IBM

Student Text

Introduction to

IBM System/360 Architecture

Preface

This text is intended to introduce the student to the characteristics of System/360. It is expected that the student has some knowledge of computing systems.

No attempt at completeness has been made and, therefore, it is expected that the student will refer to the appropriate Systems Reference Library (SRL) publications for additional detail.

Minor Revision (April 1968)

This edition, C20-1667-1, is a minor revision of, but does not obsolete, the preceding edition, C20-1667-0. Minor changes have been made on pages 18 and 19.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for readers' comments. If the form has been removed, comments may be addressed to IBM, Technical Publications Department, 112 East Post Road, White Plains, N. Y. 10601

© International Business Machines Corporation 1967

Contents

Introduction 1
System Features for New Application Approaches 2
Channel Concept 3
Selector and Multiplexor Channels 3
Interrupts 4
Program Status Words and their Control of Interrupts 6
Data Representation 8
Arithmetic Operations 10
Sign Codes 12
Boundary Alignment 13
General Registers and Storage Addressing 14
Instruction Formats 16
RR Format 17
RX Format 17
RS Format 17
SI Format 17
SS Format 17
Protection Features 18
Floating-Point Arithmetic 20
Channel Organization 21
Summary 24
Questions and Exercises 25
Answers to Questions and Exercises 26

Introduction

This text introduces the student to the architecture of System/360. Such System/360 features as channels, automatic interrupts, and general purpose registers are presented. Storage addressing, instruction formats, data formats, and the various types of arithmetic operations are also discussed.

Questions and exercises are provided at the end of this text to help the student review the material; answers follow the questions.

System Features for New Application Approaches

The demands made upon a data processing system normally increase in the volume of processing to be done and in the scope of applications for which the system is utilized. To allow for growth in volume, System/360 was designed for implementation over a wide cost and performance range and to maintain program compatibility among the various models. For growth in application scope, the logical structure is that of a general purpose system for commercial, scientific, communication, and control applications.

To the user, a concern more immediate than growth considerations is cost versus performance. Before selecting higher-performance equipment, it is important to achieve maximum throughput from a lower-performance (and lower-priced) system. Achieving maximum throughput means decreasing the time required to process a total number of jobs so that the backlog of jobs is reduced. There is often, however, an opposing objective of decreasing the turnaround, or response, time for a given job. A report that takes three minutes of processor time is needed within an hour, but another four-hour run in progress requires two more hours for completion. Can we disrupt the program in progress? The answer has depended on the system and the programmed facilities available for restarting an interrupted program.

Because System/360 was designed to encompass solutions to such problems in all areas of data processing, it is helpful to further examine some of these conventional problems and to consider recent application approaches.

The most basic concept of computing, with which we are all familiar, is a program of sequential instructions. The processing unit fetches an instruction, decodes and executes it, increments an instruction counter, and then repeats this sequence of operations. A branch causes the contents of the instruction counter to be replaced with another address, and processing is continued from this address. This machine instruction fetch-execute-increment cycle is still basic to digital computers. In programming, however, we have come a long way from routines that read a card, process the data from the card, and write the results with no concurrent or overlapped operation.

The degree of concurrent operation that can be achieved depends not only on machine facilities but also on the programming employed.

The processing unit may be used for some portion of time and encounters recurring delays while awaiting input/output operations. Then the I/O equipment may be idled while processing takes place. Further, a system must often be configured for the largest job at the installation. That largest job may be run infrequently and the many smaller jobs that use the processing unit's time may utilize only a small portion of the total system's capacity. Lost time on the processing unit, lost time on I/O equipment, and less-than-maximum storage utilization are all wasteful.

The designers of System/360 sought solutions to these problems with a design that allows and encourages maximum utilization of available system resources. First, this design philosophy recognizes that data processing systems and programming systems should be integrated and not developed independently. New and sophisticated control techniques incorporated into the equipment for maximum utilization of resources take over many functions that previously were the concern of the problem programmer or of programming systems programmers. This last statement is not intended to imply that programming systems are not essential to utilize the system, but rather that there is a larger degree of interplay between equipment and program. In fact, the equipment was designed to run with a monitor program in control. System/360 and its control program are indistinguishable to the problem programmer.

Another consideration in the system's design was to facilitate the newer application approaches to computing, such as communications and multiprogramming.

Communications applications include time sharing, message switching, and the whole area of teleprocessing. Time sharing or conversational mode is the use of a number of remote terminals where each terminal has access to the computer. Here each terminal may be regarded as a personal computer, and all the independent users have access to a single computer virtually simultaneously because of ultra-high processing and switching speeds.

Message switching involves a telecommunications network where messages from remote points are sent to a central location for routing to their destination.

A common teleprocessing application is the processing of inquiries from remote terminals. Each terminal user introduces data to the system, and programs residing in the system perform whatever processing is required. The message may be simply a query for information stored within the system or it may be data to be entered and processed (with or without an answerback).

The program that handles the messages is called the foreground program. Other processing may take place between the servicing of messages. This "background" program is interrupted and the "foreground" program assumes control upon the receipt of a message. When the message is processed and no other messages are held pending, the foreground program relinquishes control to the background program.

Maximum utilization of system resources becomes particularly vital to a communication (or teleprocessing) system where input is unscheduled, where jobs are stacked (that is, where a series of jobs is run under the control of a supervisory program with a minimum of operator intervention), and in multiprogramming.

In applications involving multiprogramming optimum use is made of all facilities by having the system operate upon multiple programs or routines (tasks). While one task awaits data from an I/O device, another task utilizes the processing unit, and still other tasks utilize other I/O devices. As soon as a task utilizing the processing unit must wait for an I/O operation, it relinquishes control of the processing unit, and a waiting task assumes control. (The size, speed, and configuration of the system determine whether multiprogramming is practicable.)

Channel Concept

One of the system features that facilitate the simultaneous operations necessary for maximum utilization of the system's resources is channel circuitry. The electronic circuitry of a channel may be regarded as a small, independent computer that responds to its own set of commands. Channels provide the ability to read, write, and compute concurrently.

Each channel has its own program in main storage, and this program must be initiated by the supervisory program. A Start I/O instruction, for example, has the effect of selecting a specified I/O device and channel, and, if the device is available, starting the operation or operations specified by the channel program. In addition to the Start I/O instruction, there are three other instructions for communication between the processing unit and the channel: Test I/O, Halt I/O, and Test Channel.

These instructions are issued by the supervisory program, which contains an Input/Output Control System (IOCS). The address part of the instruction specifies the channel and the I/O device. When the channel and the device verify that the operation can be executed, the processing unit is released. The channel fetches its program from main storage and executes it. The transfer of data to or from main storage and the initiation of new operations by the channel program do not prevent processing of instructions by the processing unit.

Communications from the processing unit to the channels and I/O devices are discussed under "Channel Organization".

Selector and Multiplexor Channels

There are two types of channels: selector and multiplexor. Selector channels are used for the attachment of high-speed devices such as magnetic tapes, files, and drums. Multiplexor channels are intended primarily for low-speed devices. More than one device is

usually attached to either a multiplexor or selector channel through one or more control units. The control unit's functions are indistinguishable to the user from the functions of the I/O device, and in fact, some control units are physically housed within the I/O device. A control unit functions only with the type(s) of device(s) for which it is designed. Multiple I/O devices can be attached to a single control unit. Multiple tape units, for example, may be attached to a single tape control unit (see Figure 1).

When multiple slow-speed devices such as card readers are attached to a multiplexor channel, they can operate simultaneously through a time-sharing (interleaved) principle and processing can take place concurrently. When high-speed devices are attached to a multiplexor channel, only one device can operate at a time and the channel is said to be operating in burst mode. Operation of the Model 30 or 40 multiplexor channels in burst mode inhibits all other activity on the system. Selector channels always operate in burst mode and processing and I/O overlapping occur on all models.

As many as six selector channels can be operating concurrently with processing on Models 65 and 75.

Only one multiplexor channel can be connected to a processing unit. The number of selector channels that can be attached varies from two on a Model 30 to six on Models 65 and 75. The important thing to remember is that channels all appear to function identically to the user; it is only the degree of simultaneity of channel operations and overlapped processing that differs among the various models.

Interrupts

We have seen that the processing unit may initiate an input/output operation and resume processing while the channel proceeds independently. The processing unit must, however, maintain control over I/O operations. When an I/O operation is completed and a channel is free, another operation in the channel should be begun, if possible, to gain maximum channel utilization. Instead of having the problem program repeatedly interrogating channels to see whether they are free, the channels themselves signal the processing unit when they become free — that is, upon completion of a channel program. The channel signals cause the supervisory program to take appropriate action such as starting another I/O operation. These signals belong to one class of interrupts that the processing unit must be prepared to handle.

Here we begin to see how the circuitry takes over functions that were formerly the programmer's concern. The automatic interrupt system may be contrasted with a programmed branch in which the contents of the instruction counter are replaced rather than incremented. These branches are the programmer's concern. With the automatic interrupt system, however, an application program is written to include conventional testing and branching, but ignores those branches that will be handled as automatic interrupts. When an interrupt occurs, the contents of the equivalent of an instruction counter are automatically replaced. This suspends the operation of the program in progress temporarily. In addition the control and status information needed to restart the program are automatically stored by the interrupt system itself.

