support a character set slightly different from the one shown in the table. Additionally, a number of application areas (such as printing and publishing, magnetic-ink character recognition, and some programming languages) also require unique character-set support.

Some examples of the use of the EBCDIC table are shown in the following figure:

Character	Type	Bit Pattern	Hex	Hole Pattern	
				Zone Punches	Digit Punches
SEL	Control Character	00 00 0100	04	12 - 9	- 4
%	Special Graphic	01 10 1100	6C	0	- 8 - 4
R	Upper Case	11 01 1001	D9	11	- 9
a	Lower Case	10 00 0001	81	12 - 0	- 1
	Control Character, function not yet assigned	00 11 0000	30	12 - 11 - 0 - 9	- 8 - 1

Bit Positions
01 23 4567

LzLabs GmbH. Ex. 1016-542



Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	Digit Punches	00 0	00 1	00 2	00 3	01 4	01 5	01 6	01 7	10 8	10 9	10 A	10 B	11 C	11 D	11 E	11 F	Digit Punches
			00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	Bit Positions 2,3
			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	First Hexadecimal Digit
		12					12	12		12	12	12		12	12				Zone Punches
		11						11	11	11		11	11	11		11			
		0					0		0	0	0		0	0			0		
		9		9	9	9	9	9	9	9									
0000	0	8-1	(1) NUL	(2) DLE	(3) DS	(4)	SP (5)	& (6)	- (7)	(8)					(9) {	(10) {	(11) \	(12) 0	8-1
0001	1	1	SOH	DC1	SOS		RSP		/ (13)		a	j	~		A	J	(14) NSP	1	1
0010	2	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2	2
0011	3	3	ETX	DC3	WUS	IR					c	l	t		C	L	T	3	3
0100	4	4	SEL	RES/ ENP	BYP/ INP	PP					d	m	u		D	M	U	4	4
0101	5	5	HT	NL	LF	TRN					e	n	v		E	N	V	5	5
0110	6	6	RNL	BS	ETB	NBS					f	o	w		F	O	W	6	6
0111	7	7	DEL	POC	ESC	EOT					g	p	x		G	P	X	7	7
1000	8	8	GE	CAN	SA	SBS					h	q	y		H	Q	Y	8	8
1001	9	8-1	SPS	EM	SFE	IT				\	i	r	z		I	R	Z	9	9
1010	A	8-2	RPT	UBS	SM/SW	RFF	¢	!	| (15)	:					SHY				8-2
1011	B	8-3	VT	CU1	CSP	CU3	.	$	,	#									8-3
1100	C	8-4	FF	IFS	MFA	DC4	<	*	%	@									8-4
1101	D	8-5	CR	IGS	ENQ	NAK	(	)	_	'									8-5
1110	E	8-6	SO	IRS	ACK		+	;	>	=									8-6
1111	F	8-7	SI	IUS/ ITB	BEL	SUB	|	¬	?	"								EO	8-7
		12					12				12	12		12	12	12		12	Zone Punches
		11						11				11	11	11		11	11	11	
		0							0		0		0	0	0		0	0	
		9	9	9	9										9	9	9	9	

Card Hole Patterns

(1) 12-0-9-8-1	(4) 12-11-0-9-8-1	(7) 11	(10) 11-0	(13) 0-1			
(2) 12-11-9-8-1	(5) No Punches	(8) 12-11-0	(11) 0-8-2	(14) 11-0-9-1			
(3) 11-0-9-8-1	(6) 12	(9) 12-0	(12) 0	(15) 12-11			

Formatting Character Representations

NSP	Numeric Space
RSP	Required Space
SP	Space
SHY	Syllable Hyphen

Control Character Representations

ACK	Acknowledge	ETX	End of Text	RFF	Required Form Feed
BEL	Bell	FF	Form Feed	RNL	Required New Line
BS	Backspace	FS	Field Separator	RPT	Repeat
BYP/INP	Bypass/Inhibit Presentation	GE	Graphic Escape	SA	Set Attribute
CAN	Cancel	HT	Horizontal Tab	SBS	Subscript
CR	Carriage Return	IFS	Interchange File Separator	SEL	Select
CSP	Control Sequence Prefix	IGS	Interchange Group Separator	SFE	Start Field Extended
CU1	Customer Use 1	IR	Index Return	SI	Shift In
CU3	Customer Use 3	IRS	Interchange Record Seporator	SM/SW	Set Mode/Switch
DC1	Device Control 1	IT	Indent Tab	SO	Shift Out
DC2	Device Control 2	IUS/ITB	Interchange Unit Separator/ Intermediate Transmission Block	SOH	Start of Heading
DC3	Device Control 3			SOS	Start of Significance
DC4	Device Control 4	LF	Line Feed	SPS	Superscript
DEL	Delete	MFA	Modify Field Attribute	STX	Start of Text
DLE	Data Link Escape	NAK	Negative Acknowledge	SUB	Substitute
DS	Digit Select	NBS	Numeric Backspace	SYN	Synchronous Idle
EM	End of Medium	NL	New Line	TRN	Transparent
ENQ	Enquiry	NUL	Null	UBS	Unit Backspace
EO	Eight Ones	POC	Progrom-Operator Communication	VT	Vertical Tab
EOT	End of Transmission			WUS	Word Underscore
ESC	Escape	PP	Presentation Position		
ETB	End of Transmission Block	RES/ENP	Restore/Enable Presentation		

Special Graphic Characters

¢	Cent Sign	,	Comma
.	Period, Decimal Point	%	Percent
<	Less-than Sign	_	Underscore
(	Left Parenthesis	>	Greater-than Sign
+	Plus Sign	?	Question Mark
|	Logical OR	`	Grave Accent
&	Ampersand	:	Colon
!	Exclamation Point	#	Number Sign
$	Dollar Sign	@	At Sign
*	Asterisk	'	Prime, Apostrophe
)	Right Parenthesis	=	Equal Sign
;	Semicolon	"	Quotation Mark
¬	Logical NOT	~	Tilde
-	Minus Sign, Hyphen	{	Opening Brace
/	Slash	}	Closing Brace
!	Vertical Line	\	Reverse Slant

LzLabs GmbH. Ex. 1016-543

A
A (ADD) binary instruction 7-8
abbreviations, list of vii
absolute address 3-4
absolute storage 3-4
access-control bits in storage key 3-8
access exceptions 6-24
  priority of 6-26
access key 3-9
  for channel-program execution 3-10
  for channel-subsystem monitoring 3-10
  for CPU 3-10
access, storage 5-21
  (See also reference)
active allegiance 15-9
activity-control field (SCSW) 16-14
AD (ADD NORMALIZED) instruction 9-6
  example A-34
ADD (A,AR) binary instructions 7-8
ADD DECIMAL (AP) instruction 8-5
  example A-28
ADD HALFWORD (AH) instruction 7-8
  example A-8
ADD LOGICAL (AL,ALR) instructions 7-9
ADD NORMALIZED (AD,ADR,AE,AER,AXR) instructions 9-6
  example A-34
ADD UNNORMALIZED (AU,AUR,AW,AWR) instructions 9-7
  example A-34
address 3-2
  absolute 3-4
  arithmetic 3-5,5-5
    unsigned binary 7-3
  base 5-5
  channel program
    in ORB 15-20
    used for IPL 17-13
  comparison 12-1
    effect on CPU state 4-2
  CPU 4-30
  data (I/O) 15-22
    invalid specification 16-25
  effective (See effective address)
  failing-storage (See failing-storage address)
  format 3-2
  generation 5-4
    for storage addressing 3-6
  I/O (See I/O addressing)
  instruction (See instruction address)
  instruction 3-5
  invalid 6-13
  limit-checking control (I/O)
    in IRB 16-12
    in ORB 15-19
    used for IPL 17-13
  limit-checking facility (I/O)
    definition 17-1
    effect of I/O-system reset 17-11
    limit mode (bits in PMCW) 15-3
  logical (See logical address)
  numbering of byte locations 3-2
  PER 4-12
  primary virtual 3-5
  real 3-4
  secondary virtual 3-5
  size of 3-6
    controlled by addressing mode 5-4
  storage 3-2
  summary information 3-36
  transformation 3-4
    by DAT 3-21
    by prefixing 3-12
  translation (DAT) 3-21
    by LOAD REAL ADDRESS instruction 10-18
    control of 3-22
  type of 3-4
  virtual 3-5
  wraparound (See wraparound)
  24-bit and 31-bit D-1,3-2,5-5
address-compare controls 12-1
address space 3-13
  control bit
    in PSW 4-6
    use in address translation 3-22
  created by DAT 3-21
  number (See ASN)
addressing exception 6-13
  as an access exception 6-24
addressing mode 5-4,5-5
  bit in entry-table entry 5-19
  bit in PSW 4-6
  effect on address size 3-6
  effect on operand-address generation 5-5
  effect on sequential instruction-address generation 5-5
  effect on wraparound 3-6
  in examples A-8
  programming note describing use of 5-8
  with BRANCH AND SAVE AND SET MODE instruction 7-11
  with BRANCH AND SET MODE instruction 7-11
ADR (ADD NORMALIZED) instruction 9-6
AE (ADD NORMALIZED) instruction 9-6
  example A-34
AER (ADD NORMALIZED) instruction 9-6
AFT (ASN first table) 3-15
AFTE (ASN-first-table entry) 3-15
AFTO (ASN-first-table origin) 3-14
AFX (ASN-first-table index) 3-14
  invalid bit 3-15
  translation exception 6-15
AH (ADD HALFWORD) instruction 7-8
  example A-8
AKM (authorization key mask) 5-19
AL (ADD LOGICAL) instruction 7-9
alert
  as class of machine-check conditions 11-11
  interruption condition (I/O) 16-5
  status bit (I/O) 16-16
allegiance
  active 15-9
  channel path 15-9
  dedicated 15-10
  effect of CLEAR SUBCHANNEL 15-9
  working 15-9
allowed interruptions 6-5
ALR (ADD LOGICAL) instruction 7-9

LzLabs GmbH. Ex. 1016-545

alter-and-display controls 12-2
alteration
  general-register (PER event) 4-15
  storage (PER event) 4-15
AND (N,NC,NI,NR) instructions 7-9
  examples A-8
AP (ADD DECIMAL) instruction 8-5
  example A-28
AR (ADD) binary instruction 7-8
architectural mode 1-1
  indication of 12-2
  selection by
    architectural-mode-selection controls 12-2
    IML controls 12-3
arithmetic
  binary 7-3
  decimal (See decimal instructions)
  floating-point (See floating-point instructions)
  logical (See unsigned binary arithmetic)
ASCII, handled by architecture iv
ASN (address-space number) 3-13
  authorization 3-18
  first table (AFT) 3-15
    index (AFX) 3-14
    origin (AFTO) 3-14
  in entry-table entry 5-19
  second table (AST) 3-15
    index (ASX) 3-14
    origin (ASTO) 3-15
  trace-control bit 4-8
  translation 3-14
    exceptions 6-30
    specification exception 6-15
  translation controls 3-14
ASN-translation-control bit 3-14,5-15
assembler language A-7
  instruction formats in (See individual instruction descriptions)
assigned storage locations 3-38
  comparison with System/370 D-9
assists for MVS/XA v
AST (ASN second table) 3-15
ASTE (ASN-second-table entry) 3-15
ASTO (ASN-second-table origin) 3-15
ASX (ASN-second-table index) 3-14
  invalid bit 3-15
  translation exception 6-15
AT (authority table) 5-16
ATL (authority-table length) 3-15
ATO (authority-table origin) 3-15
attached segment-table or page-table entry 3-31
attachment of I/O devices 13-3
attention (I/O-device status) 16-19
AU (ADD UNNORMALIZED) instruction 9-7
  example A-34
AUR (ADD UNNORMALIZED) instruction 9-7
authority
  table (AT) 5-16
    designation 3-15
  testing for 3-18
authorization
  index (AX) 3-18,5-16
  key mask (AKM) 5-19
auxiliary storage 3-2,3-21
AW (ADD UNNORMALIZED) instruction 9-7
AWR (ADD UNNORMALIZED) instruction 9-7
AX (authorization index) 5-16
AXR (ADD NORMALIZED) instruction 9-6

