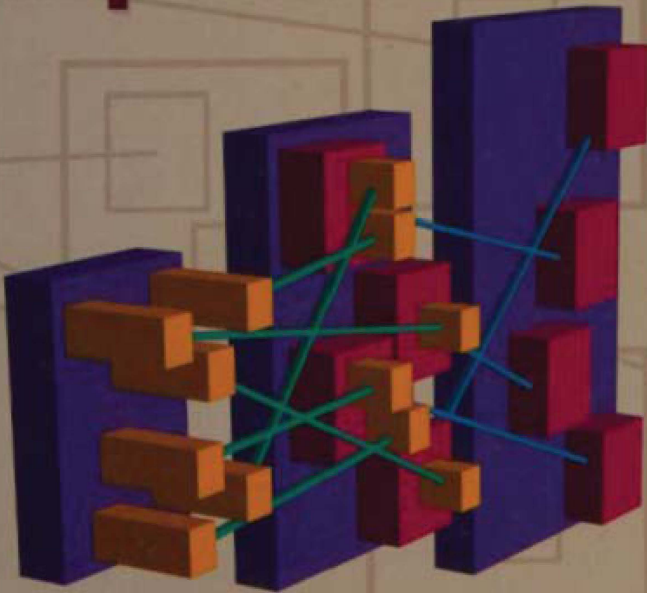


Computer Architecture and Implementation



Harvey G. Cragon

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS

The Edinburgh Building, Cambridge CB2 2RU, UK <http://www.cup.cam.ac.uk>
40 West 20th Street, New York, NY 10011-4211, USA <http://www.cup.org>
10 Stamford Road, Oakleigh, Melbourne 3166, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain

© Cambridge University Press 2000

This book is in copyright. Subject to statutory exception
and to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without
the written permission of Cambridge University Press.

First published 2000

Printed in the United States of America

Typeface Stone Serif 9.25/13 pt. System $\text{\TeX}$ 2_ε [TB]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging-in-Publication Data

Cragon, Harvey G.

Computer architecture and implementation / Harvey G. Cragon.
p. cm.

ISBN 0-521-65168-9 (hard.)

1. Computer architecture. I. Title.

QA76.9.A73C74 2000

004.2'2—dc21

99-16243

CIP

ISBN 0 521 65168 9 hardback

CONTENTS

Preface

page viii

1 COMPUTER OVERVIEW

1.0 Introduction	1
1.1 von Neumann Model	2
1.2 The von Neumann Architecture	5
1.2.1 The von Neumann Instruction Set Architecture	6
1.2.2 Instruction Interpretation Cycle	10
1.2.3 Limitations of the von Neumann Instruction Set Architecture	13
1.3 Historical Notes	13
1.3.1 Precursors to the von Neumann Instruction Set Architecture	14
REFERENCES	23
EXERCISES	24

2 PERFORMANCE MODELS AND EVALUATION

2.0 Introduction	26
2.1 Performance Models	26
2.2 Amdahl's Law	37
2.3 Moore's Law	39
2.4 Learning Curve	42
2.5 Grosch's Law	44
2.6 Steady-State Performance	45
2.7 Transient Performance	46
REFERENCES	47
EXERCISES	47

3 USER INSTRUCTION SET DESIGN

3.0 Introduction	50
3.1 Lexical Level	51

3.2	Instruction Set Architecture	51
3.2.1	Addresses	53
3.2.2	Operations	62
3.2.3	Data Types	74
3.2.3.1	User Data Types	75
3.2.3.2	Program Sequencing Data Types	88
3.2.4	Instruction Composition	93
3.3	CISC and RISC Architectural Styles	98
3.4	Static and Dynamic Instruction Statistics	103
3.5	Arithmetic	109
3.5.1	Addition/Subtraction	111
3.5.2	Multiplication and Division	114
3.5.3	Floating-Point Arithmetic	118
3.5.3.1	Precision Treatment	121
	REFERENCES	127
	EXERCISES	128
4	MEMORY SYSTEMS	131
4.0	Introduction	131
4.1	Hierarchical Memory	132
4.2	Paged Virtual Memory	133
4.3	Caches	145
4.4	Interleaved Real Memory	163
4.5	Virtual- and Real-Address Caches	171
4.6	Segmented Virtual Memory	173
4.7	Disk Memory	176
4.8	System Operation	179
	REFERENCES	181
	EXERCISES	181
5	PROCESSOR CONTROL DESIGN	183
5.0	Introduction	183
5.1	Hardwired Control	185
5.2	Microprogrammed Control	193
	REFERENCES	201
	EXERCISES	202
6	PIPELINED PROCESSORS	204
6.0	Introduction	204
6.1	Performance Models	205
6.2	Pipeline Partitioning	207
6.3	Pipeline Delays	211
6.3.1	Branch Delays	212
6.3.2	Structural Hazards	225
6.3.3	Data Dependencies	227