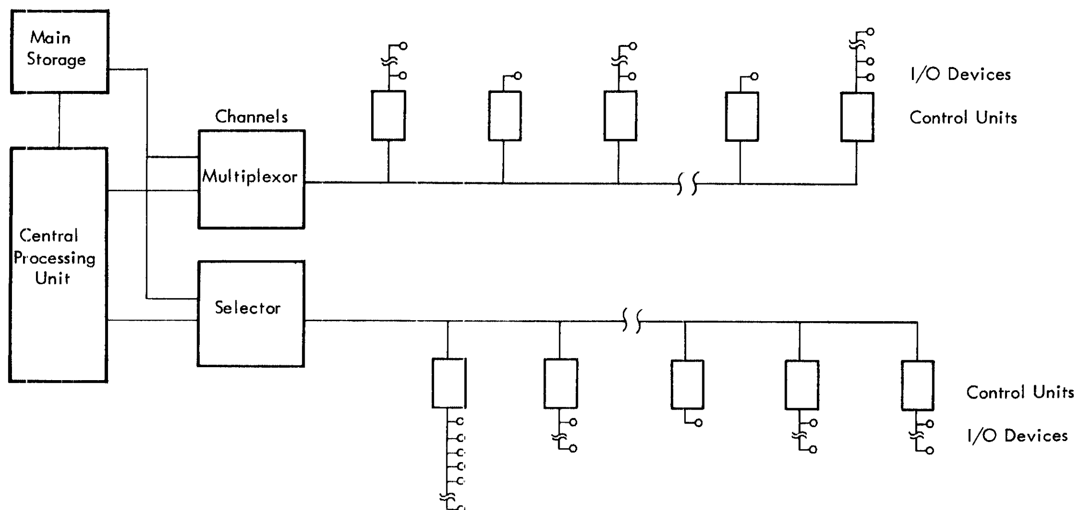


Figure 1. IBM System/360 basic logical structure

There are five classes of interrupts: input/output, program, supervisor call, external, and machine check.

- *Input/output interrupts.* The signal to the processing unit that a channel is free is typical of the class of interrupts called I/O interrupts. Special conditions in the channel or in an I/O unit cause the processing unit to take appropriate action.
- *Program interrupts.* Unusual conditions encountered in a problem program create program interrupts. Eight of the 15 possible conditions involve overflow, improper divides, lost significance, and exponent underflow. (Lost significance and exponent underflow may occur in floating-point arithmetic operations.) The remaining seven deal with improper addresses, attempted execution of invalid instructions, and similar conditions.
- *Supervisor call interrupts.* The significance of supervisor call interrupts will become apparent when we examine in more detail the effects of interrupts. Suffice it to say that Supervisor Call is an instruction that the program uses to cause an interrupt.
- *External interrupts.* Through an external call interrupt, the processing unit can respond to signals from the interrupt key on the system control panel, a built-in timer system, other processing units, or special devices.
- *Machine check interrupts.* A machine check condition initiates an automatic recording of the status of the system into a special scan-out area of main storage and then causes a machine check interrupt. A machine check can be caused only by a hardware malfunction and not by invalid data or instructions.

Some classes of interrupts can be ignored or held pending under program control. This prevents the interrupt from occurring and the interrupt is said to be "masked". An anticipated overflow is an example of an interrupt that the programmer would mask.

When the system is executing instructions of a problem program, it operates in what is called the problem program state. Interrupts that occur while the system is operating in the problem program state cause the processing unit to switch to the supervisory state. To ensure that the system has control over I/O functions, the control program takes control when an I/O instruction is required by a problem program. The control program operates in the supervisory state and includes a resident IOCS. Instructions that are executable only in the supervisory state are called "privileged".

A Supervisor Call instruction in a program is one method of causing a switch from the problem state to the supervisory state; that is, the problem program passes control to the supervisory program. An interruption code within the instruction may be used to convey messages from the calling program to the supervisory program. Two messages that the supervisory program would require are: (1) notification from the problem program that it is finished so that the supervisor can read in a new program, and (2) notification of requests to start I/O operations for the problem program. As soon as the I/O operation has begun, the supervisor program returns control to the problem program, which can continue processing while the I/O operation is taking place. Upon completion of the I/O operation, an I/O interrupt occurs. The supervisor program now determines whether any abnormal conditions were detected during the operation and takes appropriate action. The overall status of the processing unit is determined by alternatives other than the supervisor or problem state. These alternatives provide control of system resources by preventing a problem program from stopping the operation of the processing unit. There is no Halt instruction. In the problem state, processing instructions are valid but all I/O instructions and a group of control instructions are invalid. In the supervisory state, all instructions are valid.

The other alternative states are: running versus waiting state, masked versus interruptible state, and stopped versus operating state (see Figure 2).

In the running state, instruction fetching and execution proceeds in the normal manner. The wait state is typically entered by the program to await an interrupt – for example, an I/O interrupt or operator intervention from the console. In the wait state, no instructions are processed, the timer is updated, and I/O and external interrupts are accepted unless masked.

The processing unit may be interruptible or masked for the system (I/O or external), program, and machine interruptions. When the processing unit is interruptible for a class of interruptions, these interruptions are accepted. When the processing unit is masked, the system interruptions remain pending, but the program and machine-check interruptions are ignored. Instructions that alter the overall status of the processing unit are privileged.

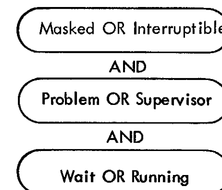


Figure 2. Alternative states of the processing unit in operation

Program Status Words and their Control of Interrupts

Passing control between problem programs and the supervisory program and returning to the right place in a program following an interrupt is accomplished with program status words (PSW's). Traditionally when information was required at some later point in a program, it was the programmer's responsibility to store it. With System/360, since the problem programmer cannot anticipate many interrupts, they become the responsibility of the system. Two storage locations are associated with each of the five classes of interrupts. One of the locations contains the address of the routine in the supervisory program that handles this class of interrupt. When an interrupt occurs, the system automatically replaces the current or active PSW, which contains an instruction counter plus other machine status information, with the PSW appropriate to this interrupt. This "new" PSW indicates among other things that the system is operating in the supervisory state and specifies the address of the routine that handles this class of interrupt. The PSW of the interrupted program is automatically stored as the "old" PSW (see Figure 3). The routine in the supervisory program that handles this interrupt will be run. Its last processing step will be to restore the old PSW as the active or current PSW, and the interrupted program will resume processing at the point where it was interrupted. Unlike the automatic switching of PSW's when an interrupt occurs, the replacement of the current PSW with the old PSW is accomplished by an instruction in the supervisory program. This programmed, rather than automatic, function was a deliberate design choice. Why, we may ask, does the Load PSW instruction need to address storage, since the system could readily determine the cause of the last interrupt? The answer is that in multiprogramming we frequently do not wish to return to the "task" last interrupted, but prefer that the control program stack up and control a sequence of PSW's.

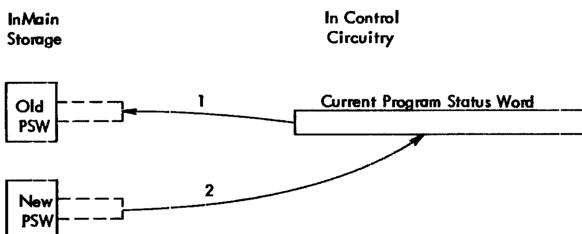


Figure 3. Interrupt program switching

Because the principle of the interrupt system is best understood in terms of the various PSW's, let us take a moment to examine their place and function. The old and new PSW's have permanent address assignments in main storage. The current PSW is contained in the control circuitry of the processing unit and, like an instruction counter, is updated as the program progresses. The new PSW locations contain the address of a routine to handle their particular class of interrupts. The addresses of these routines are not normally changed, and for a particular interrupt the same address will be read out each time this interrupt occurs. For each new PSW there is an old PSW that acts simply as temporary storage for the current PSW when an interrupt occurs. The interrupt causes the current PSW to be stored as the old PSW, and the new PSW becomes the current PSW. At the conclusion of the interrupted routine, the old PSW replaces the current PSW, restoring the system to its prior state and allowing the continuation of the interrupted program.

Old and new PSW's contained in storage are identical in format to the current PSW, since they are called upon and become "current". The location of old and new PSW's is shown in Figure 4. In the next topic, "Data Representation", we shall see that PSW's are doublewords with individual bits labeled 0-63. We can see now from the table that a machine check will cause the current PSW to be placed in storage locations beginning at 0048 and a new PSW will be brought out from locations beginning at 0112.

	Address	Length	Purpose
0	0000 0000	double word	Initial program Loading PSW
8	0000 1000	double word	Initial program Loading CCW1
16	0001 0000	double word	Initial program Loading CCW2
24	0001 1000	double word	External old PSW
32	0010 0000	double word	Supervisor call old PSW
40	0010 1000	double word	Program old PSW
48	0011 0000	double word	Machine check old PSW
56	0011 1000	double word	Input/output old PSW
64	0100 0000	double word	Channel status word
72	0100 1000	word	Channel address word
76	0100 1100	word	Unused
80	0101 0000	word	Timer
84	0101 0100	word	Unused
88	0101 1000	double word	External new PSW
96	0110 0000	double word	Supervisor call new PSW
104	0110 1000	double word	Program new PSW
112	0111 0000	double word	Machine check new PSW
120	0111 1000	double word	Input/output new PSW
128	1000 0000		Diagnostic scan-out area*

*The size of the diagnostic scan-out area depends upon the particular system's CPU and I/O channels.