B
B field of instruction 5-5
backed-up bit 11-17
backup condition 11-17
BAL (BRANCH AND LINK) instruction 7-10
  example A-9
BALR (BRANCH AND LINK) instruction 7-10
  example A-9
BAS (BRANCH AND SAVE) instruction 7-10
base address 5-5
  register 2-3
basic I/O functions 15-1
basic operator facilities 12-1
basic sense command 15-36
BASR (BRANCH AND SAVE) instruction 7-10
BASSM (BRANCH AND SAVE AND SET MODE) instruction 7-11
BC (BRANCH ON CONDITION) instruction 7-12
  example A-9
BCR (BRANCH ON CONDITION) instruction 7-12
BCT (BRANCH ON COUNT) instruction 7-13
  example A-10
BCTR (BRANCH ON COUNT) instruction 7-13
  example A-10
bimodal addressing D-1,5-4
  (See also addressing mode)
binary
  (See also fixed point)
  arithmetic 7-3
  negative zero 7-2
  number representation 7-2
    examples A-2
  overflow 7-3
    example A-2
  sign bit 7-2
binary-to-decimal conversion 7-20
bit numbering 3-2
block-concurrent storage references 5-27
block of I/O data 15-18
block of storage 3-4
  (See also page)
  testing for usability 10-45
borrow 7-41
boundary alignment 3-3
  for instructions 5-2
branch address 5-6
  in trace entry 4-10
BRANCH AND LINK (BAL,BALR) instructions 7-10
  examples A-9
BRANCH AND SAVE (BAS,BASR) instructions 7-10
BRANCH AND SAVE AND SET MODE (BASSM) instruction 7-11
BRANCH AND SET MODE (BSM) instruction 7-11
BRANCH ON CONDITION (BC,BCR) instructions 7-12
  example A-9
BRANCH ON COUNT (BCT,BCTR) instructions 7-13
  example A-10
BRANCH ON INDEX HIGH (BXH) instruction 7-13
  examples A-10
BRANCH ON INDEX LOW OR EQUAL (BXLE) instruction 7-13
branch-trace-control bit 4-8

LzLabs GmbH. Ex. 1016-546

branching 5-6
BSM (BRANCH AND SET MODE) instruction 7-11
buffer storage (cache) 3-2
burst mode (channel-path operation) 13-3
bus-out check (bit in I/O sense data) 15-37
busy
  as I/O-device status 16-21
  control unit 16-19,16-21
  handling in I/O 13-6
  in I/O operations 13-9
  in SIGNAL PROCESSOR 4-33
BXH (BRANCH ON INDEX HIGH) instruction 7-13
  examples A-10
BXLE (BRANCH ON INDEX LOW OR EQUAL) instruction 7-13
byte 3-2
byte index (BX) 3-22
byte-multiplex mode (channel-path operation) 13-3

C
C (COMPARE) binary instruction 7-14
cache 3-2
carry 7-3
CBC (checking-block code) 11-2
  in registers 11-9
  in storage 11-6
  in storage keys 11-7
CCW (channel-command word) 15-20
  address
    in ORB 15-20
    in SCSW 16-28
  byte count 15-21
  chaining 15-23
    command 13-8,15-21,15-27
    data 13-8,15-21,15-26
  check (in subchannel logout) 16-37
  command codes (See I/O commands)
  current 15-20
  designation of storage area 15-21,15-22
  format control
    in IRB 16-11
    in ORB 15-19
    used for IPL 17-13
  format-0 and -1 contents 15-21
  in IPL, assigned storage locations for 3-38
  indirect data addressing 13-7,15-30
  invalid address 16-24
  invalid format 16-25
  prefetch control
    in IRB 16-11
    in ORB 15-19
    used for IPL 17-13
  prefetching 15-27
  role in I/O operations 13-6
CD (COMPARE) floating-point instruction 9-8
CDR (COMPARE) floating-point instruction 9-8
  examples A-35
CDS (COMPARE DOUBLE AND SWAP) instruction 7-14
CE (COMPARE) floating-point instruction 9-8
central processing unit (See CPU)
CER (COMPARE) floating-point instruction 9-8
CH (COMPARE HALFWORD) instruction 7-16
  example A-11
chaining
  check (I/O-subchannel status) 16-27
  of CCWs 13-8,15-23
    command 15-21,15-27
    data 15-21,15-26
  of CRWs 17-16,17-17
change
  bit in storage key 3-8
  recording 3-11
channel
  command word (See CCW)
  commands (See I/O commands)
  control check (I/O-subchannel status) 16-26
  data check (I/O-subchannel status) 16-26
  end (I/O-device status) 16-21
  program 13-6,15-20
    requirements for RESUME SUBCHANNEL 14-8
  program serialization 5-30
  report 17-15
    generated as a result of RCHP 14-7
    pending 17-15
  report word (See CRW)
channel path 13-1,13-3,13-4
  active allegiance 15-9
  available for selection 15-10
  dedicated allegiance 15-10
  effect of I/O-system reset 17-10
  identifier (See CHPID)
  not operational 16-13
  reset 17-9
    effect of I/O-system reset 17-11
  reset function
    execution of 15-42
    initiation by RESET CHANNEL PATH 14-6
    reset signal 17-8
  status word 14-13
  storing of status 14-13
  types 13-5
  working allegiance 15-9
channel report pending 11-17
  subclass-mask bit 11-22
channel subsystem 2-5
  addressing 13-5
  damage 11-17
  effect of I/O-system reset 17-9
  monitoring facilities 17-2
    effect of I/O-system reset 17-11
  operations 15-17
  power-on reset 4-29
  recovery 11-4,17-14
  structure 13-2
  timer 17-2
    effect of I/O-system reset 17-11
  timing facility 17-2
characteristic (of floating-point number) 9-1
check bits 3-2,11-2
check stop 11-10
  as signal-processor status 4-35
  indicator 12-2
  state 4-2
    due to malfunctioning manual operation 12-1
    effect on CPU timer 4-23
    entering of 11-10,11-13

LzLabs GmbH. Ex. 1016-547

malfunction alert when entering 6-11
system 11-11
checking block 11-2
code (See CBC)
checkpoint 11-2
synchronization 11-3
CHPID (channel-path identifier) 13-5
used in RESET CHANNEL PATH 14-6
CHPID (channel-path identifier) in PMCW 15-5
CL (COMPARE LOGICAL) instruction 7-16
CLC (COMPARE LOGICAL) instruction 7-16
example A-11
CLCL (COMPARE LOGICAL LONG) instruction 7-17
example A-13
clear function
clear signal 17-8
execution of 15-11
indication of 16-13
initiated by CLEAR SUBCHANNEL 14-3
pending 16-15
signaling 15-12
subchannel modification 15-12
clear reset 4-28
clear signal 17-8
issued as part of CSCH 15-12
CLEAR SUBCHANNEL (CSCH) instruction 14-3
after RESET CHANNEL PATH 14-7
clear signal 17-8
effect on device status 15-12
function initiated by 15-11
path management for 15-11
clearing operation
by clear-reset function 4-28
by load-clear key 12-3
by system-reset-clear key 12-5
by TEST BLOCK instruction 10-45
CLI (COMPARE LOGICAL) instruction 7-16
example A-12
CLM (COMPARE LOGICAL CHARACTERS UNDER MASK) instruction 7-17
example A-12
clock (See TOD clock)
clock comparator 4-21
external interruption 6-9
save areas for 3-40
subclass-mask bit 6-10
validity bit for 11-20
clock unit 4-21
CLR (COMPARE LOGICAL) instruction 7-16
example A-12
code
checking-block (See CBC)
condition (See condition code)
decimal digit and sign 8-2
instruction-length (See ILC)
interruption (See interruption code)
monitor (See monitor code)
operation 5-2
PER (See PER code)
version 10-43
codes, interruption, summary of 6-2
command
chaining 13-8,15-27
(See also chaining of CCWs)
effect of status modifier 15-28
code in CCW 15-21,15-22
invalid 16-24
reject (bit in I/O sense data) 15-36
retry 15-39
effect on PCI 15-29
commands (See I/O commands)
common-segment bit 3-24
COMPARE (C,CR) binary instructions 7-14
COMPARE (CD,CDR,CE,CER) floating-point instructions 9-8
examples A-35
COMPARE AND SWAP (CS) instruction 7-14
examples A-38
COMPARE DECIMAL (CP) instruction 8-5
example A-29
COMPARE DOUBLE AND SWAP (CDS) instruction 7-14
COMPARE HALFWORD (CH) instruction 7-16
example A-11
COMPARE LOGICAL (CL,CLC,CLI,CLR) instructions 7-16
examples A-11
COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) instruction 7-17
example A-12
COMPARE LOGICAL LONG (CLCL) instruction 7-17
example A-13
comparison
address 12-1
decimal 8-5
floating-point 9-8
logical 7-4
signed-binary 7-4
TOD-clock 4-22
comparison between System/370 and 370-XA modes D-1
compatibility 1-3
I/O-operation 13-1
completion
of instruction execution 5-11
of unit of operation 5-12
conceptual sequence 5-21
effect on storage-operand accesses 5-28
conclusion
of I/O operations 13-7,16-1
during data transfer 15-41
during initiation 15-39
for immediate commands 15-40
of instruction execution 5-11
concurrency of storage references 5-27
condition code 5-6
deferred 16-9
for I/O operations 14-2
in PSW 4-6
summary C-1
tested by BRANCH ON CONDITION instruction 7-12
validity bit for 11-19
conditions for interruption (See interruption)
configuration 2-2
of storage 3-4
connective (See logical connective)
consistency (storage operand) 5-27
example A-42,A-45
console device 12-1
control 4-1
as an I/O command 15-35
instructions 10-1
manual (See manual operations)
register 2-3
comparison with System/370 D-7
description and assignments 4-6
save areas for 3-41
validity bit for 11-20