6.4	Interrupts	231
6.5	Superscalar Pipelines	234
6.6	Pipelined Processor Memory Demand	237
	REFERENCES	239
	EXERCISES	239
7	INPUT/OUTPUT	241
7.0	Introduction	241
7.1	I/O System Architecture	241
7.2	I/O Device Requirements	249
7.3	Buses and Controllers	252
7.3.1	Bus Design Examples	258
7.4	Serial Communication	258
7.5	Clock Synchronization	263
7.6	Queuing Theory	265
7.6.1	Open-System Queuing Model	265
7.6.2	Closed-System Queuing Model	271
	REFERENCES	273
	EXERCISES	274
8	EXTENDED INSTRUCTION SET ARCHITECTURES	275
8.0	Introduction	275
8.1	Operating System Kernel Support	276
8.2	Virtual-Memory Support	283
8.3	Interrupt Support	290
8.4	Input/Output Support	293
8.5	Cache Support	295
8.6	Multiprocessor Support	298
8.7	Other User ISA Enhancements, MMX	302
8.7.1	Other Intel ISA Support	307
	REFERENCES	308
	EXERCISES	308
	<i>Index</i>	310
	<i>Trademarks</i>	318

PREFACE

This textbook is intended for use in a one-semester, upper-division undergraduate course. It is expected that the students will have had courses in logic design with state machines and assembly language programming. A course in data structures as well as basic operating systems would be helpful.

There are two major themes running through this book. The first theme is an overview of the major ideas and design philosophies of computer architecture and implementation with some insight into their historical development. We are inclined to forget the work of our predecessors in computer design; the few pages devoted to this topic may help to rectify this neglect.

The second theme is the introduction and use of analytic modeling of computer performance. I believe that engineering students should take an engineering approach to the study of computer architecture and implementation. Various design trade-off issues will be examined with analytical models. What is an engineering approach? Consider the following two quotations.

“What distinguished the engineer from the technician is largely the ability to formulate and carry out the detailed calculations of forces and deflections, concentrations and flows, voltages and currents, that are required to test a proposed design on paper with regard to failure criteria. The ability to calculate is the ability to predict the performance of a design before it is built and tested.” (Petroske 1996).

“... it is much easier to have some vague notion about any subject, no matter what, than to arrive at the real truth about a single question, however simple that may be.” (Descartes 1629).

In addition to the two major themes, this book will introduce students to a new vocabulary. Whenever a new topic is approached, it is inevitable that one learns a new vocabulary. Therefore, whenever a new word or term is introduced, this book attempts to provide a definition along with common synonyms. We recommend that students obtain a copy of a computer dictionary to support their studies. Reference can also be made to an on-line dictionary of computer terms at (<http://wombat.doc.ic.ac.uk/foldoc/index.html>).

One of the problems in the computer field is the lack of a standard set of terms. For example, the Intel x86 architecture uses the term linear address whereas I prefer the

term virtual address, as used with the IBM S370 and S390. The reason for the preference is that the IBM S370 predates the virtual-memory version of the Intel x86 and established precedence. Thus, in the text, wherever possible, the term with precedence is used.

The von Neumann architecture is used in Chapter 1 to introduce the basic principles of computer architecture. Students should know what von Neumann proposed with this architecture and what its limitations were. These limitations were overcome in subsequent architectures while the underlying design remained unchanged.

Two architecture families will be extensively used for illustrations: The IBM S360/370/390 and the Intel x86 up to the Pentium Pro with MMX extensions. These two families represent the two most widely used processors today. For mainframe applications, the IBM family is preeminent in the world. The Intel family is found in approximately 90% of all personal computers. These two families are the most likely to be encountered by students after they finish their studies.

The sequence of presentation in this book differs from that found in other texts. It starts with a computer overview in Chapter 1 that covers the von Neumann architecture and some of its precursors. Chapter 2 introduces analytical performance modeling, learning curves, and other related evaluation techniques. The modeling technique will be used throughout the balance of the book.

Chapter 3 describes the issues in user instruction set design. The instruction set architecture taxonomy of addresses, data types, and operations is introduced. This chapter also provides a basic treatment of computer arithmetic and the IEEE floating-point standard.

Because a memory-centric view of a computer is taken, the text turns to memory implementation in Chapter 4. I believe that students should be exposed to memory systems before processors, as the process state is held in memory and is the focus of how problems are solved. Further, I approach the memory system by first looking at virtual memory that provides the functionality required of large programs and large data sets.

Performance follows functionality, from a designer's point of view; thus caches are covered after virtual memory. The gap between the cache and the disk is then covered by describing interleaved memory and DRAM organizations. Performance models for hierarchical memories are used throughout the chapter to illustrate design trade-offs.

Following the chapter on memory, the issues of processor implementation are addressed. First, hardwired and microprogrammed implementations are covered in Chapter 5. Then Chapter 6 addresses the ideas and issues of pipelining. The techniques of performance models are interwoven with the functional descriptions of processor implementation.

Chapter 7 covers input/output. In this chapter we look at requirements for various workloads and examine how these requirements are satisfied in contemporary systems. Bus design and serial communications are discussed along with clock recovery and clock synchronization. A brief introduction to queuing theory is included in this chapter.

The concluding chapter, Chapter 8, covers extensions to the user instruction set architecture that support operating systems, virtual memory, caches, and multiprocessors.

TABLE 3.2 INSTRUCTION FORMAT BITS

Architecture Type	Bits
Three address, memory to memory	$P + 3A$
Two address, memory to memory	$P + 2A$
Three address, register to register	$P + 3R$
Load/store	$P + A + R$
Two address, register to register	$P + 2R$
Load/store	$P + A + R$
One address, accumulator	$P + A$
Zero address, stack	P
Push/pop	$P + A$
Multiple accumulator	$P + A + R$

- Multiple accumulator:

Op-Code	A	R
---------	---	---

$$R(R) \leftarrow (R(R)) \text{ op } (M(A)).$$

The multiple accumulator architecture selects a register from the register file to be an accumulator. Because both a memory and a register address are available, special load and store instructions are not required.