Figure 4. Permanent storage assignments

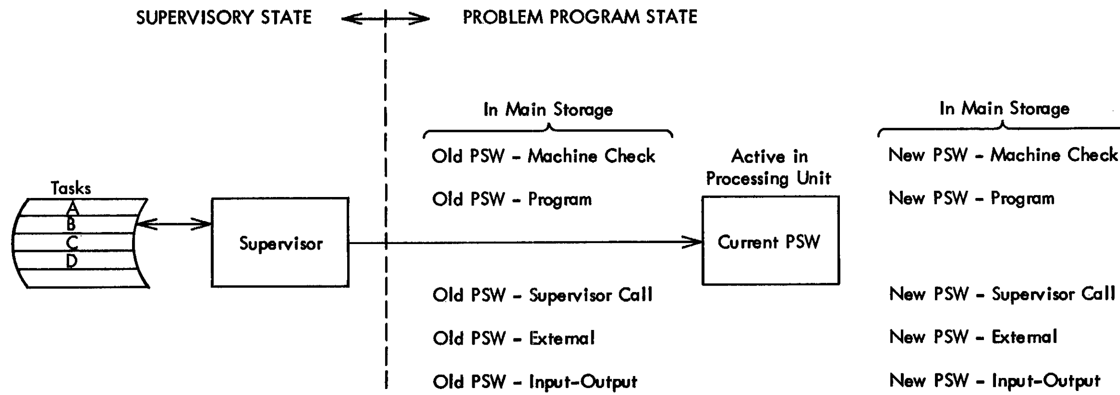


Figure 5. Problem program PSW active in processing unit contrasted with input/output operations in supervisory state

In a typical System/360 environment, more than one task is contending for time on the processing unit, and while one interrupt is being serviced, perhaps another interrupt occurs, while still another interrupt is held pending.

In Figure 5, the current PSW would reflect the status of a task B, which is being executed in the problem state.

In Figure 6, an interrupt has caused the processing unit to switch to the supervisory state. A new I/O PSW is replacing the active PSW and the active PSW is being stored as the old I/O PSW. Upon leaving the I/O routine (which is executed using the resident I/O supervisory program), the old I/O PSW will again become the current PSW, unless other interrupts occur.

We have seen that an interrupt causes a type of branch. What, we may ask, is the difference between a program branch and one caused by an interrupt? The portion of the PSW that has been compared with an instruction counter is called the instruction address. When a branch occurs, only the contents of the instruction address within the PSW are changed. On an interrupt the entire PSW is replaced. The PSW contains other status and control information in addition to the instruction address, which the processing unit requires. This includes such information as program status (supervisor versus problem state, masked versus interruptible state, stopped versus operating state, and running versus waiting state).

When interrupts occur is not the concern of the problem programmer. With reference to machine cycle time, it is interesting to note that the machine designers chose an optimum economic "interruptible" point, since status information must be saved and restored. This turns out to be after an instruction is finished and the next instruction is not yet started. In the case of I/O, external, or supervisor call interrupts, then, the current instruction will be completed before the interrupt is taken. However, in the case of program and machine errors, the end may be forced by suppressing or terminating the instruction's execution.

Other aspects and details of the automatic interrupt system are found in the appropriate SRL publications.

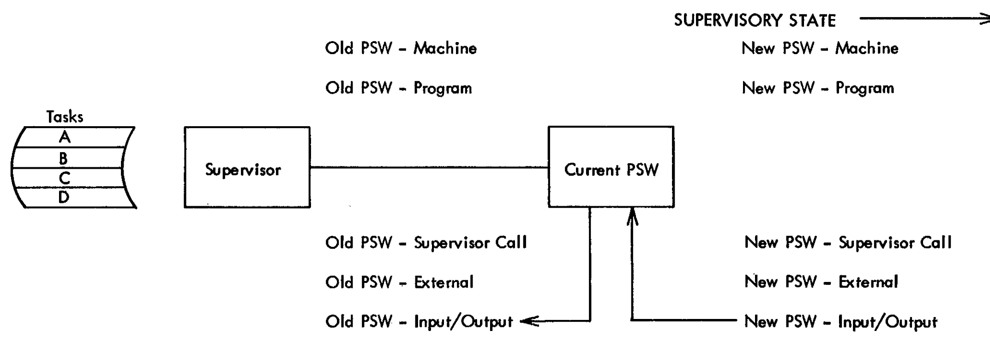


Figure 6. Switching of PSW's during an input/output interrupt

Data Representation

The most familiar method of data representation in commercial applications of computers has been binary coded decimal in which six bits are used to represent 64 alphameric and special characters. Records consist of many fields of widely different lengths. Scientific computers, on the other hand, generally operate upon fixed-word-length fields of binary data.

Several data formats can be used for processing with the System/360 to accommodate commercial and scientific applications. An eight-bit unit of information, called a byte, is fundamental to the formats. An initial byte may be addressed as an operand of an instruction, with the number of bytes used specified by the instruction. Because eight rather than six bits are used to represent a character, up to 256 possible characters could be represented in the Extended Binary Coded Decimal Interchange Code (EBCDIC) shown in Figure 7. Except for certain teleprocessing equipment, the code that makes use of characters is either EBCDIC or an eight-bit extension of a seven-bit code proposed by the International Standards Organization.

The chart shows bit positions, which determine bit patterns, at the top and to the left of each table.

The hole pattern of punched cards is shown at the bottom and to the right of each table.

The table at the upper left shows control characters. The explanation of their meaning is given in a separate listing. The characters PF, for example, indicate "punch off".

Exceptions to the tabular representation of hole patterns to specify a binary bit pattern, a control character, or a graphic character are identified by numbers circled in the table, and the proper hole patterns are shown in a separate listing below the tables. The examples given below the tables are self-explanatory and serve to ensure correct reading of the tables. To illustrate this, the last example in the list is an exception indicated by the number 4 circled in the table at the upper left.

For further practice, translate the name John Doe into EBCDIC and use initial capitals and lowercase letters. The results should be:

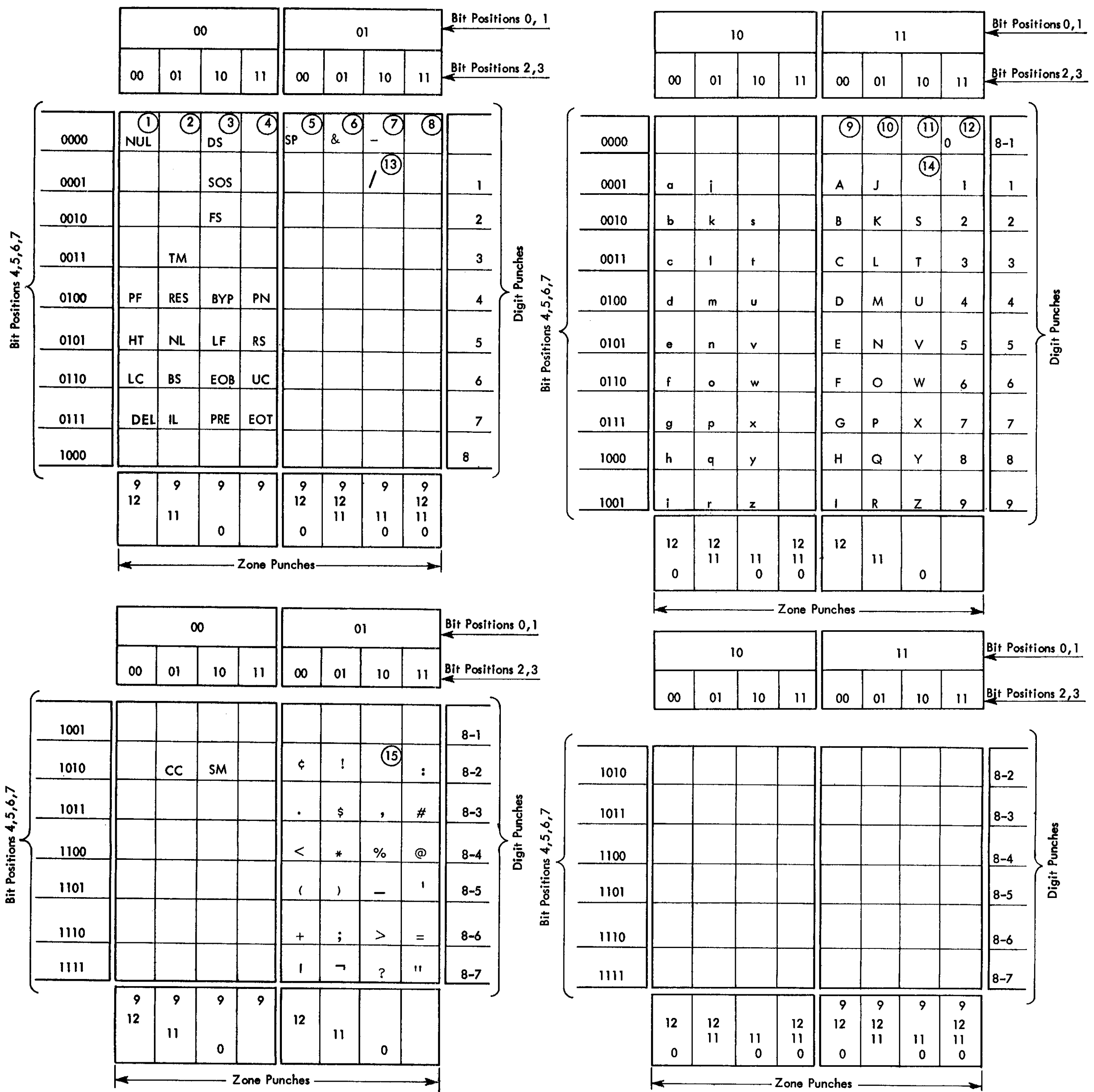
11010001	10010110	10001000	10010101
J	o	h	n

11000100	10010110	10000101
D	o	e

Note that in the tables the digits 0-9 have these bit configurations:

0	11110000	5	11110101
1	11110001	6	11110110
2	11110010	7	11110111
3	11110011	8	11111000
4	11110100	9	11111001

We may well ask what purpose the four leading 1s serve. The answer is that they provide a collating sequence in which numbers are higher than alphabetics in alphameric fields, but they are not used in arithmetic operations. Instead, an instruction is provided that "packs" two decimal digits into a byte by eliminating the leading 1s (see Figure 8). The decimal digits 0-9 are represented in the four-bit binary coded decimal form by 0000 through 1001. The elimination of the leading 1s (or zone portion) is accomplished with the Pack instruction.