LzLabs GmbH. Ex. 1016-548

control-register assignment
(CRx.y indicates control register x, bit position y)
CR0.1:
SSM-suppression-control bit 6-22,10-40
CR0.2:
TOD-clock-sync-control bit 4-19,4-21
CR0.3:
low-address-protection-control bit 3-11
CR0.4:
extraction-authority-control bit 5-15
CR0.5:
secondary-space-control bit 3-23,5-15
CR0.6:
fetch-protection-override control bit 3-10
CR0.8-12:
translation format 3-23
CR0.16:
malfunction-alert subclass-mask bit 6-11
CR0.17:
emergency-signal subclass-mask bit 6-10
CR0.18:
external-call subclass-mask bit 6-10
CR0.19:
TOD-clock sync-check subclass-mask bit 6-11
CR0.20:
clock-comparator subclass-mask bit 6-10
CR0.21:
CPU-timer subclass-mask bit 6-10
CR0.22:
service-signal subclass-mask bit 6-11
CR0.25:
interrupt-key subclass-mask bit 6-10
CR1.0:
space-switch-event-control bit 3-23,6-21
CR1.1-19:
primary segment-table origin (PSTO) 3-23
CR1.25-31:
primary segment-table length (PSTL) 3-23
CR3.0-15:
PSW-key mask (PKM) 5-15
CR3.16-31:
secondary ASN (SASN) 3-14
CR4.0-15:
authorization index (AX) 3-18,5-16
CR4.16-31:
primary ASN (PASN) 3-14
CR5.0:
subsystem-linkage-control bit 5-15,5-18
CR5.1-24:
linkage-table origin (LTO) 5-18
CR5.25-31:
linkage-table length (LTL) 5-18
CR6.0-7:
I/O-interruption subclass-mask bits 6-12
CR7.1-19:
secondary segment-table origin (SSTO) 3-24
CR7.25-31:
secondary segment-table length (SSTL) 3-24
CR8.16-31:
monitor-mask bits 6-18
CR9.0:
PER successful-branching-event-mask bit 4-11
CR9.1:
PER instruction-fetching-event-mask bit 4-11
CR9.2:
PER storage-alteration-event-mask bit 4-11
CR9.3:
PER general-register-alteration-event-mask bit 4-11
CR9.16-31:
PER general-register-mask bits 4-11
CR10.1-31:
PER starting address 4-11
CR11.1-31:
PER ending address 4-11
CR12.0:
branch-trace-control bit 4-8
CR12.1-29:
trace-entry address 4-8
CR12.30:
ASN-trace-control bit 4-8
CR12.31:
explicit-trace-control bit 4-8
CR14.3:
channel-report-pending subclass-mask bit 11-22
CR14.4:
recovery subclass-mask bit 11-22
CR14.5:
degradation subclass-mask bit 11-22
CR14.6:
timing-facility-damage subclass-mask bit 11-22
CR14.7:
warning subclass-mask bit 11-22
CR14.12:
ASN-translation-control bit 3-14,5-15
CR14.13-31:
ASN-first-table origin (AFTO) 3-14
control unit 2-5,13-4
busy 16-19,16-21
effect of I/O-system reset 17-9
end (I/O-device status) 16-19
model and type number in basic sense data 15-38
sharing of 13-4
type 15-11
conversion
binary-to-decimal 7-20
decimal-to-binary 7-19
floating-point-number
basic example A-7
instruction-sequence examples A-36
CONVERT TO BINARY (CVB) instruction 7-19

LzLabs GmbH. Ex. 1016-549

example A-14
CONVERT TO DECIMAL (CVD) instruction 7-20
example A-14
count field
in CCW 15-21
invalid 16-24
in SCSW 16-33
counter updating (example) A-39
counting operations 7-13
CP (COMPARE DECIMAL) instruction 8-5
example A-29
CPU (central processing unit) 2-2
address 4-30
assigned storage locations for 3-39
when stored during external interruptions 6-9
checkpoint 11-2
hangup due to string of interruptions 4-3
identification (ID) 10-43
model number 10-43
power-on reset 4-29
registers 2-2
save areas for 3-40
reset 4-27
as signal-processor order 4-32
retry 11-2
serialization 5-29
signaling 4-31
state 4-2
no effect on TOD clock 4-18
timer 4-22
external interruption 6-10
save areas for 3-40
subclass-mask bit 6-10
validity bit for 11-20
version code 10-43
CR (See control register)
CR (COMPARE) binary instruction 7-14
CRW (channel-report word) 17-15
chaining 17-16,17-17
contents of 17-16
overflow 17-17
solicited 17-16
storing 14-13
CS (COMPARE AND SWAP) instruction 7-14
examples A-38
CSCH (See CLEAR SUBCHANNEL instruction)
current CCW 15-20
current PSW 4-3,5-6
(See also PSW)
stored during interruption 6-2
CVB (CONVERT TO BINARY) instruction 7-19
example A-14
CVD (CONVERT TO DECIMAL) instruction 7-20
example A-14

D
D (DIVIDE) binary instruction 7-20
example A-15
D field of instruction 5-5
damage
channel-subsystem 11-17
instruction-processing 11-15
processing 11-18
system 11-15
timing-facility 11-16
mask bit for 11-22
DAT (See dynamic address translation)
DAT mode (bit in PSW) 4-5
use in address translation 3-22
data
address (I/O) 15-22
invalid 16-25
chaining of (I/O) 13-7,15-26
check
bit in I/O sense data 15-37
measurement block 16-36
exception 6-15
format for
decimal instructions 8-1
floating-point instructions 9-2
general instructions 7-2
indirect addressing of (I/O) 13-7,15-30
prefetching for I/O operation 15-23
streaming (I/O) 13-3
effect of CCW count on 15-27
DCTI (device-connect-time interval)
in ESW 16-40
in measurement block 17-5
DD (DIVIDE) floating-point instruction 9-8
DDR (DIVIDE) floating-point instruction 9-8
DE (DIVIDE) floating-point instruction 9-8
decimal
comparison 8-5
digit codes 8-2
divide exception 6-16
instructions 8-1
examples A-28
number representation 8-1
examples A-5
operand overlap 8-3
overflow
exception 6-16
mask in PSW 4-6
sign codes 8-2
decimal-to-binary conversion 7-19
dedicated allegiance 15-10
deferred condition code 16-9
degradation (machine-check condition) 11-16
degradation subclass-mask bit 11-22
delay in storing 5-25
deletion of malfunctioning unit 11-4
DER (DIVIDE) floating-point instruction 9-8
examples A-35
designation (origin and length)
authority-table 3-15
entry-table 5-18
linkage-table 5-18
in AST entry 3-16
of storage area for data (I/O) 15-22
page-table 3-24
primary segment-table 3-23
secondary segment-table 3-24
segment-table 3-23
in AST entry 3-15
destructive overlap 5-28,7-28
device 2-5,13-4
active 16-15
address 13-5
busy 16-21
connect-time interval (DCTI)
in ESW 16-40
in measurement block 17-5
connect-time measurement 17-7
effect of suspension 15-32

LzLabs GmbH. Ex. 1016-550

enable 15-3,17-7
console 12-1
disconnect-time interval, in measurement block 17-5
effect of I/O-system reset 17-9
end (I/O-device status) 16-21
model and type number in basic sense data 15-38
not ready 15-36
number 13-5
assignment of 13-6
in PMCW 15-4
valid bit in PMCW 15-4
ready 16-22
status conditions 16-18
DIAGNOSE instruction 10-4
used for service-processor communication 4-23
digit codes (decimal) 8-2
digit selector 8-6
direct-access storage 3-2
disabling, for interruptions 6-5
disallowed interruptions 6-5
displacement 5-5
display (manual controls) 12-2
DIVIDE (D,DR) binary instructions 7-20
example A-15
DIVIDE (DD,DDR,DE,DER, DXR) floating-point instructions 9-8
DIVIDE (DD,DDR,DE,DER) floating-point instructions, examples A-35
DIVIDE DECIMAL (DP) instruction 8-5
example A-29
divide exception
decimal 6-16
fixed-point 6-17
floating-point 6-17
doubleword 3-3
concurrency of reference 5-27
DP (DIVIDE DECIMAL) instruction 8-5
example A-29
DR (DIVIDE) binary instruction 7-20
dump (standalone) 12-4
dynamic address translation (DAT) 3-21
mode bit in PSW 4-5
sequence of table fetches 5-24

E
early exception recognition 6-8
EBCDIC, handled by architecture iv
ECC (error checking and correction) 11-2
ECW (extended-control word) 16-8,16-43
format bit in SCSW 16-12
ED (EDIT) instruction 8-6
examples A-29
EDIT (ED) instruction 8-6
examples A-29
EDIT AND MARK (EDMK) instruction 8-9
example A-31
editing instructions 8-3
EDMK (EDIT AND MARK) instruction 8-9
example A-31
effective address 3-5
controlled by addressing mode 5-4
generation 5-4
used for storage interlocks 5-22
effective segment-table designation 3-26
EKM (entry key mask) 5-19
emergency signal
as signal-processor order 4-31
external interruption 6-10
subclass-mask bit 6-10
enabled (bit in PMCW) 15-2
enabling
for interruptions 6-5
of subchannel 15-2,16-6
ending of instruction execution 5-11
entry
index (EX) 5-17
table (ET) 5-18
designation 5-18
trace 4-9
EPAR (EXTRACT PRIMARY ASN) instruction 10-5
epoch (for TOD clock) 4-20
equipment check
as signal-processor status 4-35
bit in I/O sense data 15-37
ERC (error-recovery code) 17-17
error
checking and correction 11-2
effect of DIAGNOSE instruction 10-5
I/O alert 16-38
in PSW format 6-8
intermittent 11-5
recovery code (ERC) 17-17
solid 11-5
state of TOD clock 4-19
storage 11-18
storage-key 11-18
ESAR (EXTRACT SECONDARY ASN) instruction 10-5
ESW (extended-status word) 16-8,16-35
format bit in SCSW 16-8
format 0 16-36
format 1 16-40
format 2 16-40
format 3 16-41
ET (entry table) 5-18
ETL (entry-table length) 5-18
ETO (entry-table origin) 5-18
event 6-12
monitor 7-26
PER 4-11
space-switch 6-21
EX (entry index) 5-17
translation exception 6-17
EX (EXECUTE) (See EXECUTE instruction)
exceptions 6-12
access 6-24
addressing 6-13
AFX-translation 6-15
ASN-translation 6-30
ASN-translation-specification 6-15
associated with PSW 6-8
ASX-translation 6-15
data (decimal) 6-15
decimal-divide 6-16
decimal-overflow 6-16
early recognition of 6-8
EX-translation 6-17
execute 6-16
exponent-overflow 6-16
exponent-underflow 6-16
fixed-point-divide 6-17
fixed-point-overflow 6-17
floating-point-divide 6-17
for invalid translation addresses and formats 3-30
late recognition of 6-8
LX-translation 6-17
operand 6-18
operation 6-18
page-translation 6-19