The various instruction compositions (and local storage) are never found as “pure” architectures except in RISC designs. CISC ISAs are composed of several of these design options. The Intel Pentium supports load/store register file for integers, accumulator for integers, and an evaluation stack for floating point. A combination of the accumulator, register-to-register, and multiple accumulator architectures are used with the IBM S360.¹⁷

Bit Budget

Observe that each of these register organizations has differing numbers of bits in their instruction formats. A major consideration in ISA design in the early decades of the computer industry was the minimization of the bits required to represent a program. The number of bits is known as the bit budget.¹⁸ To illustrate how the bit budget is computed, assume that an op-code has P bits, a memory address has A bits, and a register address has R bits. Table 3.2 tabulates the number of bits required for each instruction format.

¹⁷ The designation S360 is used in this text to identify all members of this ISA family: S360, S370, S390. The specific designation will be used when the feature being discussed applies to one of the family members.

¹⁸ Bit budget may be measured in bits, bytes, words, or instructions. Any consistent metric will suffice.

system. Following the technique of the example in Subsection 3.2.3.1 we find that 3 bits are saved in the exponent for equal range.

Comment

From a bits-lost-or-saved point of view, base 256 seems to be a poor choice: lose 4 bits in the fraction, save 3 bits in the exponent.

Logical Data Types

Many computers define logical data types, also called Booleans, that are a vector of bits allocated to a byte, half-word, word, or double-word primary data type. These bit vectors are used to record binary events or states. Logical data types are used primarily for housekeeping tasks within a program and are rarely found as the input or the output of a program.

The S360 allocates logical values to any of the data types, bytes, half words, words, and double words. The Pentium Pro can allocate a bit field of any length, as shown in Figure 3.11. One address specifies the start of the field, and a length parameter specifies the number of bits in the field.

3.2.3.2 Program Sequencing Data Types

A number of architected data types are used for controlling the sequence of the executing programs. These data types are the program counter and the indicator bits.

Program Counter

With modern computers, a program is stored in sequential memory locations and is accessed by the program counter. The program counter data type is an unsigned integer having a length equal, in many architectures, to one of the primary data types of the ISA. It is convenient to have the program counter data type the same length as a user data type, so it can be stored in memory with a context switch.²⁹ The Pentium Pro program counter is a 32-bit register called the extended instruction pointer (EIP) register.

The IBM S360 takes a different approach. The program counter (called the instruction address) is allocated in the 64-bit program status word (PSW), a double word that can be conveniently stored. The S360 and S370 program counters are 24 bits, whereas the S390 has a 31-bit program counter.

Branches require an EA that is loaded into the program counter when a branch is taken. There are three common forms of branch EAs that become the new program counter value. The displacement is the displacement field of the branch instruction:

Direct	$PC \leftarrow (\text{displacement})$
PC relative	$PC \leftarrow (PC) + (\text{displacement})$
Base relative	$PC \leftarrow (\text{base}) + (\text{displacement})$

²⁹ Some single-chip microcomputers do not follow this convention. The length of the program counter is set to address the on-chip program ROM. A special hardware stack is provided to save the program counter for the return address when calling a subroutine or when responding to an interrupt.