① 12-0-9-8-1
② 12-11-9-8-1
③ 11-0-9-8-1
④ 12-11-0-9-8-1
⑤ No Punches
⑥ 12
⑦ 11
⑧ 12-11-0
⑨ 12-0
⑩ 11-0
⑪ 0-8-2
⑫ 0
⑬ 0-1
⑭ 11-0-9-1
⑮ 12-11

Control Character

NUL Null
PF Punch Off
HT Horizontal Tab
LC Lower Case
DEL Delete
TM Tape Mark
RES Restore
NL New Line
BS Backspace
IL Idle
CC Cursor Control
DS Digit Select
SOS Start of Significance
FS Field Separator
BYP Bypass
LF Line Feed
EOB End of Block
PRE Prefix
SM Set Mode
PN Punch On
RS Reader Stop
UC Upper Case
EOT End of Transmission
SP Space

Special Graphic Characters

¢ Cent Sign
. Period, Decimal Point
< Less-than Sign
(Left Parenthesis
+ Plus Sign
| Vertical Bar, Logical OR
& Ampersand
! Exclamation Point
$ Dollar Sign
* Asterisk
) Right Parenthesis
; Semicolon
¬ Logical NOT
- Minus Sign, Hyphen
/ Slash
, Comma
% Percent
_ Underscore
> Greater-than Sign
? Question Mark
: Colon
Number Sign
@ At Sign
' Prime, Apostrophe
= Equal Sign
" Quotation Mark

Example	Type	Bit Pattern Bit Positions 01 23 4567	Hole Pattern Zone Punches	Digit Punches
PF	Control Character	00 00 0100	12 -9	4
%	Special Graphic	01 10 1100	0	8 - 4
R	Upper Case	11 01 1001	11	9
a	Lower Case	10 00 0001	12 -0	1
	Control Character, function not yet assigned	00 11 0000	12 - 11 - 0 -9	8 - 1

Figure 7. Extended Binary Coded Decimal Interchange Code

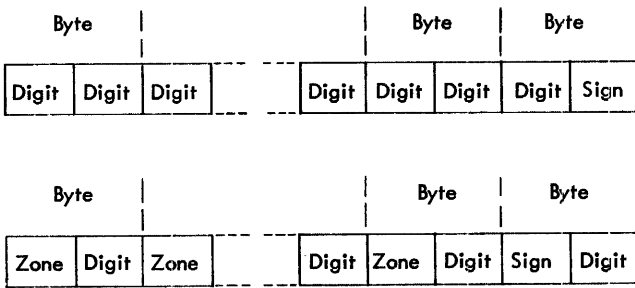


Figure 8. Packed and zoned decimal number formats

Arithmetic Operations

There are four classes of processing operations: fixed-point arithmetic, floating-point arithmetic, logical operations, and decimal arithmetic. Fixed-point arithmetic and logical operations are part of the standard instruction set. The decimal option is intended primarily for commercial applications and the floating-point arithmetic option is intended for engineering and scientific applications.

Fields of two, four, and eight bytes are called halfwords, words, and doublewords respectively (see Figure 9).

In fixed-point arithmetic the basic arithmetic operand is a signed value recorded as a binary integer, that is, a whole number (positive or negative) as contrasted with a fraction. It is called fixed-point because the machine interprets the number as a binary integer; that is, the point is to the right of the least significant position. The programmer has the responsibility for keeping track of an assumed point within a field.

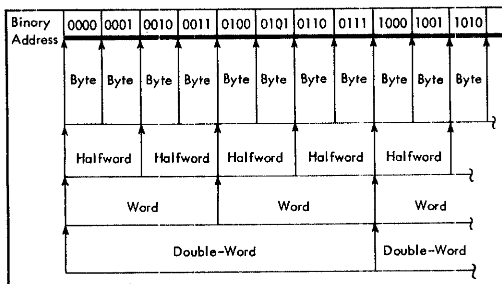


Figure 9. Halfwords, words, and doublewords as they appear in main storage

Fixed-point numbers occupy a fixed-length format consisting of a one-bit sign followed by the 31-bit integer field; alternatively, some operations may be performed on halfwords, and some multiply, divide, and shift instructions use a doubleword.

Until numeric data is ready for output on a device that uses characters, such as a printer or punch (character-set oriented), storage is most economically used by holding the data in binary or packed decimal digits.

In the following example of fixed-point arithmetic we shall, for the sake of simplicity, ignore the sign and fixed-length requirement.

Assume that a card reader has read the number 4096. The number itself will be transferred to main storage as four bytes of EBCDIC:

11110100 11110000 11111001 11110110

If this number is to be processed using fixed-point arithmetic, the PACK instruction is first issued and the number takes the binary coded decimal form:

0100 0000 1001 0110

A Convert to Binary instruction is then issued and, after its execution, the number takes the pure binary form:

1000000000000

which is 2^{12}.

Note that the decimal values of bit positions are:

128	64	32	16	8	4	2	1
7	6	5	4	3	2	1	0

The number itself is now ready for processing in fixed-point format. (Note that we have not illustrated the sign and length requirement.) After processing, a Convert to Decimal instruction and either an Unpack or an Edit instruction are used to prepare the output for a device using characters such as a printer or punch. If the results of processing are to be stored for further processing in binary form, the Convert to Decimal instruction and the Unpack or Edit instruction are omitted. If the results are to be stored as packed decimal digits, the Unpack instruction is omitted. Figure 10 shows this processing sequence.

No conversion from packed decimal to binary is necessary if the decimal instruction set is used. Instead, addition, subtraction, multiplication, division, and comparison are performed on packed binary coded decimal digits (see Figure 11). While fixed-point operations are performed on fixed-length fields, all decimal operations are performed on variable-length fields, the length of which is specified in the instruction. The address tells where the data is located, and the length specification tells how much data the instruction is to operate upon. From 0 to 15 bytes may be specified, so that, in effect, a 16-byte field may be addressed in arithmetic operations. A length specification of zero will address only the byte designated in the instruction address.

Where numerical information such as a part number is not operated upon arithmetically, it may be processed in the zoned format – that is, without packing the digits.

Now consider the facts that lead the programmer to decide whether to use decimal or binary arithmetic operations. Decimal arithmetic can make the programmer and the system more productive when processing requires relatively few computational steps between input and output. When extensive processing is required, as in many scientific applications, storage and circuitry are more efficiently utilized with binary numbers.

Note that the number 4096 requires 32 bit positions in EBCDIC, 16 bit positions in packed binary coded decimal, and 13 bit positions in pure binary. Does not the economy of the binary configuration suggest the efficiency of binary operations? Figure 12, however, demonstrates that the decimal instruction set is a more direct route from input to output. The criterion for selection is the amount of processing to be done in the blocks labeled "process with binary" and "process with decimal".

As shown in Figure 12, the system will accept as input any code that is eight bits or less. For these other codes, such as a teletype code, tables are set up in storage, and translate instructions permit conversion of entire records of up to 256 characters with a single instruction. The figure lists output as binary, packed decimal digits, or EBCDIC. Actually, as with input, the output could be in any code up to eight bits through the use of translation tables.

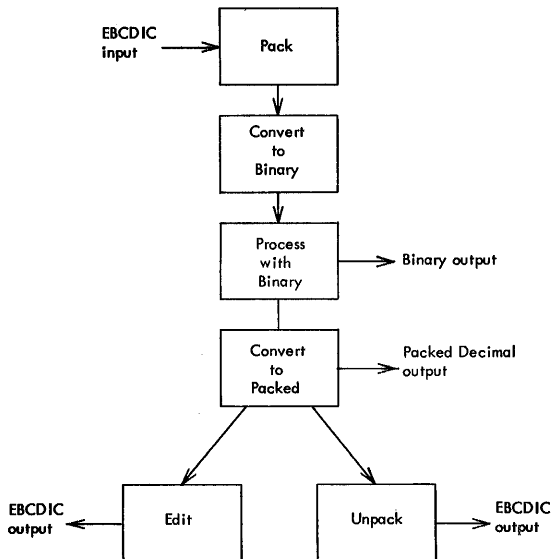


Figure 10. Fixed-point arithmetic processing sequence on EBCDIC input

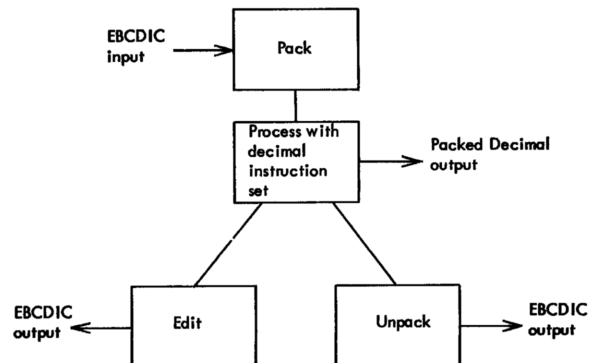


Figure 11. Processing sequence using the decimal instruction set on EBCDIC input

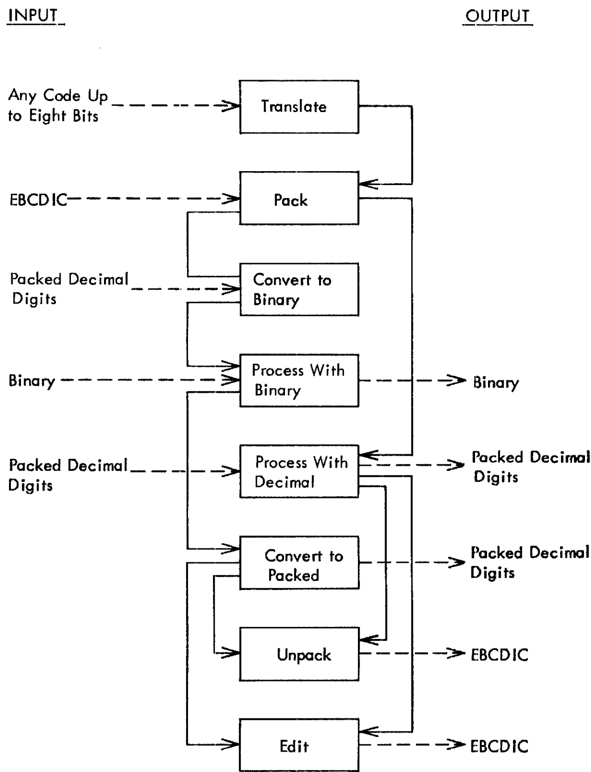


Figure 12. Various input processing sequences involving arithmetic

Sign Codes

When digits are read from cards, all unsigned digits are assigned the zone 1111 for EBCDIC. The sign patterns generated for EBCDIC are 1100 for plus and 1101 for minus. The usual case is that the sign occupies the zone positions of the least significant digit of a field. A three-digit field, then, would have this format:

zone digit zone digit sign digit

In EBCDIC a minus 123 would appear as:

1111	0001	1111	0010	1101	0011
	1		2	–	3

After a Pack instruction is issued, the four-bit sign pattern occupies the four least significant bit positions of the field, and other zone bits are eliminated. A packed three-digit signed field, then, has this format:

digit digit digit sign

The digits and the sign code occupy four bit positions each. A minus 123, for example, has this bit configuration:

0001	0010	0011	1101
1	2	3	–

A binary number used as a fixed-point operand occupies 31 bits of a word, or 15 bits of a halfword, in main storage. Another bit in the most significant position carries the sign, which is 0 for plus and 1 for minus (see Figure 13). Recall now that fixed-point operands are fixed in length. When the integer represented occupies less than a word or halfword, the sign bit is used to fill the unused high-order bit positions. The decimal number 4096, which we have seen is 1000000000000 in binary, can be represented in a halfword as 0001000000000000 if the sign is plus, or as 1111000000000000 in two's complement notation if the sign is minus. For a further explanation of complement notation see *Number Systems* (C20-1618).

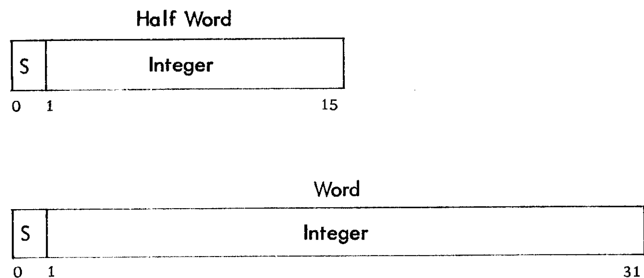


Figure 13. Fixed-point number format

We have seen that System/360 can be used as a fixed-point binary computer with fixed-length operands and that it can perform decimal arithmetic on records characterized by many fields of varying length. A consecutive group of n bytes constitutes a field of length n. We need these variable- and fixed-length capabilities for the most efficient handling of both commercial and scientific applications. It should be emphasized that storage is addressable to the byte. Some instructions that address a byte always operate upon that byte and the next three consecutive bytes, so that a four-byte word is the operand. Other instructions require that the programmer specify as part of the instruction the number of bytes that constitute the operand.

Mention has been made of bytes, halfwords, and doublewords. Actually, as many as 256 bytes can be specified as operands in some instructions, such as data transfers.

Storage addresses within the system are represented by binary integers starting at zero. The location of a stored field is specified by the address of the leftmost byte of the field.

Boundary alignment is a programming restriction on fixed-length operands that requires some explanation. A variable-length field of data may start at any byte location. A fixed-length field of two, four, or eight bytes must have an address whose decimal equivalent is a multiple of two, four, or eight bytes respectively. A word address, for example, must be divisible by four. These are called integral boundaries. In binary, it turns out that the address must have:

- One low-order zero bit for a halfword
- Two low-order zero bits for a word
- Three low-order zero bits for a doubleword

Because the operation code is examined to determine whether fixed-length data is a halfword, word, or doubleword, the system can check to see that data is aligned on proper boundaries. A violation will cause a program interrupt that can be identified by the interruption code of the program status word as being "specification". Figure 14 shows various alignment possibilities.

The assembler language processor provides facilities that automatically position or allow us to force the required boundary alignments.

Boundary alignment restrictions were designed to force us to place words at consecutive integral boundaries to guarantee efficient machine operation when a program written for one model of System/360 is run on another model.

To illustrate, suppose that we correctly stored a halfword in location 512 and 513 and then incorrectly stored a series of fullwords beginning at location 514 (which is not divisible by 4). When we reference this data on a Model 50, which accesses a fullword on a single storage fetch, here is what would have to happen without boundary restrictions. An instruction that references the halfword at location 512 would also access half of the fullword beginning at location 514. Another storage access would be necessary to reference the other half of the fullword, and each successive fullword access would then fetch only half of the word we are seeking.

Thus, to guarantee efficiency and to maintain program compatibility among the various models, boundaries are identical for each model.

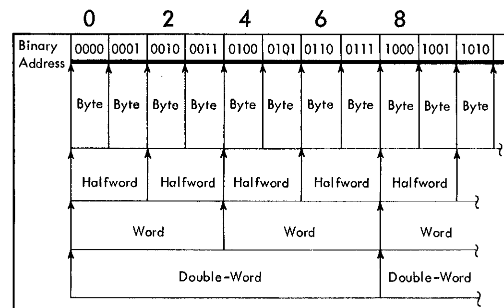


Figure 14. Integral boundaries for halfwords, words, and doublewords

General Registers and Storage Addressing

A set of 16 general purpose registers is standard. General registers can be used as index registers, relocation registers, accumulators for fixed-point arithmetic, and for logical operations.

Only four bits in an instruction are required to designate a register. Each register has a capacity of one 32-bit word.

Before considering the details of how these registers are utilized, it is helpful to see *why* registers were designed as part of the system.

Access time to storage increasingly limits performance as processor speeds improve. Using a single faster-access accumulator decreases overall processing time compared with the time required for storage-to-storage arithmetic. To efficiently utilize the single faster accumulator, however, it is necessary that data be refetched whenever it is reused and that results be stored temporarily for later use. Many of these fetch and store operations can be eliminated when multiple accumulators are available as registers.

Just as multiple registers improve the efficiency of arithmetic and logical operations, they can also provide a means of efficient address specification and modification.

Because the ability to address vast amounts of main storage is a desirable feature, an internal address of 24 binary bits is used. This permits up to 16,777,216 unique bytes to be addressed ($2^{24} = 16{,}777{,}216$).

An instruction, then, that involves a storage address would appear to require 24 bits to address the operand. Instead, instructions that designate a main storage location specify a register. A four-bit field in the instruction allows the specification of one of the registers numbered 0-15 as shown in Figure 15. The low-order 24 bits of this register contain an address referred to as the base address (B). The instruction must also contain a twelve-bit number called the displacement (D), which provides for relative addressing of up to 4095 bytes beyond the base address. The base and displacement are added together to produce an effective address.

Recall now that four bits of the instruction specify a register and twelve bits specify a displacement. With 16 bits we are able to specify a 24-bit address.

In addition to the base register, many System/360 instructions designate another general register called an index register. In these cases, the effective address is calculated by adding together the contents of the base register, the contents of the index register, and the displacement field (see Figure 16).

The contents of all general registers and storage locations participating in the addressing or execution part of an operation remain unchanged, except for the storing of the final result. This permits multiple instructions to reference a register containing the same base or index value.

Economy in instruction length through the use of the base-displacement addressing approach is one advantage of register utilization in addressing. Another significant advantage is the relocation facility provided. Since the instructions of a program reference registers, the contents of these registers can be specified at load time, so that programs and data can be located in main storage almost at will. When the pro-

R Field	Reg. No.	General Registers	Floating-Point Registers
0000	0	← 32 Bits →	← 64 Bits →
0001	1		
0010	2		
0011	3		
0100	4		
0101	5		
0110	6		
0111	7		
1000	8		
1001	9		
1010	10		
1011	11		
1100	12		
1101	13		
1110	14		
1111	15		

Figure 15. General purpose registers

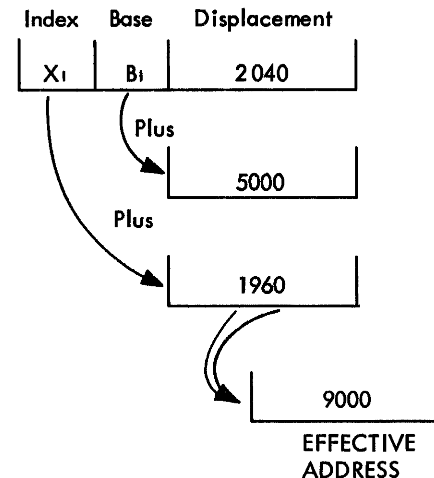


Figure 16. Address generation

gram is to be used at another time, other values can be specified in the base and index registers, so that the program can be executed from another segment of storage.

If, during the processing of a program, it is desirable to use these registers for other purposes, their contents can be stored in core storage. The registers would then be loaded with some other value, and processing continued. Note that the registers must be reloaded with their appropriate base values before executing a segment of the program that assumes the registers contain these values.

This approach of saving the contents of the registers and then restoring them as they are needed removes any limitation problem that might result from the fact that the system has only 15 registers usable for addressing. Register 0 cannot be used for address modification. A specification of 0 in either the base or index of an instruction means no base or index reference. This approach was taken to avoid the waste of having a register permanently filled with 0s when not indexing or when a base of 0 was desired. Certain instructions allow this register to be used as an accumulator, but when 0 is used in the base or index field, the system interprets it as meaning no base or index register.