LzLabs GmbH. Ex. 1016-551

PC-translation-specification 6-19
primary-authority 6-19
privileged-operation 6-20
protection 6-20
secondary-authority 6-21
segment-translation 6-21
significance 6-21
special-operation 6-22
specification 6-22
trace 6-31
trace-table 6-23
translation-specification 6-23
EXCLUSIVE OR (X,XC,XI,XR) instructions 7-21
examples A-15
EXECUTE (EX) instruction 7-22
effect of address comparison 12-1
example A-17
exceptions while fetching target 6-7
PER event for target 4-14
execute exception 6-16
exigent machine-check condition 11-11
explicit-trace-control bit 4-9
exponent 9-1
(See also floating point)
overflow 9-1
exception 6-16
underflow 9-1
exception 6-16
mask in PSW 4-6
extended-control word (See ECW)
extended floating-point number 9-2
extended-status flags (in subchannel logout) 16-36
extended-status word (See ESW)
external
call
as signal-processor order 4-31
external interruption due to 6-10
pending (signal-processor status) 4-35
subclass-mask bit 6-10
interruption 6-9
clock-comparator 4-22,6-9
CPU-timer 4-22,6-10
emergency-signal 6-10
external-call 6-10
interrupt-key 6-10
malfunction-alert 6-11
service-signal 4-23,6-11
TOD-clock-sync-check 6-11
mask in PSW 4-5
externally initiated functions 4-23
I/O 17-12
EXTRACT PRIMARY ASN (EPAR) instruction 10-5
EXTRACT SECONDARY ASN (ESAR) instruction 10-5
extraction-authority-control bit 5-15

F
facilities, comparison with System/370 D-1
failing-storage address 11-21
assigned storage locations for 3-41
validity bit for 11-19
fetch protection 3-9
bit in storage key 3-8
override-control bit 3-10
fetch reference 5-25
access exceptions for 6-26
fetching
of DAT-table entries 5-24
of instructions 5-23
field 3-2
field separator 8-6
field-validity flags (in subchannel logout) 16-37
fill byte 8-6
fixed-length field 3-2
fixed logout
assigned storage locations for 3-41
machine-check 11-22
fixed point
(See also binary)
divide exception 6-17
overflow exception 6-17
mask in PSW 4-6
floating interruption conditions 6-6,11-21
clearing of 4-28
floating point
(See also exponent)
comparison 9-8
conversion
basic example A-7
instruction-sequence examples A-36
data format 9-2
divide exception 6-17
instructions 9-1
examples A-34
numbers 9-1
examples A-5
register 2-3
save areas for 3-41
validity bit for 11-19
shifting (See normalization)
format
CCW (channel-command word) 15-21
control 15-35
read 15-34
read backward 15-35
sense 15-36
sense ID 15-38
write 15-34
decimal data 8-1
floating-point data 9-2
general data 7-2
IDAW 15-30
information 3-2
instruction 5-2
PSW 4-5
fraction 9-1
fullword (See word)
function control 16-13
clear function 16-13
halt function 16-13
start function 16-13
function-pending time
accumulated 17-4
in measurement block 17-5

G
G (giga), multiplier iv
general instructions 7-2
examples A-8
general registers 2-3
alteration-event-mask bit 4-11
alteration of (PER event) 4-15
PER-mask bits 4-11
save areas for 3-41
validity bit for 11-19
glue module 5-9
guard digit 9-3

LzLabs GmbH. Ex. 1016-552

H
halfword 3-3
concurrency of reference 5-27
halt function
execution of 15-13
halt signal 17-8
indication of 16-13
initiated by HALT SUBCHANNEL 14-4
pending 16-15
signaling 15-14
halt signal 17-8
issued as part of HSCH 15-14
HALT SUBCHANNEL (HSCH) instruction 14-4
after RESET CHANNEL PATH 14-7
effect on SCSW count field 15-15
function initiated by 15-13
halt signal 17-8
path management for 15-13
HALVE (HDR,HER) instructions 9-10
example A-36
HDR (HALVE) instruction 9-10
example A-36
HER (HALVE) instruction 9-10
hexadecimal (hex) representation 5-3
high-speed data transfer (I/O) 13-3
HSCH (See HALT SUBCHANNEL instruction)

I
I field of instruction 5-4
I/O (input/output) 2-5
address-limit-checking facility 17-1
addressing 13-5
channel-path identifier (See CHPID)
device address 13-5
device number 13-5
subchannel number 13-5
basic functions 15-1
commands 15-22,15-32
(See also particular command)
comparison with System/370 D-6
control unit (See control unit)
data blocking 15-18
device (See device)
effect on CPU timer 4-22
error
alert (in subchannel logout) 16-38
with machine check 11-5
externally initiated functions 17-12
I/O-system reconfiguration 17-14
initial program loading (IPL) 17-12
instructions 14-1,14-2
condition code 14-2
deferred condition code 16-9
format 14-1
interface, OEMI v
interruption 6-11,16-1
(See also interruption)
action 16-6
alert condition 16-5
clearing of 13-8
conditions 13-8,16-2
initial-status control (in IRB) 16-12
initial-status control (in ORB) 15-19
initial-status indication (in IRB) 16-12
intermediate condition 16-4
masks 13-8
parameter 15-2,15-19
parameter, assigned storage locations for 3-40
primary condition 13-7,16-4
priority 16-5
program-controlled (See PCI)
secondary condition 13-7,16-4
solicited condition 16-3
subclass 13-8,16-5
subclass code in PMCW 15-2
subclass mask 16-6
subclass-mask bits 6-12
unsolicited condition 16-3
zero condition code (in IRB) 16-12
mask in PSW 4-5
monitoring (See measurement)
operations 13-6,15-17
conclusion (See conclusion of I/O operations)
immediate 15-40
initiated 16-12
initiation of 13-6
suspend-control 15-19
suspension of 13-8,15-31
termination (See conclusion of I/O operations)
sense data 15-36
(See also particular sense bit)
support functions 17-1
system reset 17-9
as part of subsystem reset 4-28
IAC (INSERT ADDRESS SPACE CONTROL) instruction 10-6
IC (INSERT CHARACTER) instruction 7-23
IC (instruction counter) (See instruction address)
ICM (INSERT CHARACTERS UNDER MASK) instruction 7-23
examples A-17
ID (See CPU identification and also sense ID)
IDA (indirect-data-address) flag (in CCW) 15-21
IDAW (indirect-data-address word) 15-30
check (in subchannel logout) 16-37
contents of 15-30
invalid address specification 16-24
invalid specification 16-25
idle subchannel 16-14
ILC (instruction-length code) 6-6
assigned storage locations for 3-39
for program interruptions 6-12
for supervisor-call interruption 6-31
IML (initial microprogram loading), controls 12-3
immediate operand 5-4
immediate operation 15-40
incorrect length (I/O-subchannel status) 16-24
for control command 15-36
for write command 15-34
incorrect state (signal-processor status) 4-35
index
for address generation 5-5
instructions for handling 7-13
into ASN first and second tables 3-14
into authority table 5-16
into entry and linkage tables 5-17
into measurement-block area (I/O) 17-6

LzLabs GmbH. Ex. 1016-553

register 2-3
indicator
check-stop 12-2
load 12-3
manual 12-3
mode 12-2
test 12-5
wait 12-5
indirect data address
flag (IDA flag in CCW) 15-21
role in I/O operations 15-30
word (See IDAW)
indirect data addressing (I/O) 15-30
information format 3-2
initial CPU reset 4-27
as signal-processor order 4-32
initial microprogram loading (IML), controls 12-3
initial program loading (See IPL)
initial-status-interruption control
in IRB 16-12
in ORB 15-19
used for IPL 17-13
inoperative (signal-processor status) 4-35
input/output (See I/O)
INSERT ADDRESS SPACE CONTROL (IAC) instruction 10-6
INSERT CHARACTER (IC) instruction 7-23
INSERT CHARACTERS UNDER MASK (ICM) instruction 7-23
examples A-17
INSERT PROGRAM MASK (IPM) instruction 7-23
INSERT PSW KEY (IPK) instruction 10-7
INSERT STORAGE KEY EXTENDED (ISKE) instruction 10-7
INSERT VIRTUAL STORAGE KEY (IVSK) instruction 10-7
installation 2-2
instruction address
handling by DAT 3-22
in address translation 3-5
in entry-table entry 5-19
in PSW 4-6
validity bit for 11-19
instruction-fetching-event mask 4-11
instructions
(See also by name and mnemonic)
backing up of 11-17
classes of 2-2
comparison with System/370 D-4
control 10-1
damage to 11-15,11-18
decimal 8-1
examples A-28
ending, types of 5-11
examples of use A-7
execution 5-6
fetching of 5-23
access exception for 6-26
PER event 4-14
floating-point 9-1
examples A-34
format 5-2
general 7-2
examples A-8
I/O, role in I/O operations 13-6
interruptible 5-11
length of 5-3
list of B-1
modification by EXECUTE instruction 7-22
prefetching of 5-23
privileged 4-5
for control 10-1
processing damage 11-15,11-18
semiprivileged 4-5,10-1
sequence of execution 5-2
stepping of (rate control) 12-4
effect on CPU state 4-2
effect on CPU timer 4-23
integer
binary 7-2
address as 5-5
examples A-2
decimal 8-2
integral boundary 3-3
interface
control check (I/O-subchannel status) 16-27
I/O, OEMI v
interlock of storage 5-22
for update references 5-26
intermediate interruption condition (I/O) 16-4
intermediate-status bit (I/O) 16-17
intermittent errors 11-5
internal storage 2-2
interrupt key 12-3
external interruption 6-10
subclass-mask bit 6-10
interruptible instructions 5-11
COMPARE LOGICAL LONG 7-19
MOVE LONG 7-28
PER event affecting the ending of 4-13
stopping of 4-2
TEST BLOCK 10-45
interruption 6-2
(See also masks)
action 6-2
I/O 16-6
machine-check 11-12
classes 6-5
code 6-5
assigned storage locations for 3-39
external 6-9
I/O 6-12,14-15,16-6
program 6-12
summary of 6-2
supervisor-call 6-31
conditions 6-2
alert (I/O) 16-5
clearing of 4-27
floating 6-6,11-21
I/O 13-8,16-2
intermediate (I/O) 16-4
primary (I/O) 16-4
secondary (I/O) 16-4
solicited (I/O) 16-3
unsolicited (I/O) 16-3
effect on instruction sequence 5-11
external 6-9
I/O (See I/O interruption)
identification, assigned storage locations for 3-39
machine-check
(See machine-check interruption)
code 11-14
masking of 6-5
parameter (I/O)
assigned storage locations for 3-40
in ORB 15-19
in PMCW 15-2
used for IPL 17-13