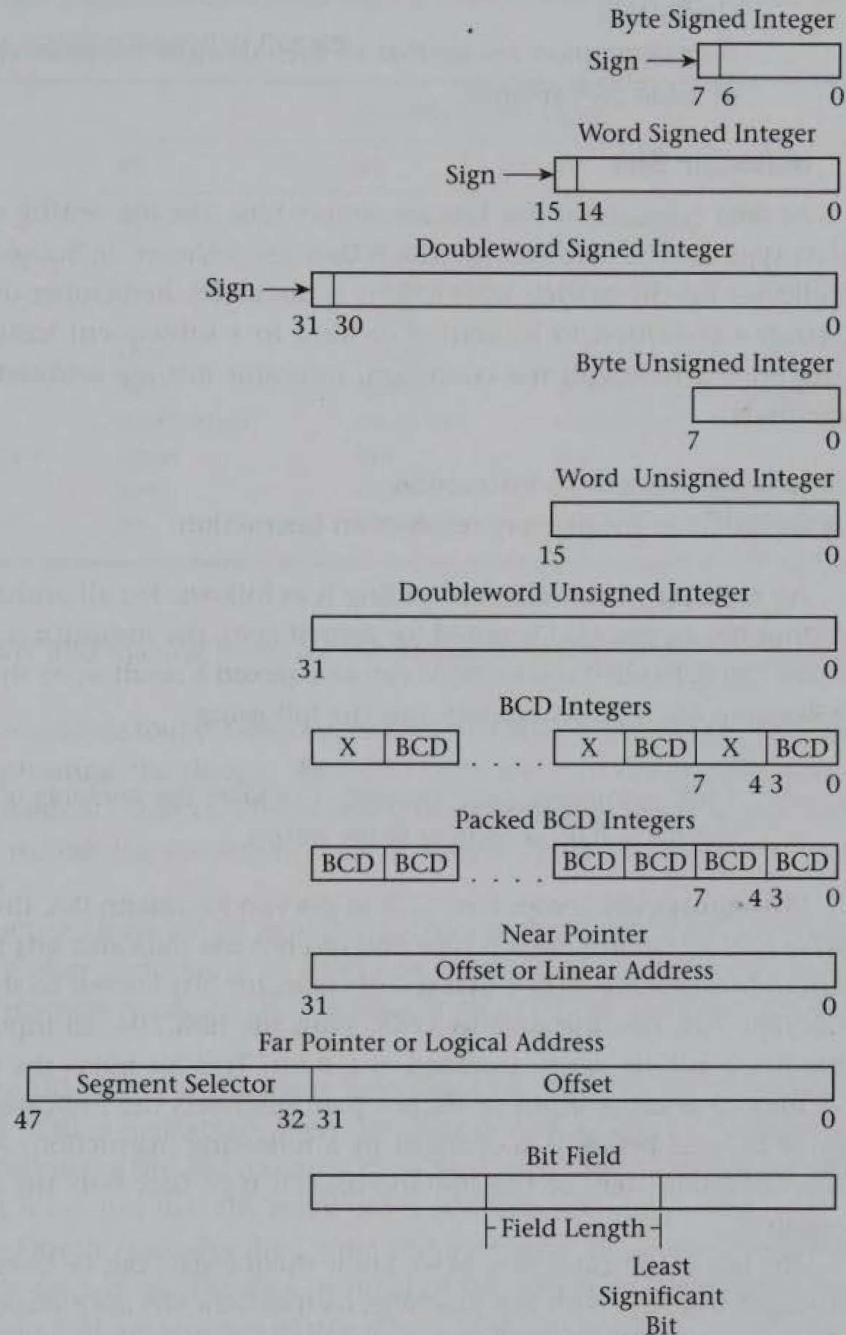


Figure 3.11 Pentium Pro data types (courtesy of Intel Corporation)

EXAMPLE

What happens when a displacement is added to the PC or a base value and there is a carry out?

Solution

There are two solutions to this design problem. First, the carry can be detected and an interrupt signaled, indicating an error. Second, the effective address wraps around and the branch is taken to a low-order address.

Comments

A designer must ensure that all such unusual situations do not result in unpredictable performance.

Indicator Bits

As data types, indicator bits are simply bits. The interesting characteristics of this data type are the methods by which they are set/reset. In Subsection 3.2.2 the use of indicator bits by branch instructions is discussed. Remember that the indicator bits permit a condition to be carried forward to a subsequent instruction that will test and branch based on the condition. Indicator bits are set/reset or recorded by two methods:

- as a side effect of an instruction
- explicitly as the primary result of an instruction

An example of side effect recording is as follows. For all arithmetic operations, the output bus of the ALU is tested for zero; if zero, the indicator is set to 1; otherwise it is reset to 0. Explicit instructions can also record a result in an indicator. For example, a Pentium Pro instruction performs the following:

CMP – compare two operands. Compare the contents of two registers and set the status flag according to the results.

With most early computers, such as the von Neumann ISA, the branch instructions were of the test-and-branch type and did not use indicator bits to carry forward condition information.³⁰ The first use of indicator bits known to the author is found in the IBM 704, first installed in 1955. With the IBM 704, an input byte is checked by hardware and its parity recorded in a P bit. This bit holds the parity condition until there is another input operation that sets/resets the indicator. Thus the indicator must be used before it is changed by a following instruction. All other branches in this computer were of the test-and-branch type that tests the condition of various registers.

The use of indicator bits, as we know them today, can be traced to the IBM Stretch installed in 1961. With this machine, 64 indicator bits are grouped into an architected double word that, in addition to being tested by branch instructions, could be stored and reloaded for context switching. Seven of the bits recorded the result of an integer ALU operation as side effects: negative, zero, positive, negative sign, low, equal, and high. Ten of the bits recorded the floating-point ALU results and exceptions.

IBM S360 Condition Codes. The IBM S360 carried forward the architected indicator bits idea from the IBM Stretch in a restricted fashion. The condition code is allocated to the architected PSW along with other status information. The designers of the S360 believed that the indicators of the Stretch could be encoded, thereby

³⁰ The von Neumann ISA could branch on a previously computed value in the accumulator, ≥ 0 . Likewise, the Bell Model V relay calculator could branch on the sign of a previously computed value.

TABLE 3.13 IBM S360 CONDITION CODES

Instruction	Condition Code Value			
	00	01	10	11
ADD logical	zero, no carry	not zero, no carry	zero, carry	—
SUBTRACT logical	=	not zero, no carry	zero, carry	not zero, carry
Test under mask	zero	mixed	—	ones
ADD	zero	<zero	>zero	overflow
Move long	count equal	count low	count high	dest. overlap
Compare floating point	equal	low	high	—
ADD decimal	zero	<zero	>zero	overflow
Store clock	set	not set	error	not operational

saving bits in the PSW and making room for the program counter and other control information.

Two bits are used to encode four conditions; encoded indicators are called condition codes. Further complicating the design, the conditions are recorded as side effects and are context dependent. That is, the meaning of the encoded value depends on the instruction that records the condition. A small sample of the condition coding is shown in Table 3.13.

The branch instructions have a 2-bit field that is compared with the 2-bit condition code. For example, if after performing an ADD one wants to test the result for zero, the 2 bits in the instruction would be 00. If a branch on overflow is desired, the bits would be 11.

The system of the IBM S360 opens many opportunities for programming errors. Say, for example, an ADD is performed; then the clock is stored. The information in the condition code following the add has been destroyed. A branch on 00 would now report that the clock is set, not that the result of the previous add is not zero.