There are multiple load and multiple store register instructions that make saving and restoration relatively simple operations.

The time spent in storing and restoring registers is quite small when compared with the time saved by having each instruction that references core storage contain only a 16-bit address field rather than a 24-bit address field. Similarly, the space used to preserve the contents of the registers is small compared with the space saved by reducing the instruction length.

Note that when we refer to a "base" or "index" we are referring to the use to which one of the 16 general purpose registers is being put, and not to a specialized register.

General registers are an important aspect of System/360. However, it is not only possible, but normal practice, to delegate to the assembly program almost all the clerical work of assigning base registers and computing displacements. Registers are used for addressing in a variety of ways. Some of the methods used in connection with the assembler language are examined under "Programming with Base Registers and the USING Instruction" in C20-1646.

Relocation has been mentioned as one of the advantages of base-displacement addressing. Let us consider a simple situation in which we benefit from the ability to relocate programs. Assume that programs A and B are to be run together. Program A is located in 2000 consecutive storage locations as shown in Figure 17a. The next 3000 storage locations are occupied by program B. The following 2000 locations are unused, but, except for these locations, we shall consider that no other storage is available.

The next day program C, which requires 4000 bytes of storage, is to be run with program B. After looking at yesterday's storage map, we see that we have only 2000 consecutive locations available (either in the locations previously occupied by program A or in the unused area).

The register used on the previous day to load program B can have its contents modified by a load register instruction, so that today the base value is 2000 bytes higher than yesterday. Upon reloading program B, its starting address and all subsequent addresses will be 2000 positions higher. Thus we have relocated program B, and the last 2000 positions of program B will now occupy the storage segment previously unused. Four thousand consecutive locations are now available for program C, as shown in Figure 17b.

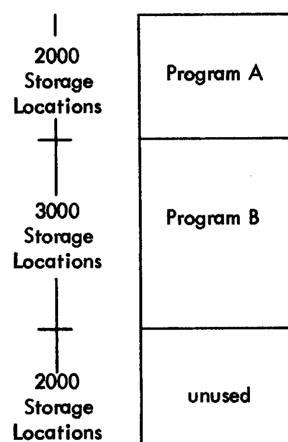


Figure 17a. Consecutive ascending locations in storage when program B is run with program A

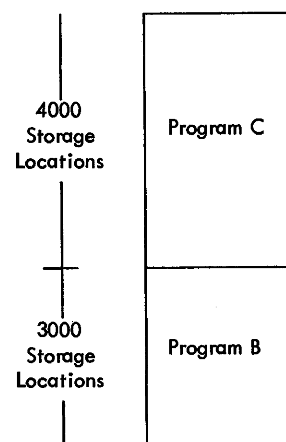


Figure 17b. Consecutive ascending locations in storage after relocation of program B to run with program C

Instruction Formats

We have seen that variable-length fields as well as words can be addressed. Instruction length is also variable. Some instructions cause no reference to main storage; others cause one or more references to main storage. To conserve storage space and save time in instruction execution, instruction length is variable and can be one, two, or three halfwords. Instructions specify the operation to be done and the location of data. Data may be located in main storage, registers, or a combination of the two. Instruction length is related to the number of storage addresses necessary for the operation. As a result, instructions will be of different lengths depending on the location of data. Instructions of different lengths can be arbitrarily combined in the same program.

When both operands are in registers, only eight binary bits are needed for register addresses. Since eight binary bits are used for the operation and eight bits for operands, the shortest instruction consists of one halfword and there is no reference to main storage.

When both operands are in main storage, a total of 32 bits are needed for the addresses (one four-bit base and one twelve-bit displacement for each of the two addresses) and, because the operation code and length specification(s) will require additional bits, the longest instruction (three halfwords in length) is used.

Figure 18 shows the five basic instruction formats. The format codes are RR, RX, RS, SI, and SS, which indicate the general locations of the operand or operands. RR denotes a register-to-register operation; RX, a register-to-indexed storage operation; RS, a register-to-storage operation; SI, a storage and immediate operand operation; and SS, a storage-to-storage operation. An "immediate operand" is a byte of data used as an operand that is carried in the instruction itself.

In the formats shown in Figure 18, R_1 specifies the address of the register containing the first operand. The second operand location, if any, is defined differently for each format.

In the RR format, the R_2 field specifies the address of the general register containing the second operand.

In the RX format, the contents of the general registers specified by the X_2 and B_2 fields are added to the contents of the D_2 field to form an address designating the storage location of the second operand.

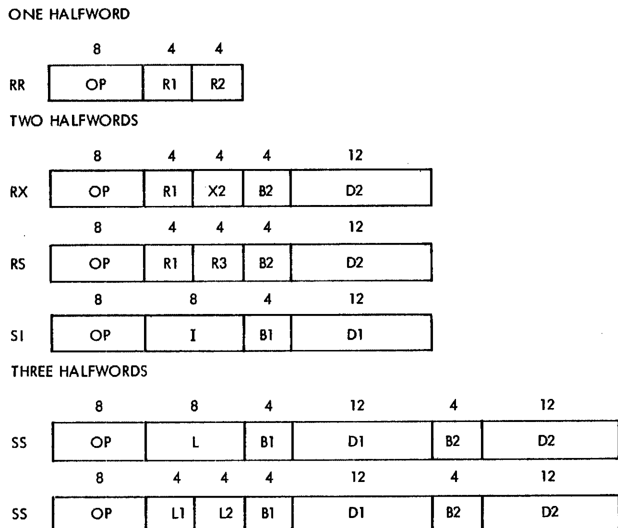


Figure 18. Instruction formats

The symbology employed in the RS format is explained with the example shown below. In shift operations employing the RS format, the designations of fields differ from the example shown, but this does not concern us here.

In most cases the results replace the first operand except for the Store instruction, and the Convert to Decimal instruction, where the result replaces the second operand.

The contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged, except for the storing of the final result.

In the following examples of the instruction formats, the operands are expressed as decimal numbers, and the operation codes are expressed in the symbolic assembly language.

RR Format

OP Code	R_1	R_2
AR	7	9
0 7	8 11	12 15

Execution of this Add instruction adds the contents of general register 9 to the contents of general register 7, and the sum is placed in general register 7.

RX Format

OP Code	R_1	X_2	B_2	D_2
ST	3	10	14	300
0 7	8 11	12 15	16 19	20 31

Execution of this Store instruction stores the contents of general register 3 at a main storage location addressed by the sum of 300 and the low-order 24 bits of general registers 14 and 10.

RS Format

OP Code	R_1	R_3	B_2	D_2
LM	3	9	11	300
0 7	8 11	12 15	16 19	20 31

This Load Multiple instruction causes the set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 to be loaded from the locations designated by the second operand address.

The storage area from which the contents of the general registers are obtained starts at the location designated by the second operand address and continues through as many words as needed. The general registers are loaded in the ascending order of their addresses, starting with the register specified by R_1 and continuing up to and including the register specified by R_3.

It was pointed out earlier that the storing and restoration of registers is a relatively simple matter. There is also a multiple store instruction that provides for the storing of the registers, while this multiple load instruction provides for their restoration.

SI Format

OP Code	I_2	B_1	D_1
MVI	$	12	100
0 7	8 15	16 19	20 31

With this Move Immediate instruction in the example shown, a dollar sign ($) is to be placed in location 2100, leaving locations 2101-2104 unchanged. Let Z represent a four-bit zone. Assume that:

Register 12 contains	00	00	20	00	
Location 2100-2104 (before)	Z0	Z1	Z2	Z3	Z0
Locations 2100-2104 (after)	$	Z1	Z2	Z3	Z0

SS Format

OP Code	L_1	L_2	B_1	D_1	B_2	D_2
AP	4	4	6	64	6	68
0 7	8 11	12 15	16 19	20 31	32 35	36 47

With this Add Decimal instruction, the second operand is added to the first operand, and the sum is placed in the first operand location. If necessary, high-order zeros are supplied for either operand. Note that in the register-to-register (RR) instruction example, the addition is on fixed-length binary fields.

The decimal arithmetic instruction in the SS format operates on data in the packed format with two decimal digits placed in one eight-bit byte. The length of the fields is specified explicitly in the instruction rather than implied in the operation code.

In each format (RR, RX, RS, SI, or SS) the first byte contains the operation code in the binary code, which is the actual machine language. In binary, the length and format of an instruction are specified by the first two bits of the operation code.

BIT POSITION	INSTRUCTION LENGTH	INSTRUCTION FORMAT
00	One halfword	RR
01	Two halfwords	RX
10	Two halfwords	RS or SI
11	Three halfwords	SS

During instruction decoding, the processing unit examines these first two bits of the operation code and determines how many bytes to fetch for this instruction. These bit configurations are part of the machine instruction, so that when, for example, we specify an Add register-to-register instruction, we are not concerned with specifying the instruction length.

We have seen that for fixed-length instructions the length of the *operand* is implicit in the instruction, and for variable-length operands the length is specified in the instruction. We have also seen that the length of the *instruction* itself is part of the operation code.

Protection Features

System/360 was designed for operation with a supervisory program that schedules and governs the execution of multiple programs, handles exceptional conditions, and coordinates and issues input/output instructions.