LzLabs GmbH. Ex. 1016-554

pending 6-5
external 6-9
machine-check 11-13
relation to CPU state 4-2
priority 6-32
access exceptions for 6-26
ASN-translation exceptions 6-30
external 6-9
I/O 16-5
PER event 4-13
program-interruption conditions 6-26
trace exceptions 6-31
program (See program interruption)
program-controlled (I/O) (See PCI)
response block (See IRB)
restart 6-31
string (See string of interruptions)
subclass code (I/O) 15-2
supervisor-call 6-31
suppress-suspended-interruption control (I/O)
in IRB 16-12
in ORB 15-19
intervention required (bit in I/O sense data) 15-36
invalid
address 6-13
ASN-first-table entry 3-15
ASN-second-table entry 3-15
CBC 11-2
in registers 11-9
in storage 11-6
in storage keys 11-7
linkage-table entry 5-18
operation code 6-18
order (signal-processor status) 4-35
page 3-25
parameter (signal-processor status) 4-35
segment 3-24
translation address 3-30
translation format 3-23
exception recognition 3-30
INVALIDATE PAGE TABLE ENTRY (IPTE) instruction 10-8
effect when CPU is stopped 4-2
IPK (INSERT PSW KEY) instruction 10-7
IPL (initial program loading) 4-29,17-12
assigned storage locations for 3-38
effect on CPU state 4-3
IPM (INSERT PROGRAM MASK) instruction 7-23
IPTE (INVALIDATE PAGE TABLE ENTRY) instruction 10-8
IRB (interruption-response block)
contents of 16-6
extended-control word (See ECW)
extended-status word (See ESW)
subchannel-status word (See SCSW)
ISKE (INSERT STORAGE KEY EXTENDED) instruction 10-7
IVSK (INSERT VIRTUAL STORAGE KEY) instruction 10-7

K
K (kilo), multiplier iv
key
access (See access key)
check (in subchannel logout) 16-36
manual (See manual operations)
mask
authorization 5-19
entry 5-19
PSW (PKM) 5-15
PSW (See PSW key)
storage (See storage key)
subchannel (See subchannel key)
key-controlled protection 3-9
exception for 6-20

L
L (LOAD) binary instruction 7-24
example A-18
L fields of instruction 5-4
LA (LOAD ADDRESS) instruction 7-24
examples A-18
LASP (LOAD ADDRESS SPACE PARAMETERS) instruction 10-10
last-path-used mask (See LPUM)
late exception recognition 6-8
LCDR (LOAD COMPLEMENT) floating-point instruction 9-11
LCER (LOAD COMPLEMENT) floating-point instruction 9-11
LCR (LOAD COMPLEMENT) binary instruction 7-24
LCTL (LOAD CONTROL) instruction 10-17
LD (LOAD) floating-point instruction 9-10
LDR (LOAD) floating-point instruction 9-10
LE (LOAD) floating-point instruction 9-10
left-to-right addressing 3-2
length
field 3-2
instruction 5-3
register operand 5-4
variable (storage operands) 5-4
LER (LOAD) floating-point instruction 9-10
LH (LOAD HALFWORD) instruction 7-25
examples A-19
limit mode 15-3
link information
for BRANCH AND LINK instruction 7-10
for BRANCH AND SAVE AND SET MODE instruction 7-11
for BRANCH AND SAVE instruction 7-10
linkage (subroutine) 5-7
linkage index (LX) 5-17
linkage table (LT) 5-18
designation (LTD) 5-18
in AST entry 3-16
length (LTL) 5-18
origin (LTO) 5-18
LM (LOAD MULTIPLE) instruction 7-25
LNDR (LOAD NEGATIVE) floating-point instruction 9-11
LNER (LOAD NEGATIVE) floating-point instruction 9-11
LNR (LOAD NEGATIVE) binary instruction 7-25
LOAD (L,LR) binary instructions 7-24
example A-18
LOAD (LD,LDR,LE,LER) floating-point instructions 9-10
LOAD ADDRESS (LA) instruction 7-24
examples A-18
LOAD ADDRESS SPACE PARAMETERS (LASP) instruction 10-10

LzLabs GmbH. Ex. 1016-555

LOAD AND TEST (LTDR,LTER) floating-point instructions 9-11
LOAD AND TEST (LTR) binary instruction 7-24
load-clear key 12-3
LOAD COMPLEMENT (LCDR,LCER) floating-point instructions 9-11
LOAD COMPLEMENT (LCR) binary instruction 7-24
LOAD CONTROL (LCTL) instruction 10-17
LOAD HALFWORD (LH) instruction 7-25
  examples A-19
load indicator 12-3
LOAD MULTIPLE (LM) instruction 7-25
LOAD NEGATIVE (LNDR,LNER) floating-point instructions 9-11
LOAD NEGATIVE (LNR) binary instruction 7-25
load-normal key 12-3
LOAD POSITIVE (LPDR,LPER) floating-point instructions 9-12
LOAD POSITIVE (LPR) binary instruction 7-26
LOAD PSW (LPSW) instruction 10-17
LOAD REAL ADDRESS (LRA) instruction 10-18
LOAD ROUNDED (LRDR,LRER) instructions 9-12
load state 4-2
  assigned storage while in 3-38
  in IPL 4-29
load-unit-address controls 12-3
loading, initial (See IPL, IML)
location not provided 6-13
logical
  address 3-5
    handling by DAT 3-22
  arithmetic (See unsigned binary arithmetic)
  comparison 7-4
  connective
    AND 7-9
    EXCLUSIVE OR 7-21
    OR 7-33
  data 7-2
logical-path mask (See LPM)
logout
  fixed
    assigned storage locations for 3-41
    machine-check 11-22
  subchannel (I/O) 16-36
long floating-point number 9-2
long I/O block 16-24
loop control 5-7
loop of interruptions (See string of interruptions)
low-address protection 3-11
  control bit 3-11
  exception for 6-20
LPDR (LOAD POSITIVE) floating-point instruction 9-12
LPER (LOAD POSITIVE) floating-point instruction 9-12
LPM (logical-path mask) 15-4
  effect on system performance 15-8
  in ORB 15-20
  used for IPL 17-13
LPR (LOAD POSITIVE) binary instruction 7-26
LPSW (LOAD PSW) instruction 10-17
LPUM (last-path-used mask) 15-4
  in ESW 16-37
LR (LOAD) binary instruction 7-24
LRA (LOAD REAL ADDRESS) instruction 10-18
LRDR (LOAD ROUNDED) instruction 9-12
LRER (LOAD ROUNDED) instruction 9-12
LT (linkage table) 5-18
LTD (linkage-table designation) 5-18
LTDR (LOAD AND TEST) floating-point instruction 9-11
LTER (LOAD AND TEST) floating-point instruction 9-11
LTL (linkage-table length) 5-18
LTO (linkage-table origin) 5-18
LTR (LOAD AND TEST) binary instruction 7-24
LX (linkage index) 5-17
  invalid bit 5-18
  translation exception 6-17

M
M (mega), multiplier iv
M (MULTIPLY) binary instruction 7-32
  example A-22
machine check 11-2
  (See also malfunction)
  comparison with System/370 D-9
  fixed logout 11-22
  interruption 6-12,11-11
    action 11-12
    code (MCIC) 3-40,11-14
    floating conditions 11-21
  logout 11-22
  mask in PSW 4-5
  subclass masks 11-21
main storage 3-2
  (See also storage)
  power-on reset 4-29
  shared, in multiprocessing 4-30
malfunction 11-2
  (See also machine check)
  correction of 11-2
  effect of DIAGNOSE instruction 10-5
  effect on manual operation 12-1
  indication of 11-4
malfunction alert (external interruption) 6-11
  subclass-mask bit 6-11
  when entering check-stop state 11-10
manual indicator 12-3
  (See also stopped state)
manual operations 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 4-33
  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

LzLabs GmbH. Ex. 1016-556

masks 6-5
(See also interruption)
in BRANCH ON CONDITION instruction 7-12
in COMPARE LOGICAL CHARACTERS UNDER MASK instruction 7-17
in INSERT CHARACTERS UNDER MASK instruction 7-23
in PSW 4-5
in STORE CHARACTERS UNDER MASK instruction 7-38
monitor 6-18
PER event 4-11
PER general-register 4-11
program-interruption 6-13
subclass
external 6-9
I/O 6-12
machine check 11-21
maximum negative number 7-2
MC (MONITOR CALL) instruction 7-26
MCIC (machine-check-interruption code) 11-14
MD (MULTIPLY) floating-point instruction 9-13
MDR (MULTIPLY) floating-point instruction 9-13
example A-36
ME (MULTIPLY) floating-point instruction 9-13
measurement
block (I/O) 17-4
data check 16-36
index 15-5,15-8,17-6
key 17-5
key, used as access key 3-10
origin 17-5
program check 16-36
protection check 16-36
update enable 15-3,17-6
update facility 17-4
update mode 17-6
device-connect-time-measurement facility (I/O) 17-7
mode control 15-3
parameters (I/O)
accrued 17-4
effect of CSCH 14-4
effect of HSCH 14-5
MER (MULTIPLY) floating-point instruction 9-13
message byte 8-6
MH (MULTIPLY HALFWORD) instruction 7-32
example A-23
microprogram, initial loading of 12-3
mode
addressing (See addressing mode)
architectural (See architectural mode)
burst (channel-path operation) 13-3
byte-multiplex (channel-path operation) 13-3
device-connect-time measurement (I/O) 17-7
indicator 12-2
measurement block update (I/O) 17-6
multipath (See multipath mode)
primary-space 3-22
real 3-22
requirements for semiprivileged instructions 5-15
secondary-space 3-22
single-path (See single-path mode)
System/370 (See System/370 mode)
model, CPU 10-43
MODIFY SUBCHANNEL (MSCH) instruction 14-6
MONITOR CALL (MC) instruction 7-26
monitor-class number 6-18
assigned storage locations for 3-40
monitor code 6-18
assigned storage locations for 3-40
monitor event 6-18
monitor masks 6-18
monitoring
(See also measurement)
channel-subsystem facilities 17-2
for PER events (See PER)
with MONITOR CALL 6-18,7-26
MOVE (MVC,MVI) instructions 7-27
examples A-17,A-19
MOVE LONG (MVCL) instruction 7-27
examples A-20
MOVE NUMERICS (MVN) instruction 7-30
example A-21
MOVE TO PRIMARY (MVCP) instruction 10-19
MOVE TO SECONDARY (MVCS) instruction 10-19
MOVE WITH KEY (MVCK) instruction 10-20
MOVE WITH OFFSET (MVO) instruction 7-31
example A-21
MOVE ZONES (MVZ) instruction 7-31
example A-22
MP (MULTIPLY DECIMAL) instruction 8-10
example A-32
MR (MULTIPLY) binary instruction 7-32
example A-22
MSCH (MODIFY SUBCHANNEL) instruction 14-6
multipath mode 15-4
entering 15-18
multiple-access reference 5-27
MULTIPLY (M,MR) binary instructions 7-32
examples A-22
MULTIPLY (MD,MDR,ME,MER,MXD,MXDR,MXR) floating-point instructions 9-13
example A-36
MULTIPLY DECIMAL (MP) instruction 8-10
example A-32
MULTIPLY HALFWORD (MH) instruction 7-32
example A-23
multiprocessing 4-30
manual operations for 12-5
programming considerations for A-38,8-3
timing-facility interruptions for 4-21
TOD clock for 4-18
multiprogramming examples A-38
MVC (MOVE) instruction 7-27
examples A-17,A-19
MVCK (MOVE WITH KEY) instruction 10-20
MVCL (MOVE LONG) instruction 7-27
examples A-20
MVCP (MOVE TO PRIMARY) instruction 10-19
MVCS (MOVE TO SECONDARY) instruction 10-19
MVI (MOVE) instruction 7-27
example A-20
MVN (MOVE NUMERICS) instruction 7-30
example A-21
MVO (MOVE WITH OFFSET) instruction 7-31
example A-21
MVS/XA assists v
MVZ (MOVE ZONES) instruction 7-31