Intel Architecture Family Indicator Bits. Intel literature refers to indicators as flags, the term used in this discussion. The first member of the x86 family, the i8008, has four implemented flags that are set/reset as side effects to other operations: carry, zero, positive, parity. Because the flags are implemented, their values cannot be stored and retrieved for context switching.

As the evolution of the x86 family progressed to the i8080 and the i8086, the i286 was the first member with architected flags. With the i286, 16 flags are allocated in a register that can be stored and retrieved for context switching. However, only 12 flags were assigned, leaving 4 bits for expansion.

The flags of the Intel Pentium Pro are shown in Figure 3.12. The flags are allocated to a 32-bit word. Note there are three classes of flags: status, control, and system. The auxiliary carry flag is used for performing a decimal adjust after a decimal addition. The alignment check flag is of interest because the x86 memory is byte addressable. If, for example, a word (4 bytes) is addressed and the address is not

- 3.20 The Atanasoff Berry Computer, described in Chapter 1, uses Gaussian elimination to solve linear algebraic equations; see Figure 1.9. Show how the ABC solves for x and y by using binary data types; $3x + 2y = 12$, $4x + y = 11$.
- 3.21 Can the two steps of the x86 decimal-adjust operation be combined into one step? If so, describe the operation. If not, explain why not.
- 3.22 The text states in Section 3.3 that one advantage of a fixed-length instruction is that instructions will not cross page boundaries. Explain why this is so.
- 3.23 Consult Intel documentation on any member of the x86 family and explain how signed decimal arithmetic is performed.
- 3.24 The combinations of fixed-length/variable-length instructions and op-codes are discussed in Subsection 3.2.4. Why is the combination of fixed-length instructions and op-codes not included in the list of design options?
- 3.25 The program counter of the VAX is allocated to register 15 of the register file. Discuss the advantages and disadvantages of this design.
- 3.26 Consult the documentation for the IBM S390 and construct the equivalent of Table 3.13 IBM S360 op-code classes.
- 3.27 The example in Subsection 3.2.3 shows that base 16 requires 2 fewer bits than base 2 for the same range. The equality was solved by assuming one base 16 exponent bit. Show another solution of the equality that makes no such assumption and shows that the result is general.
- 3.28 The average length of an 8086 instruction was given as 2.25 bytes in Section 3.3. This average was found by running programs in 1989. At this time, do you think the number of bytes per Pentium Pro instructions is less than or greater than 2.25? Explain your answer and give an estimate that you would use for modeling purposes.
- 3.29 Provide a rationale for the round-to-nearest-domain function of the IEEE floating-point standard for the case in which $R = 1$ and $S = 0$.
- 3.30 Figure 3.9 shows three classes of interrupt handling methods. However, the text suggests that there are four: the combinations of between instructions or within and instruction and continuation and restart. Explain how these four methods are accommodated in Figure 3.9.
- 3.31 Generalize the rules for making the correction to biased exponents after an addition or subtraction.
- 3.32 Are the Intel integer data types one of the six data types described in Table 3.6? Explain your answer.

that has a control bit (bit 15) set to supervisor mode.⁷³ The user PSW is saved in memory location 32. After the supervisor call interrupt has been serviced, the privileged load PSW (LPSW) instruction restores the old PSW and restarts the interrupted program.

Comment

If the PR instruction is executed and the processor is not in the supervisor mode, an illegal op-code interrupt will be generated when the instruction is decoded.

The IBM S390 introduces semiprivileged instructions that can be executed in the user mode when certain authority requirements are met. Likewise, the Intel Pentium Pro has four levels of privilege. Level 0 is the operating system kernel, levels 1 and 2 are the operating system services (device drivers, etc.), and level 3 is the user or application level.

In this chapter, we discuss the privileged instructions, their operations, data types, and addresses. Two classes of extended user instructions are also discussed in Section 8.7: multiprocessor support and MMX enhancements to the Intel Pentium.

8.1 OPERATING SYSTEM KERNEL SUPPORT

A subset of privileged instructions, operating system kernel support, is discussed in this section. There are some operating system instructions that are similar to user instructions except that they use different registers. Processors may have a set of registers that are addressed only by the privileged instructions. Thus there are, in essence, other processors embedded in the user processor. However, there are common resources such as caches, ALUs, and implemented registers used by all instructions.

The processor state consists of the system data types that are collections of bits, fields of bits, and words allocated to registers that are used as semifixed indicators of some state of the process and processor. These registers hold indicator bits that define at any time the current operating mode and conditions of the processor. The processor state registers usually include the indicator bits, discussed in Chapter 3 in the context of branching. In addition to the state register data types, the memory addresses associated with these data types are described along with the operations that can be performed on the data types. The processor state registers for the IBM S370 and the Intel Pentium Pro are

IBM S370 (IBM 1976):

PSW

Control registers

Intel Pentium Pro (Intel 1996):

Flag register

Control registers

Instruction pointer register

⁷³ IBM calls this bit *P* for problem state bit.

The PSW, flag register, and the instruction pointer register hold bits that may be changed dynamically during program execution. Control registers hold bits that are not normally modified during program execution. These bits establish the mode of operation and are swapped to/from memory because of a context switch or, with some processors, a subroutine call.