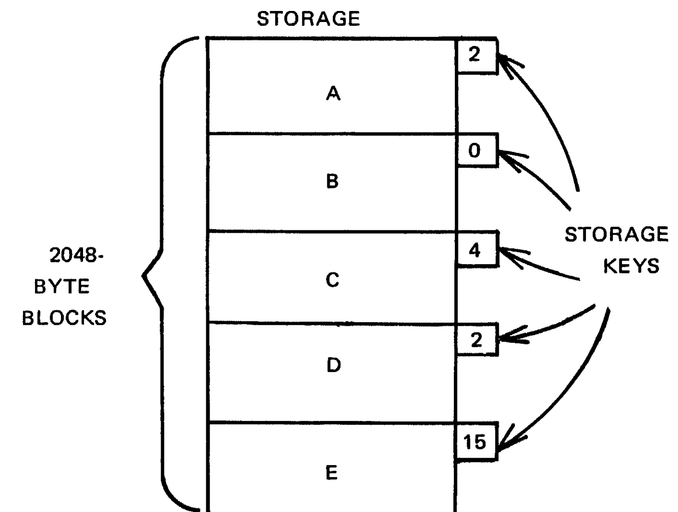
In addition, the computing system and the supervisory programs are designed to prevent one program, such as a problem program, from modifying another program, such as the supervisor program. A means is provided by which the supervisor program can change any area of main storage, while the problem program can change only its own assigned areas. It is desirable for example, that the supervisor program be able to change the main storage locations containing the new program status words. However, we would not want the problem program to be able to modify this same area. It is undesirable to have any part of the supervisor program changeable by the problem program. The feature that prevents data from being brought into a protected area of core, and thus prevents one program from destroying another, is called store protection.

Store protection is an optional feature on some models of the System/360 and is standard on others. It has been pointed out that medium-to-large-scale systems are utilized most efficiently in a multiprogramming environment and that the system is adept at handling more than one program concurrently. In such cases the supervisor program, utilizing the store protection feature, assigns programs to particular areas of storage.

For protection purposes, main storage is divided into blocks of 2048 bytes each. Each 2048-byte block of storage has a five-bit key associated with it, which may be used to establish the right of access. The supervisory program may store any five-bit combination in these keys. (Note that the supervisory program and not the problem program has access to the storage keys.) The same key may be assigned to more than one block and these blocks of 2048 bytes need not be contiguous.

The current PSW, as we have seen, acts as an instruction counter. Another of its functions is to keep track of the protection key of the program with which each instruction is associated. When an instruction attempts to store information in core, the protection key of the current PSW is compared with the high-order four bits of the storage key of the affected block. (The fifth or low-order bit is used only when an additional feature called fetch protection is provided; this will be discussed on the next page.) When storing is specified not by a program instruction but by channel operation, a protection key supplied to the channel from the channel address word (CAW) is similarly compared with the storage key of the area in which the data is to be stored. The CAW is explained later under "Channel Organization." It has already been pointed out, however, that channels have their own programs, and to understand store protection, we should be aware that the protection key in the CAW provides protection on input operations from channels similar to that provided by the PSW on internal operations.

Storage takes place only when the protection key and the storage key match or when the protection key is zero. This is shown in the example given in Figure 19.



When PSW or channel protection key is	Program can store data in storage blocks				
	A	B	C	D	E
2	Yes	No	No	Yes	No
0	Yes	Yes	Yes	Yes	Yes
4	No	No	Yes	No	No
15	No	No	No	No	Yes

Figure 19. Example of store protection

If the PSW, then, contains a nonzero protection key, a store operation will not occur in an area of storage with the zero key. If, on the other hand, the protection key is zero, a store operation can be executed using any area of storage without regard for its storage key. The supervisory program will sometimes require this zero master key in its PSW. The protection key of the current PSW in the problem program cannot be changed by the problem programmer. Thus problem program interference with the supervisory program or with other programs is prevented.

When an instruction causes a protection mismatch, execution of the instruction is suppressed or terminated, and program execution is altered by an interrupt.

Fetch protection, in addition to store protection, is available on some models of the System/360. When store and fetch protection is installed, each 2048-byte block can be protected against the fetching of information from the block as well as the storing of information in the block. The low-order bit of the block's five-bit storage key indicates whether store protection only or store and fetch protection applies to that block. A zero in that bit position indicates that only store protection applies. A one indicates that protection applies to both storing and fetching. The high-order four bits of the storage key are used to determine whether or not there is a protection mismatch. A protection mismatch due to a fetch violation causes the execution of the instruction to be terminated.

Floating-Point Arithmetic

In fixed-point computation the position of digits must be aligned for each operand to express their integral or fractional value. The separation of the integral and fractional portion of a number denoted by a point in written notation is the programmer's responsibility.

Scientific and engineering computations often involve multiplications and divisions where the magnitude of the quantities involved varies from very small fractions to large integers.

To relieve the programmer of the responsibility of shifting to position intermediate and final results, floating-point notation and circuitry to operate upon it have been characteristics of scientific computers. Floating-point arithmetic is an optional feature on Models 30 and 40 and is standard on the higher-performance models.

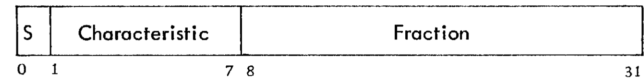
Four 64-bit floating-point registers identified by the numbers 0, 2, 4, and 6 are provided, as shown in Figure 20. The operation code determines whether a general purpose or floating-point register is to be used in an operation. An attempt to execute a floating-point instruction on a system not equipped with the feature will result in a program interrupt.

The notation used for floating-point arithmetic can express decimal values ranging from about 5.4×10^{-79} to about 7.2×10^{75}. It is basically a mathematical shorthand that reduces a number to a fraction and an exponent (or characteristic). Either a short (32-bit) or long (64-bit) format operand may be specified. The short format permits a maximum number of operands to be placed in storage and gives the shortest execution time. The long format is used when greater precision is desired. The formats differ only in the length of the fraction, as shown in Figure 21.

R Field	Reg. No.	General Registers	Floating-Point Registers
0000	0	← 32 Bits →	← 64 Bits →
0001	1		
0010	2		
0011	3		
0100	4		
0101	5		
0110	6		
0111	7		
1000	8		
1001	9		
1010	10		
1011	11		
1100	12		
1101	13		
1110	14		
1111	15		

Figure 20. General and floating-point registers

Short Floating-Point Number



Long Floating-Point Number

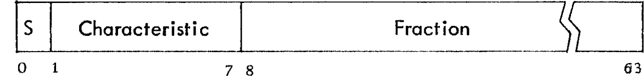


Figure 21. Short and long floating-point number formats

In the section entitled "Channel Concept" mention was made of communications between the processing unit and the channel. We shall now examine in more detail the ways in which the processing unit, the channels, the control units, and the I/O devices communicate with each other.

System/360 is designed for use in conjunction with a supervisor program that allocates equipment to multiple programs and also monitors the execution of each problem program. The supervisor program must also monitor I/O operations. To permit unrelated problem programs to execute I/O operations concurrently, the channel hardware together with the supervisor program provides a means of assigning to each program the required I/O facilities. This assignment consists of establishing a path not only for transferring data between the I/O device and the designated area of main storage, but also for exchanging control and status information between the program and the I/O facility.

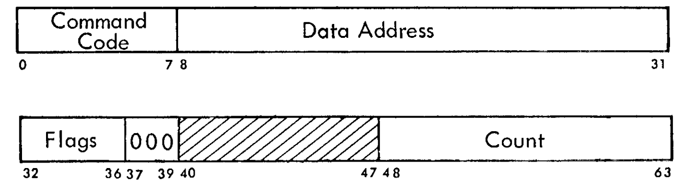
Input/output control units are attached to the channel by a standard connection, called the I/O interface. This interface is common to all channels and control units. It provides an information and signal sequence that is common to all types of I/O control units. The interface has nine one-way lines for input and nine lines for output to accommodate one byte including parity. Other lines carry status and control information. The important thing to remember is that identical lines are used for all control units including those for tape, disk, card, etc. The channel operates the control unit, and the control unit is designed to meet the interface requirements.

The control unit operates the actual device. Examples of control units are tape control, communications control, card control, and printer control. The channel, in turn, operates the control unit. The processing unit controls channel activity by means of four *instructions*:

Start I/O
Test I/O
Halt I/O
Test Channel

Commands constitute the channel program. The channel programs are held in main storage until an I/O operation is initiated by a Start I/O instruction. A channel address word (CAW) is permanently assigned to contain the address of the initial channel command word (CCW) (see Figures 22, 23, and 24). CCW's are decoded by the channel, which issues *orders* to the I/O device.

Figure 22. Channel address word format



Bits 0-7 specify the command code.
Bits 8-31 specify the location of a byte in main storage.
Bits 32-36 are flag bits.
Bit 32 causes the address portion of the next CCW to be used.
Bit 33 causes the command code and data address in the next CCW to be used.
Bit 34 causes a possible incorrect length indication to be suppressed.
Bit 35 suppresses the transfer of information to main storage.
Bit 36 causes an interruption as Program Control Interrupt.
Bits 37-39 must contain zeros.
Bits 40-47 are ignored
Bits 48-63 specify the number of bytes in the operation.

Figure 23. Channel command word format

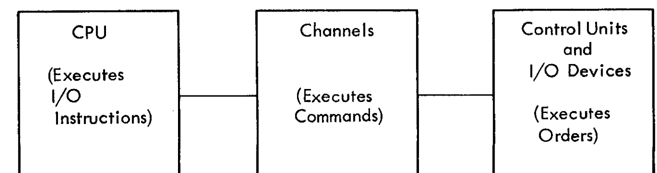


Figure 24. Relationship of I/O instructions, commands, and orders

The CCW contains the command to be executed, and for commands that initiate I/O operations it designates the storage area associated with the operation and the action to be taken whenever transfer to or from the area is completed. The CCW's can be located anywhere in main storage on doubleword boundaries, and more than one can be associated with a Start I/O instruction. The channel refers to a CCW in main storage only once, whereupon the pertinent information is stored in the channel.

The first CCW is fetched during the execution of Start I/O. Each additional CCW in the chain is obtained when the operation has progressed to the point where the additional CCW is needed.