LzLabs GmbH. Ex. 1016-557

example A-22
MXD (MULTIPLY) floating-point instruction 9-13
MXDR (MULTIPLY) floating-point instruction 9-13
MXR (MULTIPLY) floating-point instruction 9-13

N
N (AND) instruction 7-9
example A-8
NC (AND) instruction 7-9
example A-8
near-valid CBC 11-2
in storage 11-5
negative zero
binary 7-2
decimal 8-3
example A-5
new PSW 4-3
assigned storage locations for 3-38
fetched during interruption 6-2
NI (AND) instruction 7-9
example A-8
no-operation
as an I/O command (control) 15-35
instruction (BRANCH ON CONDITION) 7-12
nonvolatile storage 3-2
normalization 9-2
not operational
as CPU state 4-33
as TOD-clock state 4-19
not set (TOD-clock state) 4-19
NR (AND) instruction 7-9
example A-8
nullification of instruction execution 5-11
exceptions to 5-12
for exigent machine-check conditions 11-11
numbering
addresses (byte locations) 3-2
bits 3-2
numbers
binary 7-2
examples A-2
CPU-model 10-43
decimal 8-1
examples A-5
floating-point 9-1
examples A-5
numeric bits 8-1
moving of 7-30

O
O (OR) instruction 7-33
OC (OR) instruction 7-33
OEMI (original equipment manufacturers' information), I/O interface v
OI (OR) instruction 7-33
example A-23
of problem with OR immediate A-38
old PSW 6-2
assigned storage locations for 3-38
one's complement binary notation 7-2
used for SUBTRACT LOGICAL instruction 7-41
op code (See operation code)
operand 5-2
address generation for 5-5
exception 6-18
immediate 5-4
length 5-2
overlap 7-2
decimal 8-3
register 5-4
sequence of references for 5-25
storage 5-4
types (fetch, store, and update) 5-25
used for result 5-2
operating state 4-2
operation
code (op code) 5-2
invalid 6-18
exception 6-18
unit of 5-11
operation-request block (See ORB)
operator facilities 2-5,12-1
basic 12-1
operator intervening (signal-processor status) 4-35
OR (O,OC,OI,OR) instructions 7-33
example of problem with OR immediate A-38
examples A-23
ORB (operation-request block)
channel-program address 15-20
contents 15-19
interruption parameter 15-19
invalid 16-25
logical-path mask (LPM) 15-20
orders (signal-processor) 4-31
conditions precluding response 4-33
CPU reset 4-32
emergency signal 4-31
external call 4-31
initial CPU reset 4-32
restart 4-31
sense 4-31
set prefix 4-32
start 4-31
stop 4-31
stop and store status 4-31
store status at address 4-32
orders in I/O command code 13-6,15-22
overflow
binary 7-3
example A-2
decimal 6-16
exponent (See exponent overflow)
fixed-point 6-17,7-3
in CRW 17-17
overlap
destructive 7-28
operand 7-2
decimal 8-3
operation 5-21
overrun (bit in I/O sense data) 15-37

P
PACK (PACK) instruction 7-33
example A-23
packed decimal numbers 8-1
conversion from zoned format 7-33
conversion to zoned format 7-44
examples A-5
padding byte
for COMPARE LOGICAL LONG instruction 7-17
for MOVE LONG instruction 7-27

LzLabs GmbH. Ex. 1016-558

page 3-22
index (PX) 3-22
invalid bit 3-25
protection D-2,3-10
bit for 3-25
exception for 6-20
swapping 3-21
table 3-25
designation 3-24
length (PTL) 3-25
lookup 3-29
origin (PTO) 3-24
translation exception 6-19
as an access exception 6-24
page-frame real address (PFRA) 3-25
PAM (path-available mask) 15-5
effect of reconfiguration on 15-8
effect of resetting 15-8
effect on allegiance 15-9
parameter register for SIGNAL PROCESSOR 4-32,10-41
parameters, translation 3-22
parity bit 11-2
partial completion of instruction execution 5-12
PASN (primary address-space number) 3-14
in trace entry 4-10
path
available mask (See PAM)
installed mask (See PIM)
management 13-6
control word (See PMCW)
for CLEAR SUBCHANNEL 15-11
for HALT SUBCHANNEL 15-13
for RESUME SUBCHANNEL 15-15
for START SUBCHANNEL 15-15
multipath mode (bit in PMCW) 15-4
not operational mask (See PNOM)
operational mask (See POM)
pattern for editing 8-6
PC (PROGRAM CALL) instruction 10-22
PC-cp (PROGRAM CALL instruction, to current primary) 10-23
PC number
in trace entry 4-10
translation 5-17
PC-ss (PROGRAM CALL instruction, with space switching) 10-23
PC-translation-specification exception 6-19
PCI (program-controlled interruption)
description 15-29
flag in CCW 15-21
I/O-subchannel status 16-24
pending channel reports, effect of I/O-system reset 17-11
pending interruption (See interruption, pending)
PER (program-event recording) 4-11
address 4-12
assigned storage locations for 3-40
wraparound 4-14
code 4-12
assigned storage locations for 3-40
ending address 4-12
events 4-11
general-register-alteration 4-15
instruction-fetching 4-14
masks 4-11
priority of interruptions 4-13
program-interruption condition 6-19
storage alteration 4-15
storage-area designation 4-14
successful branching 4-14
general-register-mask bits 4-11
mask (in PSW) 4-5
subclass masks 4-11
starting address 4-12
PFRA (page-frame real address) 3-25
PIM (path-installed mask) 15-5
PKM (PSW-key mask) 5-15
PMCW (path-management-control word) 15-2
channel-path identifiers (CHPID) 15-5
PNOM (path-not-operational mask) 15-4
effect of POM on 15-9
point of damage 11-14
point of interruption 5-11
for machine check 11-13
POM (path-operational mask) 15-5
effect on PNOM 15-9
postnormalization 9-2
power controls 12-3
power-on reset 4-28
precision (floating-point) 9-1
preferred sign codes 8-2
prefetching
for I/O 15-23
of DAT-table entries 5-24
of instructions 5-23
prefix 3-12
set by signal-processor order 4-32
store-status save area for 3-41
prenormalization 9-2
primary address space 3-13
primary ASN (PASN) 3-14
primary authority 3-19
exception 6-19
primary interruption condition (I/O) 16-4
primary segment-table designation (PSTD) 3-23
primary segment-table length (PSTL) 3-23
primary segment-table origin (PSTO) 3-23
primary-space mode 3-22
primary-status bit (I/O) 16-18
primary virtual address 3-5
effective segment-table designation for 3-26
priority (See interruption)
of interruptions 6-32
of program-interruption conditions 6-26
privileged instructions 4-5
for control 10-1
for I/O 14-1
privileged-operation exception 6-20
problem state 4-5
bit in entry-table entry 5-19
processing backup 11-17
processing damage 11-18
processor (See CPU)
program
check
I/O-subchannel status 16-24
measurement block 16-36
controlled-interruption (I/O) (See PCI)
event recording (See PER)
events (See PER events)

LzLabs GmbH. Ex. 1016-559

exceptions 6-12
execution 5-2
initial loading of 4-29
interruption 6-12
priority 6-26
mask (in PSW) 4-6
validity bit for 11-19
modifiable fields of SCHIB 15-6
status word (See PSW)
PROGRAM CALL (PC) instruction 10-22
trace entry for 4-10
PROGRAM TRANSFER (PT) instruction 10-28
trace entry for 4-10
protection
check
I/O-subchannel status 16-25
measurement block 16-36
exception 6-20
as an access exception 6-24
of storage (See storage protection)
PSTD (primary segment-table designation) 3-23
PSTL (primary segment-table length) 3-23
PSTO (primary segment-table origin) 3-23
PSW (program-status word) 2-3,4-3
assigned storage locations for 3-38
comparison with System/370 D-7
current 4-3,5-6
exceptions associated with 6-8
format error 6-8
in IPL, assigned storage locations 3-38
in program execution 5-6
validity bits for 11-19
PSW key 4-5
in trace entry 4-10
mask (PKM) 5-15
used as access key 3-10
validity bit for 11-19
PT (PROGRAM TRANSFER) instruction 10-28
PT-cp (PROGRAM TRANSFER instruction, to current primary) 10-28
PT-ss (PROGRAM TRANSFER instruction, with space switching) 10-28
PTL (page-table length) 3-25
PTLB (PURGE TLB) instruction 10-33
PTO (page-table origin) 3-24
PURGE TLB (PTLB) instruction 10-33
PX (page index) 3-22

R
R field of instruction 5-4
range, floating-point 9-1
rate control 12-4
RCHP (See RESET CHANNEL PATH instruction)
read (I/O command) 15-34
read backward (I/O command) 15-35
real address 3-4
real mode 3-22
real storage 3-5
receiver check (signal-processor status) 4-35
reconfiguration of the I/O system 17-14
recovery
condition 11-11
system 11-16
recovery subclass-mask bit 11-22
redundancy 11-2
reference
bit in storage key 3-8
multiple access 5-27
recording 3-11
sequence for storage 5-21
DAT-table entries 5-24
instructions 5-23
operands 5-25
storage keys 5-24
single-access 5-27
register
base-address 2-3
control 2-3
designation of 5-4
floating-point 2-3
general 2-3
index 2-3
prefix 3-12
save areas 3-40,11-20
validation 11-9
validity bits for 11-19
remote operating stations 12-1
reporting-source code (See RSC)
reporting-source ID (See RSID)
repressible machine-check condition 11-11
RESET CHANNEL PATH (RCHP) instruction 14-6
channel-path reset 17-9
function initiated by 15-42
reset signal 17-8
RESET REFERENCE BIT EXTENDED (RRBE) instruction 10-33
reset signal
definition 17-8
in channel-path reset 17-9
in I/O-system reset 17-9,17-10
issued as part of RCHP 15-42
resets 4-24,17-9
channel-path 17-9
effect on CPU state 4-2
effect on TOD clock 4-18
I/O system 17-9
resolution
of clock comparator 4-22
of CPU timer 4-22
of TOD clock 4-18
restart
as signal-processor order 4-31
interruption 6-31
key 12-4
result operand 5-2
resume function
execution of 15-15
initiated by RESUME SUBCHANNEL 14-8
pending 16-14
RESUME SUBCHANNEL (RSCH) instruction 14-8
channel-program requirements 14-8
count in measurement block 17-4
function initiated by 15-15
path management for 15-15
retry
CPU 11-2
I/O command (See command retry)
rounding (decimal) 8-10
example A-33
RR instruction format 5-3
RRBE (RESET REFERENCE BIT EXTENDED) instruction 10-33
RRE instruction format 5-3
RS instruction format 5-3
RSC (reporting-source code) 17-17
RSCH (See RESUME SUBCHANNEL instruction)

LzLabs GmbH. Ex. 1016-560

RSID (reporting-source ID) 17-18
running (of TOD clock) 4-19
RX instruction format 5-3