The data types described in this chapter have many bits and fields allocated to computer words. Students need not memorize these allocations, as detailed knowledge can always be obtained from the manufacturers' literature. However, students should understand why the data types exist and have a general understanding of the function and the purpose of the data types.

The operating system kernel data types exist to control the system, not produce results. Thus they never leave the system as outputs. The data types are transferred to dedicated registers in the processor, where they perform their intended function. These data types may be modified by privileged instructions during program execution or by side effects to user instructions.

IBM S360, S370, and S390

The IBM S360 family of computers was designed from the beginning to be controlled by an operating system. Thus a number of hardware and software concepts were introduced that facilitate the design and use of an operating system. We look at the data types, addresses, and operations that support the operating system kernel.

Data Types. Figure 8.1 shows the PSWs for the IBM S360 basic control (BC) mode and the S370 extended control (EC) Mode. The S370 extended the S360 with virtual memory requiring additions to the S360 PSW. Note that the program counter, called the instruction address, is found in the PSW as is the 2 bit conditing code (CC). Bits 0–7 set the control of the interrupts.

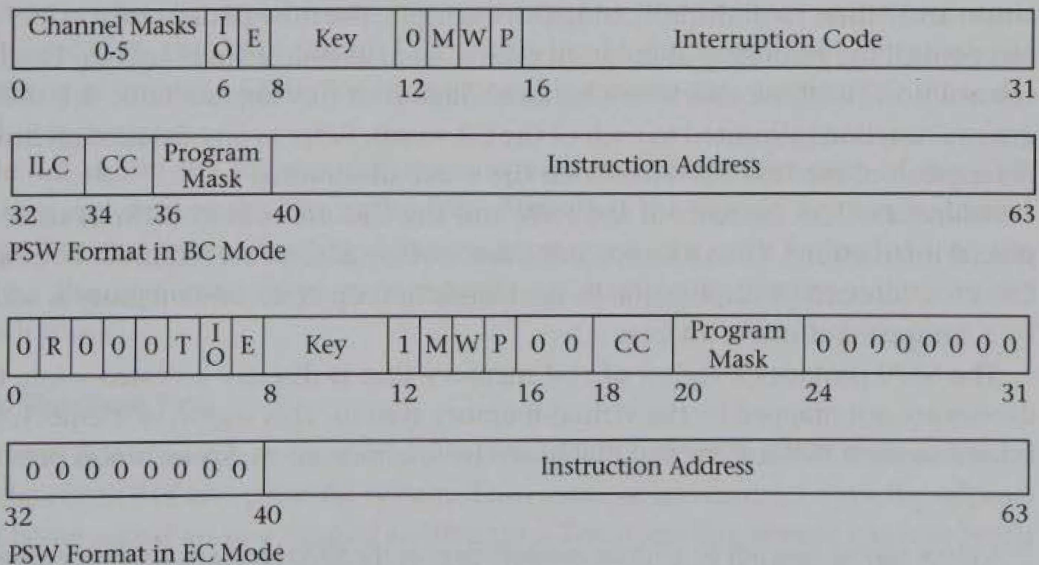


Figure 8.1 IBM S360, S370 PSWs (courtesy International Business Machines Corporation)

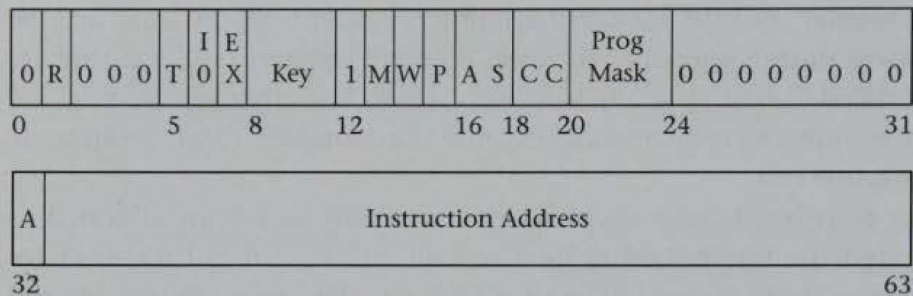


Figure 8.2 IBM S390 PSW (courtesy International Business Machines Corporation)

A 4-bit key provides protection for 2048-byte blocks in memory.⁷⁴ Each block in memory has an associated 7-bit key, called key in storage. For CPU writes, the 4-bit key in the PSW must match the corresponding 4-bit key in storage. CPU reads are permitted if the key matches or if bit 4 in the key in storage is zero. This key mechanism provides protection from unauthorized reads and writes. The key in the PSW is modified by a privileged instruction.

The 4-bit program mask augments the condition code for fixed-point overflow, decimal overflow, floating-point underflow, and floating-point significance when the result fraction of a floating-point operation is zero. An interrupt is generated when any of the mask bits are set to 1. The operating system kernel then determines the source of the interrupt by examination of the condition code, key, and/or the interrupt code.

The S390 PSW, EC mode, reallocates all the bits except the instruction address, key, and a few other bits, as shown in Figure 8.2. The major difference from the S370 is that the instruction address has been increased to 31 bits from 24. Reference should be made to an operations manual for a full description of the bits.

In addition to the PSW data type, the IBM S370 has a 16-word 32-bit control register (CR) file.⁷⁵ One or more bits in these registers is assigned to functions needing control or specification. Some of the bits are assigned to virtual-memory address translation and others to multiprocessing (for example, the time of day clock mask). Other bits control the recording of program events, such as successful branches. Finally, bits control how machine checks are handled: logout or not logout. Table 8.1 shows the general functions allocated to each of the CR words. Refer to any detailed architectural description of the IBM S370 to obtain the exact allocations.