The CCW has the format shown in Figure 23.

Bits 0-7 specify the operation to be performed. There are six valid commands:

Sense
Transfer in Channel
Read Backward
Write
Read
Control

The data address specifies the location of an eight-bit byte in main storage. It is the first location referred to in the area designated in the CCW.

The count specifies the number of eight-bit byte locations beyond the initial byte designated by the address.

It has been mentioned that channels function much like small independent computers. As such they contain registers. Bits 32 through 36 of the CCW are labeled "flags" (see Figure 23). The channel registers include a flag register that indicates command modes. These flags serve to chain data or commands for this series of CCW's, interrupt the processing unit, skip a portion of a record, suppress length indication, or terminate the operation.

These flags may be set on or off in each of the channel control words and the flag register is updated with each new CCW. Other registers within the channel circuitry are (1) a command counter, which tells the channel where to get the next command in a manner similar to that of an instruction counter in a processing unit, (2) a command register, which tells the channel which command is to be performed, (3) an address register, which tells the channel where to get or put data into core storage, (4) a count register, which indicates how many characters are to be read or written, and (5) a key register, which contains the protection key for the current operation.

The generalized CCW commands listed earlier apply to all devices. Read, Write, and Read Backward are self-explanatory. The Sense command is a request to the I/O control unit for device-dependent status information, such as the position of magnetic tape, the condition of the card stacker and hopper, or the detailed conditions detected in the last operation. This status information is transferred to the channel as data and is placed in the main storage area designated by the CCW.

Normally the detailed information provided by the sense command is not required, and an eight-bit status byte is provided to the channel (upon completion of an I/O operation) indicating the general conditions detected during the operation. This status byte is common to all I/O devices and cannot convey the detail conditions of termination provided by the sense command.

A control command causes the control unit to initiate at the I/O device an operation not involving the transfer of data – such as backspacing or rewinding magnetic tape, or positioning a disk access mechanism.

The Transfer in Channel command causes the next CCW to be fetched from the location designated by the data address field of this command instead of fetching the next sequential CCW. In effect, then, the Transfer in Channel command causes a branch from one sequence of CCW's to another.

When command chaining is specified by a flag bit in the CCW, the channel uses the new CCW to initiate a new operation at the device and permits the processor program to start with a single I/O instruction such sequences as printing multiple lines or reading multiple tape blocks. With command chaining it is possible for the channel to execute I/O programs of any number of I/O operations.

When data chaining is specified by a flag bit in the CCW, the channel uses the new CCW to designate another data area for the original I/O operation and the device continues to execute this operation. Only the allocation of storage areas is affected. Data chaining permits the reorganization of information as it is transferred between main storage and the I/O device.

The proper use of the available channel command words permits the following types of I/O functions:

Scatter-read – reading one physical record into multiple, noncontiguous areas of storage.

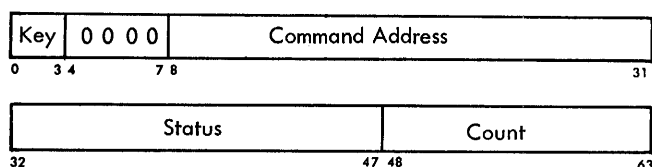
Extraction – reading only selected portions of a record into storage.

Control nondata I/O operations – for example, backspace, rewind, etc.

Command chaining – for sequentially performing operations on the same device, for example, reading over an interrecord gap.

Upon completion of the channel program, an I/O interrupt occurs; that is, the channel interrupts the processing unit. The channel makes available in main storage a channel status word (CSW). This doubleword contains an address that is eight bytes higher than the address of the last CCW used, and indicates in the count field the difference between the count in the last CCW and the amount of data transferred. The format of the channel status word is shown in Figure 25. The protection key is the key used in the operation. It is first supplied to the channel from the CAW as a result of a Start I/O instruction.

Bits 32-47 of the channel status word contain an eight-bit I/O device-status byte and a channel status byte. These two bytes provide such information as data-check, chaining check, and control unit end. The channel status word has a permanent storage assignment of locations 64 through 71 in main storage as shown in Figure 4.



Bits 0-3 contain the protection key used in the operation.
Bits 4-7 contain zeros.
Bits 8-31 specify the address plus 8 of the last CCW used.
Bits 32-47 contain an I/O device-status byte and a channel-status byte. The status bytes provide such information as data-check, chaining check, control-unit end, etc.
Bits 48-63 contain the residual count of the last CCW used.

Figure 25. Channel status word format

With the command address, status, and count fields of the CSW, the program can determine the status of an I/O device or the conditions under which an I/O operation has been terminated.

The processing unit's program depends on I/O interrupts for information concerning the progress of I/O operations. So that the processor program can tell in advance when conditions in the channel or in the device should alert the program, a mask bit is associated with each channel. A masked channel cannot cause an I/O interrupt, and consequently the supervisor program can suppress I/O interrupts by masking the channels. The conditions in the channels and devices are preserved until accepted by the processor program. The program can determine whether an interrupt condition is pending in the channel by issuing the instruction Test Channel.

Channel masking allows the processor program to accept I/O interrupts selectively by channel. However, on a given channel more than one I/O control unit can contain pending conditions that cause program interruption. The instruction Test I/O allows a program to accept interrupts selectively by I/O device. This instruction gives the program the status of the designated device and clears any interrupt condition pending in the device. Test I/O provides the same information as an I/O interrupt, since the channel status word is stored. Keeping the channels masked and interrogating devices by the Test I/O instruction prevents the program from being interfered with by conditions unrelated to the program being run.

In a real-time or communications environment, on the other hand, the processor program would keep all channels unmasked and depend on I/O interrupts for information concerning the progress of I/O events as they occur.

Summary

System/360 includes provisions for large storage capacity, simple program relocation, flexible protection, and general supervisory facilities. Provisions are also included for a variety of data formats, an extensive set of processing operations, and machine language compatibility among the various models.

To compensate for higher computational speeds relative to human reaction time, and to adapt the system to online and real-time multiprogramming tasks, the system is more highly automated by having the system resources controlled by a supervisory program. Provision for this control is embodied in these concepts:

- Supervisory mode with associated privileged instructions
- Storage protection
- Hardware monitoring
- The ability to perform interrupts
- A wait state available to the supervisor program, rather than a stop or halt instruction available to the problem programmer.

1. Can a tape unit be attached to a multiplexor channel?
2. If the problem program issues a Load PSW instruction to cause the new I/O PSW to be loaded, can the problem program cause an I/O operation to be executed?
3. The instruction address contained in the New Supervisor Call PSW addresses a routine to handle this class of interrupts. What action must this routine first take?
4. A program interrupt will occur if the Convert to Binary instruction attempts to operate upon data that contains invalid codes for packed decimal. What are the ten valid four-bit codes for packed decimal?
5. Is data punched in an IBM card as Hollerith code acceptable as input to a System/360 equipped with a card reader?
6. If floating-point arithmetic is intended for scientific and engineering applications, while the decimal instruction set is primarily for commercial applications, by whom are fixed-point arithmetic instructions used?
7. Where is the sign of a number in binary, EBCDIC, and packed decimal format located?
8. What storage location is addressed by an instruction with zeros in the index and displacement fields and the number 5 in the base register field?
9. Does the programmer select a particular instruction length?
10. If loading into storage the following:

Address	*Constant*
100	Fullword
104	Halfword
---	Fullword

at what address would the last fullword be loaded?

11. If the current PSW contains a protection key of zero and the instruction is to store data in Area A, which has a storage key of three, would a program interrupt occur?

Answers to Questions and Exercises

1. Yes. Data from a tape unit may be transmitted over a multiplexor channel, in which case the channel operates in burst mode. Like other I/O devices, tape units are attached to a control unit, which, in turn, is attached to a channel.

2. No. The Load PSW instruction is a privileged instruction, and an attempt to execute this instruction by the problem program will cause a program interrupt.

3. Because the Supervisor Call instruction contains an eight-bit code that is stored in the old supervisor call PSW in the course of interruption, the routine must first examine this code in the old PSW. The code may be regarded as a message conveyed by the instruction to the supervisor.

4. The valid packed decimal digit codes are:

   0000 0001 0010 0011 0100
   0101 0110 0111 1000 1001

which represent the digits 0-9.

5. Yes. Hollerith code is read by a card reader and transferred from the card reader's control unit as EBCDIC.

6. Fixed-point arithmetic instructions are part of the standard instruction set. Neither the optional decimal nor floating-point instruction set is sufficient in itself to perform processing.

7. A binary quantity is represented internally by a 32-bit binary number. The sign occupies the high-order (leftmost) bit position. The sign of a number in EBCDIC occupies the zone position of the least significant digit. The sign of a packed decimal number occupies the low-order four bits of the field.

8. The effective address is specified by the 24 least significant bits in register 5.

9. No. Length is not a criterion for the selection of instructions. The programmer knows the location (in storage, registers, or both) of data to be operated upon and the operations to be performed. His selection is made accordingly, and halfword, word, and three-halfword instructions are mixed within a program.

10. The fullword would be loaded at location 108; locations 106 and 107 would contain slack bytes.

11. No. When the protection key is zero, a store operation can be executed using any area of storage without regard to its storage key.

READER'S COMMENT FORM

Student Text — Introduction to
IBM System/360 Architecture

GC20-1667-1

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM.

COMMENTS

fold

fold

fold

fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
  FOLD ON TWO LINES, STAPLE AND MAIL.

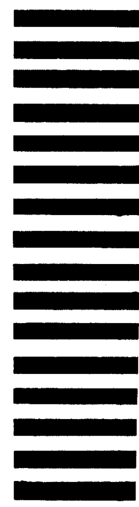
YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.



BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Technical Publications

fold fold

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]