S
S (SUBTRACT) binary instruction 7-40
S instruction format 5-3
SAC (SET ADDRESS SPACE CONTROL) instruction 10-33
SAL (SET ADDRESS LIMIT) instruction 14-9
sample count (in ESW) 17-4
SASN (secondary address-space number) 3-14
  in trace entry 4-10
save areas for registers 3-40,11-20
SCHIB (subchannel-information block)
  contents 15-1
  model-dependent area 15-6
  path-management-control word (PMCW) 15-2
  subchannel-status word (SCSW) 15-6
  summary of modifiable fields 15-6
SCHM (See SET CHANNEL MONITOR instruction)
SCK (SET CLOCK) instruction 10-34
SCKC (SET CLOCK COMPARATOR) instruction 10-35
SCSW (subchannel-status word) 16-7
  activity-control field 16-14
  CCW address 16-28
  contents of 16-8
  count 16-33
  device-status field 16-18
  function-control field 16-13
  in SCHIB 15-6
  status-control field 16-16
  subchannel-control fields 16-12
  subchannel-status field 16-23
SD (SUBTRACT NORMALIZED) instruction 9-14
SDR (SUBTRACT NORMALIZED) instruction 9-14
SE (SUBTRACT NORMALIZED) instruction 9-14
secondary address space 3-13
secondary ASN (SASN) 3-14
secondary authority 3-19
  exception 6-21
secondary error (in subchannel logout) 16-38
secondary interruption condition (I/O) 16-4
secondary segment-table designation (SSTD) 3-24
secondary segment-table length (SSTL) 3-24
secondary segment-table origin (SSTO) 3-24
secondary-space-control bit 3-23,5-15
secondary-space mode 3-22
secondary-status bit (I/O) 16-18
secondary virtual address 3-5
  effective segment-table designation for 3-26
segment 3-22
  index (SX) 3-22
  invalid bit 3-24
  table 3-24
    lookup 3-29
  translation exception 6-21
    as an access exception 6-24
segment-table designation (STD) 3-23
  effective 3-26
  primary 3-23
  secondary 3-24
self-describing block of I/O data 15-27
semiprivileged, instructions 4-5,10-1
semiprivileged programs, authorization of 5-14
sense
  as an I/O command 15-36
  as signal-processor order 4-31
sense data (I/O) 15-36
  indication 16-22
sense ID (I/O command) 15-38
sequence
  code (in subchannel logout) 16-38
  conceptual 5-21
  instruction-execution 5-2
  of storage references 5-21
SER (SUBTRACT NORMALIZED) instruction 9-14
serialization 5-29
  channel-program 5-30
  completion of store operations 5-25
  CPU 5-29
service processor 2-5,4-23
  damage 11-17
service signal
  called by DIAGNOSE 4-23
  external interruption 6-11
  subclass-mask bit 6-11
SET ADDRESS LIMIT (SAL) instruction 14-9
SET ADDRESS SPACE CONTROL (SAC) instruction 10-33
SET CHANNEL MONITOR (SCHM) instruction 14-10
  effect on measurement modes 17-2
SET CLOCK (SCK) instruction 10-34
SET CLOCK COMPARATOR (SCKC) instruction 10-35
SET CPU TIMER (SPT) instruction 10-35
set prefix (signal-processor order) 4-32
SET PREFIX (SPX) instruction 10-36
SET PROGRAM MASK (SPM) instruction 7-34
SET PSW KEY FROM ADDRESS (SPKA) instruction 10-36
SET SECONDARY ASN (SSAR) instruction 10-37
set state (TOD clock) 4-19
SET STORAGE KEY EXTENDED (SSKE) instruction 10-40
SET SYSTEM MASK (SSM) instruction 10-40
SH (SUBTRACT HALFWORD) instruction 7-40
shared storage (See storage, shared)
shared TOD clock 4-18
SHIFT AND ROUND DECIMAL (SRP) instruction 8-10
  examples A-32
SHIFT LEFT DOUBLE (SLDA) instruction 7-34
  example A-24
SHIFT LEFT DOUBLE LOGICAL (SLDL) instruction 7-35
SHIFT LEFT SINGLE (SLA) instruction 7-35
  example A-24
SHIFT LEFT SINGLE LOGICAL (SLL) instruction 7-36
SHIFT RIGHT DOUBLE (SRDA) instruction 7-36
SHIFT RIGHT DOUBLE LOGICAL (SRDL) instruction 7-36

LzLabs GmbH. Ex. 1016-561

SHIFT RIGHT SINGLE (SRA) instruction 7-37
SHIFT RIGHT SINGLE LOGICAL (SRL) instruction 7-37
shifting, floating-point (See normalization)
short floating-point number 9-2
short I/O block 16-24
SI instruction format 5-3
SID (subsystem-identification word) 14-1
  assigned storage locations for 3-40
sign bit
  binary 7-2
  floating-point 9-1
sign codes (decimal) 8-2
SIGNAL PROCESSOR (SIGP) instruction 10-41
  order codes 4-31
SIGNAL PROCESSOR instruction, comparison with System/370 D-9
signals (I/O) 17-8
  clear signal (See clear signal)
  halt signal (See halt signal)
  reset signal (See reset signal)
signed binary
  arithmetic 7-3
  comparison 7-4
  integer 7-2
    examples A-2
significance
  exception 6-21
  loss 9-2,9-7
  mask in PSW 4-6
  starter 8-6
SIGP (SIGNAL PROCESSOR) instruction 10-41
single-access reference 5-27
single-path mode 15-4,15-17
size
  notation for iv
  of address 3-6
    controlled by addressing mode 5-4
skip flag
  effect on data transfer 15-28
  in CCW 15-21
skipping data transfer (I/O) 15-28
SL (SUBTRACT LOGICAL) instruction 7-40
SLA (SHIFT LEFT SINGLE) instruction 7-35
  example A-24
SLDA (SHIFT LEFT DOUBLE) instruction 7-34
  example A-24
SLDL (SHIFT LEFT DOUBLE LOGICAL) instruction 7-35
SLL (SHIFT LEFT SINGLE LOGICAL) instruction 7-36
SLR (SUBTRACT LOGICAL) instruction 7-40
solicited interruption condition (I/O) 16-3
solid errors 11-5
source, of interruption 6-5
SP (SUBTRACT DECIMAL) instruction 8-11
space-switch event
  control bit 6-21
    after ASN translation 3-16
    in translation control 3-23
  program-interruption condition 6-21
special-operation exception 6-22
specification exception 6-22
SPKA (SET PSW KEY FROM ADDRESS) instruction 10-36
SPM (SET PROGRAM MASK) instruction 7-34
SPT (SET CPU TIMER) instruction 10-35
SPX (SET PREFIX) instruction 10-36
SR (SUBTRACT) binary instruction 7-40
SRA (SHIFT RIGHT SINGLE) instruction 7-37
SRDA (SHIFT RIGHT DOUBLE) instruction 7-36
SRDL (SHIFT RIGHT DOUBLE LOGICAL) instruction 7-36
SRL (SHIFT RIGHT SINGLE LOGICAL) instruction 7-37
SRP (SHIFT AND ROUND DECIMAL) instruction 8-10
  examples A-32
SS instruction format 5-3
SSAR (SET SECONDARY ASN) instruction 10-37
SSAR-cp (SET SECONDARY ASN instruction, to current primary) 10-37
SSAR-ss (SET SECONDARY ASN instruction, with space switching) 10-37
SSCH (See START SUBCHANNEL instruction)
SSE instruction format 5-3
SSKE (SET STORAGE KEY EXTENDED) instruction 10-40
SSM (SET SYSTEM MASK) instruction 10-40
SSM-suppression-control bit 6-22,10-40
SSTD (secondary segment-table designation) 3-24
SSTL (secondary segment-table length) 3-24
SSTO (secondary segment-table origin) 3-24
ST (STORE) binary instruction 7-37
standalone dump 12-4
standard epoch (for TOD clock) 4-20
STAP (STORE CPU ADDRESS) instruction 10-43
start
  as signal-processor order 4-31
  function 4-2
  key 12-4
start function (I/O)
  execution of 15-15
  indication of 16-13
  initiated by START SUBCHANNEL 14-11
  initiation 13-6
  overview 13-6
  pending 16-15
START SUBCHANNEL (SSCH) instruction 14-11
  count in measurement block 17-4
  deferred condition code (IRB) 16-9
  function initiated by 15-15
  operation-request block (ORB) 15-19
  path management for 15-15
state
  CPU (See CPU state)
  TOD clock 4-19
status
  alert 16-16
  control field in SCSW 16-16
  device 16-18
    attention 16-19
    busy 16-21
    channel end 16-21
    control-unit end 16-19
    device end 16-21
    effect of CSCH 15-12
    status modifier 16-19
    unit check 16-22
    unit exception 16-23
    while disabled 14-6
  intermediate 16-17

LzLabs GmbH. Ex. 1016-562

modifier
effect of in command chaining 15-28
I/O-device status 16-19
pending 16-18
primary 16-18
program (See PSW)
register for SIGP 4-31,10-41
resulting from signal-processor orders 4-34
secondary 16-18
storing of 4-30
manual key for 12-4
subchannel 16-23
chaining check 16-27
channel-control check 16-26
channel-data check 16-26
incorrect length 16-24
interface-control check 16-27
program check 16-24
program-controlled interruption (PCI) 16-24
protection check 16-25
while disabled 14-6
STC (STORE CHARACTER) instruction 7-38
STCK (STORE CLOCK) instruction 7-38
STCKC (STORE CLOCK COMPARATOR) instruction 10-42
STCM (STORE CHARACTERS UNDER MASK) instruction 7-38
examples A-24
STCPS (STORE CHANNEL PATH STATUS) instruction 14-13
STCRW (See STORE CHANNEL REPORT WORD instruction)
STCTL (STORE CONTROL) instruction 10-42
STD (segment-table designation) 3-23
STD (STORE) floating-point instruction 9-14
STE (STORE) floating-point instruction 9-14
STH (STORE HALFWORD) instruction 7-39
STIDP (STORE CPU ID) instruction 10-43
STL (segment-table length) 3-23
STM (STORE MULTIPLE) instruction 7-39
example A-25
STNSM (STORE THEN AND SYSTEM MASK) instruction 10-44
STO (segment-table origin) 3-23
stop
as signal-processor order 4-31
function 4-2
key 12-4
stop and store status (signal-processor order) 4-31
stopped bit (in signal-processor status) 4-35
stopped state
of CPU 4-2
effect on completion of store operations 5-25
of TOD clock 4-19
storage 3-2
absolute 3-4
access code (in subchannel logout) 16-37
address wraparound (See wraparound)
addressing 3-2
(See also address)
alteration
event mask 4-11
manual control for 12-2
PER event 4-15
area designation
for I/O operations 15-22
for PER events 4-14
assigned locations in 3-38
comparison with System/370 D-9
auxiliary 3-2,3-21
block 3-4
testing for usability 10-45
buffer (cache) 3-2
clearing (See clearing operation)
configuration of 3-4
direct-access 3-2
display 12-2
error 11-18
failing address (See failing-storage address)
interlocks 5-22
internal 2-2
key 3-8
error 11-18
sequence of references to 5-24
testing for usability 10-45
validation of 11-7
logical validity bit for 11-20
main 3-2
nonvolatile 3-2
operand 5-4
consistency 5-27
consistency example A-42,A-45
fetch reference 5-25
store reference 5-25
update reference 5-26
prefixing for 3-12
protection 3-8
during tracing 4-10
real 3-5
sequence of references 5-21
shared
by address spaces 3-21
by the channel subsystem and CPUs 3-4
examples A-38
in multiprocessing 4-30
size, notation for iv
validation 11-6
virtual 3-21
volatile 3-2
effect of power-on reset 4-29
STORE (ST) binary instruction 7-37
STORE (STD,STE) floating-point instructions 9-14
STORE CHANNEL PATH STATUS (STCPS) instruction 14-13
STORE CHANNEL REPORT WORD (STCRW) instruction 14-13
channel-report word (CRW) 17-15
STORE CHARACTER (STC) instruction 7-38
STORE CHARACTERS UNDER MASK (STCM) instruction 7-38
examples A-24
STORE CLOCK (STCK) instruction 7-38
STORE CLOCK COMPARATOR (STCKC) instruction 10-42
STORE CONTROL (STCTL) instruction 10-42
STORE CPU ADDRESS (STAP) instruction 10-43
STORE CPU ID (STIDP) instruction 10-43
STORE CPU TIMER (STPT) instruction 10-43
STORE HALFWORD (STH) instruction 7-39
STORE MULTIPLE (STM) instruction 7-39
example A-25
STORE PREFIX (STPX) instruction 10-44
store reference 5-25
access exceptions for 6-26