Addresses. The contents of the PSW and the CRs are moved to/from memory by special instructions. Thus a source and a destination address are required. The PSW and CRs are addressed by implication in the instruction op-code, and memory is addressed by a computed effective address.

The S370 provides a region of real memory that is directly accessed – i.e., the addresses are not mapped by the virtual-memory system. This region of memory, 0–1FC, is used to store PSWs, CRs, and machine-check information. Space is also provided to

⁷⁴ A block may or may not be a virtual-memory page as the S370 can have 4096-byte pages.

⁷⁵ The S360 did not have CRs; they were added with the S370 and their use expanded with the S390.

TABLE 8.1 IBM S370 CR ALLOCATIONS

Word	Primary Use
0	Multiprocessing, dynamic address translation, timer
1	Dynamic address translation
2	Channel masks
3	Unassigned
4	Unassigned
5	Unassigned
6	Unassigned
7	Unassigned
8	Monitor
9	Program-event recording
10	Program-event recording
11	Program-event recording
12	Unassigned
13	Unassigned
14	Machine-check handling
15	Machine-check handling

store floating-point and general register values. The reason this region is direct mapped is to ensure that its contents are never paged out to the disk; the contents are always directly accessible by the processor.

Operations. Before the operations are described, it is instructive to examine the six instruction formats of the IBM S370 shown in Figure 8.3. The instructions can be 2, 4, or 6 bytes long. A wide variety of addressing options are provided. The formats associated with each of the instructions of interest are indicated in the text by their two-character format codes, such as RR for register to register.

The IBM S360 has 11 privileged instructions: 5 for the kernel (plus 1 nonprivileged instruction), 4 for I/O, and 2 for multiprocessing. The IBM S360 set of instructions is a smaller set than the 26 of the IBM S370 and the 55 privileged and semiprivileged instructions of the IBM S390. For instructional purposes, the simple S360 set is described.

Table 8.2 shows the instructions that are used to modify the status of the CPU by loading and storing the bits in the PSW. Note that the storage key can be loaded or stored but the system mask and the program mask can only be loaded into the PSW. The diagnose instruction invokes hardware diagnostics that are unique to the processor model.

Intel Pentium Pro

The Intel Pentium Pro evolved from the Intel 8008, which was not designed to be under the control of an operating system. Thus some of the features have the appearance of being added on to a simpler architecture. The operating system has four levels of privilege with processor state data types, addresses, and operations for the support of the operating system kernel.

A processor accesses the semaphore and determines if it can use the datum or if the datum is in use by another process. Reading the semaphore must be done in such a way that any other processor cannot attempt to access the semaphore while this is going on. In other words, there must be exclusive access to the semaphore in real memory. A basic design problem in achieving this end is that the instruction must be able to read-modify-write to the memory location holding the semaphore; two data memory cycles are required.

One way of ensuring exclusive access to the semaphore is by means of an atomic instruction. An atomic instruction is an instruction that performs an operation that cannot be interrupted or terminated by another processor and the semaphore cannot be accessed by another processor during the execution of the instruction.

Two distinctly different techniques are used to achieve atomic synchronization instructions with the IBM S360 and the Intel x86 architecture (before the Pentium Pro). These are discussed in the two subsequent subsections.

IBM S360

One type of synchronization instruction is test and set, first used on the IBM S360 and included in the ISA of the S370 and S390 (Falkoff et al. 1964). This instruction is not privileged, is atomic, and performs the following steps⁸⁰:

1. As a side effect, it locks out the memory bus from other users.
2. It copies the state of *S* into the CC bits of the PSW.
3. It performs a write that sets *S* to 1. If *S* is already 1, its value remains 1. (Collectively, steps 1, 2, and 3 are the test and set instruction. After this instruction is executed, the lock on the memory bus is released.)
4. The processor then tests the condition code with a branch instruction to determine if access to the datum has been obtained. If $CC = 0$, access is obtained, go to step 5. If $CC = 1$, access is not obtained; the processor tries again by returning to step 1.
5. The processor uses the protected datum, which may include modifying the datum, and goes to Step 6.
6. It executes an atomic store instruction that clears *S* to 0, thereby releasing the datum to another user, then continues.

Data Types. There are two data types associated with synchronization. The first is the semaphore, which is a byte rather than a single bit. This approach wastes 7 bits but simplifies the design as this architecture addresses to the byte; all the bits of the byte are either 1 or 0.

The second data type is the 2-bit CC field of the PSW shown in Figures 8.1 and 8.2. These bits are set to 00 for a semaphore of 0 and 01 for a semaphore of 1.

⁸⁰ IBM does not use the word atomic. Instead they use the phrase "appears to be an interlocked-update reference as observed by other CPUs." An I/O channel is not considered to be a CPU and interlocks are not provided.

TABLE 8.18 IBM S360 PROCESSOR SYNCHRONIZATION INSTRUCTIONS

Name	Mnemonic (Format)	Function
Test and set	TS (SI)	The byte at address ($D_1 + (B_1)$) is copied into the condition code and 1 is written into all bits of the addressed byte
Move	MVI (SI)	Moves the immediate byte of all 0s in bits 8–15 into the byte at address ($D_1 + (B_1)$)

Addresses. The effective address of the semaphore byte is base plus displacement. The other address for test and set, the CC, is implied in the op-code. The atomic store instruction uses an immediate operand of all zeros for the source. The destination address is the effective address of the semaphore byte.

Operations. Synchronization is provided by two atomic instructions, shown in Table 8.18. There are a number of complications in implementing the test and set instruction. For example, during a test and set instruction, the bus and memory are locked out from other users. As this instruction can take many clocks, the bus utilization can be adversely affected. An extreme case for the S370 and S390 would be that the first time the semaphore is accessed, it may be on the disk in virtual-address space, requiring a page fault, all the while locking out other processors from the memory system.

Other Multiprocessor Instructions. The IBM S360 provides a direct path for processors to communicate without using the system bus. Each processor has eight direct-in lines, eight direct-out lines, and eight direct-signal lines (bits 8–15 of the SI format instruction). Two instructions, shown in Table 8.19, use these lines for communication between other processors in the system.

There is a strobe signal with model-specific timing specifications that are not pertinent to the discussion of ISA.

TABLE 8.19 IBM S360 INTERPROCESSOR COMMUNICATION INSTRUCTIONS

Name	Mnemonic (Format)	Function
Read direct	RDD (SI)	Bits 8–15 of the instruction are placed on the signal-out lines. The byte on the direct-in lines is written into the byte at memory address ($D_1 + (B_1)$)
Write direct	WRD (SI)	Bits 8–15 of the instruction are placed on the signal-out lines. The byte at memory address ($D_1 + (B_1)$) is placed on the direct-out lines

TABLE 8.22 INTEL PENTIUM PRO PERFORMANCE MONITOR EVENTS

Unit Monitored	Event Description
Data cache	• All memory references • Weighted number of cycles while a data cache miss is outstanding
Instruction fetch unit	• Number of instruction fetches • Number of instruction TLB misses • Number of instruction fetch misses
L2 cache	• Number of L2 instruction fetches • Number of L2 data loads • Number of L2 data stores
Floating-point unit	• Number of computational floating-point operations retired • Number of computational floating-point operations executed

Note that these performance monitoring facilities are similar to those found with the IBM S370. Table 8.1 shows that words 9, 10, and 11 are allocated to recording program events.

REFERENCES

- Dijkstra, E. W. (1968). "Co-Operating Sequential Processes," in *Programming Languages*, F. deGenuys, ed., Academic, New York.
- Falkoff, J. W., Iverson, K. E., and Sussenguth, E. H. (1964). "A Formal Description of System /360," *IBM Systems Journal*, **3**:3.
- Flynn, M. J. (1966). "Very High-Speed Computing Systems," *Proceedings of the IEEE*, **54**, pp. 1901-1990.
- IBM (1970). "IBM S/360 Principles of Operation," order GA22-6821-8.
- IBM (1976). "IBM System/370 Principles of Operation," order GA22-7000-5.
- IBM (1997). "Enterprise System Architecture/390 Principles of Operation," order SA- 7201-04.
- Intel Corporation (1993). "Pentium Processor User's Manual," Vol. 3, order 241-430-001.
- Intel Corporation (1996). "Pentium Pro Family Developer's Manual," Vols. 1-3, order 000901-001.
- Intel Corporation (1997). "Intel Architecture MMX™ Technology, Programmer's Reference Manual," order 243007-003.
- May, C., Silha, E., Simpson, R., and Warren, H., eds. (1994). *The PowerPC Architecture*, Morgan Kaufmann, San Francisco, CA.
- Peleg, A. and Weiser, U. (1996). "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, **16**:4, pp. 42-50.

EXERCISES

- 8.1** Describe the IBM S360/370 channel ISA in terms of data types, addresses, and operations.
- 8.2** What instruction could be used with the IBM S360 to determine if the condition code permits access to a datum that is protected with a semaphore?

- 8.3 Consult the Intel Pentium Pro documentation on performance monitoring and determine how the miss rate of the instruction cache and the data cache can be determined.
- 8.4 The paper "PicoJava: A Direct Execution Engine for Java Bytecode" by McGhan and O'Connor, found in *Computer*, October 1998, describes a processor for executing Java programs. Write an essay describing the salient architecture and implementation features of this processor in 500 words or less.
- 8.5 Describe the meaning of the circled numbers 1, 2, and 3, of Figure 8.6.
- 8.6 Section 4.6 describes four segment/page configurations that are used with the IBM S370. For each of these configurations, list the number of bits that are added in the two adders and the bits concatenated in the real-address register of Figure 8.6. Recall that the S370 has a 24-bit effective or logical address.
- 8.7 What is the length of the virtual address of the Pentium Pro?
- 8.8 Figure 8.8 shows the Pentium Pro segmented virtual-address system for 4K-byte pages. The system can also support 1M-byte pages. Show how the linear address and tables are reconfigured for 1M-byte pages.
- 8.9 Explain why two adders are used for the S370 virtual-address translation, Figure 8.6, whereas only one adder is used for the Pentium Pro, Figure 8.8.
- 8.10 Consider the synchronization instructions discussed in Section 8.6. What happens when two or more requests to access the semaphore arrive on exactly the same clock?
- 8.11 Derive the model for MIOPS used in the example in Section 8.7.
- 8.12 What is done with the two 32-bit results computed in the multiply and add instruction shown in Figure 8.19? In other words, how can these results be used as inputs to subsequent instructions? Note that the source operands are 16 bits.