LzLabs GmbH. Ex. 1016-563

store status 4-29
as signal-processor order 4-31
key 12-4
store status at address (signal-processor order) 4-32
STORE SUBCHANNEL (STSCH) instruction 14-14
STORE THEN AND SYSTEM MASK (STNSM) instruction 10-44
STORE THEN OR SYSTEM MASK (STOSM) instruction 10-44
STOSM (STORE THEN OR SYSTEM MASK) instruction 10-44
STPT (STORE CPU TIMER) instruction 10-43
STPX (STORE PREFIX) instruction 10-44
string of interruptions 4-3,6-32
by clock comparator 4-22
by CPU timer 4-23
STSCH (STORE SUBCHANNEL) instruction 14-14
SU (SUBTRACT UNNORMALIZED) instruction 9-15
subchannel 13-2
active 16-15
active allegiance 15-9
addressing 13-5
control information in SCSW 16-12
dedicated allegiance 15-10
effect of I/O-system reset 17-10
enabled bit in PMCW 15-2
idle 16-14
information block (See SCHIB)
key
check (in subchannel logout) 16-36
in IRB 16-8
in ORB 15-19
used as access key 3-10
used for IPL 17-13
logout 16-36
number 13-5
status (See status, subchannel)
status word (See SCSW)
working allegiance 15-9
subclass-mask bits 6-5
external-interruption 6-9
machine-check 11-21
subroutine linkage 5-7
subsystem-identification word (See SID)
subsystem-linkage-control bit 5-15,5-18
subsystem reset 4-28
SUBTRACT (S,SR) binary instructions 7-40
SUBTRACT DECIMAL (SP) instruction 8-11
SUBTRACT HALFWORD (SH) instruction 7-40
SUBTRACT LOGICAL (SL,SLR) instructions 7-40
SUBTRACT NORMALIZED (SD,SDR,SE,SER,SXR) instructions 9-14
SUBTRACT UNNORMALIZED (SU,SUR,SW,SWR) instructions 9-15
successful branching (PER event) 4-14
successful-branching-event mask 4-11
SUPERVISOR CALL (SVC) instruction 7-41
supervisor-call interruption 6-31
supervisor state 4-5
support functions (I/O) 17-1
suppress-length-indication (SLI) flag in CCW 15-33
suppression
of instruction execution 5-11
exceptions to 5-12
of unit of operation 5-12
SUR (SUBTRACT UNNORMALIZED) instruction 9-15
suspend
control bit
in IRB 16-8
in ORB 15-19
used for IPL 17-13
flag in CCW 15-21
invalid 16-25
suppress-suspended-interruption control (I/O)
in IRB 16-12
in ORB 15-19
used for IPL 17-13
suspend and resume 13-8,15-31
suspended bit (in IRB) 16-16
suspension of channel-program execution 15-19,15-21,15-31
effect on DCTI 15-32
SVC (SUPERVISOR CALL) instruction 7-41
SW (SUBTRACT UNNORMALIZED) instruction 9-15
swapping
by COMPARE (DOUBLE) AND SWAP instructions 7-14
by EXCLUSIVE OR instruction 7-21
SWR (SUBTRACT UNNORMALIZED) instruction 9-15
SX (segment index) 3-22
SXR (SUBTRACT NORMALIZED) instruction 9-14
synchronization
checkpoint 11-3
CPU timer with TOD clock 4-22
of TOD clocks 4-19,4-21
synchronous machine-check interruption conditions 11-17
system
check stop for 11-11
damage 11-15
manual control of 12-1
mask (in PSW) 4-3
validity bit for 11-19
organization 2-1
recovery 11-16
reset (See resets)
clear key 12-5
I/O (See I/O system reset)
normal key 12-5
System/370 mode
comparison with 370-XA mode D-1
selection
by architectural-mode-selection controls 12-2
by IML controls 12-3
for compatibility 1-4

T
target instruction (See EXECUTE instruction)
TB (TEST BLOCK) instruction 10-45
termination
code (in subchannel logout) 16-38
of I/O operations (See conclusion of I/O operations)
of instruction execution 5-11
for exigent machine-check conditions 11-11
of unit of operation 5-12
TEST AND SET (TS) instruction 7-41
TEST BLOCK (TB) instruction 10-45
test indicator 12-5

LzLabs GmbH. Ex. 1016-564

TEST PENDING INTERRUPTION (TPI) instruction 14-15
  interruption code stored 16-6
TEST PROTECTION (TPROT) instruction 10-47
TEST SUBCHANNEL (TSCH) instruction 14-16
  interruption-response block (IRB) 16-6
TEST UNDER MASK (TM) instruction 7-42
  examples A-25
testing for storage-block and storage-key usability 10-45
TIC (I/O command) (See transfer in channel)
time-of-day clock (See TOD clock)
timer, CPU (See CPU timer)
timing facilities 4-18
  bit in PMCW 15-4
  channel-subsystem 17-2
  damage 11-16
    for TOD clock 4-19
timing-facility damage, subclass-mask bit 11-22
TLB (translation-lookaside buffer) 3-30
  entry
    clearing of 3-34
    effect of translation changes 3-34
    state 3-31
TM (TEST UNDER MASK) instruction 7-42
  examples A-25
TOD clock 4-18
  effect of power-on reset 4-29
  effect on clock-comparator interruption 6-10
  effect on CPU-timer decrementing 4-23
  effect on CPU-timer interruption 6-10
  manual control 4-19,12-5
  sync check (external interruption) 6-11
  sync-check subclass-mask bit 6-11
  unique values 4-20
  validation 11-9
  value in trace entry 4-10
TOD-clock-sync-control bit 4-19,4-21
TOD-clock-synchronization facility 4-21
TPI (See TEST PENDING INTERRUPTION instruction)
TPROT (TEST PROTECTION) instruction 10-47
TR (TRANSLATE) instruction 7-42
  example A-26
trace D-2,4-8
TRACE (TRACE) instruction 10-48
  trace entry for 4-10
trace
  entries 4-9
  entry address 4-8
  exceptions 6-31
trace-table exception 6-23
transfer in channel (I/O command) 15-39
  invalid sequence 16-25
TRANSLATE (TR) instruction 7-42
  example A-26
TRANSLATE AND TEST (TRT) instruction 7-43
  example A-26
translation
  address 3-21
  ASN 3-13
  exception identification 3-40
  format 3-23
  lookaside buffer (See TLB)
  parameters 3-22
  PC number 5-17
  specification exception 6-23
  tables for 3-24
trial execution 5-14
  PER events during 4-12
TRT (TRANSLATE AND TEST) instruction 7-43
  example A-26
true zero 9-1
TS (TEST AND SET) instruction 7-41
TSCH (See TEST SUBCHANNEL instruction)
two's complement binary notation 7-2
  examples A-2

U
underflow (See exponent underflow)
unit check (I/O-device status) 16-22
  dedicated allegiance 15-10
unit exception (I/O-device status) 16-23
unit of operation 5-11
unnormalized floating-point number 9-2
UNPACK (UNPK) instruction 7-44
  example A-28
UNPK (UNPACK) instruction 7-44
  example A-28
unsigned binary
  arithmetic 7-3
  integer 7-2
    examples A-4
    in address generation 5-5
unsolicited interruption condition (I/O) 16-3
update reference 5-26
usable TLB entry 3-31

V
valid CBC 11-2
valid segment-table or page-table entry 3-31
validation 11-5
  of registers 11-9
  of storage 11-6
  of storage key 11-7
  of TOD clock 11-9
validity bits (in machine-check-interruption code) 11-19
validity bits (in subchannel logout) 16-37
variable-length field 3-3
version code 10-43
virtual address 3-5
virtual storage 3-21
volatile storage 3-2
  effect of power-on reset 4-29

W
wait indicator 12-5
wait state (bit in PSW) 4-5
warning (machine-check condition) 11-16
  subclass-mask bit 11-22
word 3-3
  concurrency of reference 5-27
working allegiance (I/O) 15-9

LzLabs GmbH. Ex. 1016-565

wraparound
of instruction addresses 5-5
of PER addresses 4-14
of register numbers
for LOAD MULTIPLE instruction 7-25
for STORE MULTIPLE instruction 7-39
of storage addresses 3-6
comparison with System/370 D-10
for MOVE LONG instruction 7-28
of TOD clock 4-18
write (I/O command) 15-34

X
X (EXCLUSIVE OR) instruction 7-21
X field of instruction 5-5
XC (EXCLUSIVE OR) instruction 7-21
examples A-15
XI (EXCLUSIVE OR) instruction 7-21
example A-16
XR (EXCLUSIVE OR) instruction 7-21

Z
Z bit (zero condition-code bit) 16-12
ZAP (ZERO AND ADD) instruction 8-11
example A-34
zero
condition-code (Z) bit in IRB 16-12
instruction-length code 6-6
true 9-1
ZERO AND ADD (ZAP) instruction 8-11
example A-34
zone bits 8-1
moving of 7-31
zoned decimal numbers 8-1
examples A-5

3
370-XA mode
highlights of 1-1
selection by
architectural-mode-selection controls 12-2
IML controls 12-3

LzLabs GmbH. Ex. 1016-566

IBM System/370
Extended Architecture
Principles of Operation

Order No. SA22-7085-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. 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.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure-sensitive or other gummed tape to seal this form.

What is your occupation? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the front cover or title page.)

LzLabs GmbH. Ex. 1016-567

SA22-7085-0

Reader's Comment Form

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 PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department B98
P.O. Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

IBM

IBM System/370 Extended Architecture Principles of Operation (File No. S370-01) Printed in U.S.A. SA22-7085-0

LzLabs GmbH. Ex. 1016-568