File No. S360-01
Order No. GA22-6821-8

IBM Systems Reference Library

IBM System/360 Principles of Operation

This publication is the machine reference manual for the IBM System/360. It provides a direct, comprehensive description of the system structure; of the arithmetic, logical, branching, status switching, and input/output operations; and of the interruption system.

The reader is assumed to have a basic knowledge of data processing systems and to have read the *IBM System/360 System Summary,* Form A22-6810, which describes the system briefly and discusses the input/output devices available.

For information about the characteristics, functions, and features of a specific System/360 model, use the functional characteristics manual for that model in conjunction with the *IBM System/360 Principles of Operation.* Descriptions of specific input/output devices used with the System/360 appear in separate publications. Publications that relate to the IBM System/360 Model 20 are described in the *IBM System/360 Model 20 Bibliography,* Form A26-3565. Other IBM Systems Reference Library publications concerning the System/360 are identified and described in the *IBM System/360 Bibliography,* Form A22-6822.

Ninth Edition (November 1970)
This is a reprint of GA22-6821-7 incorporating changes released in the following Technical Newsletters:

GN22-0354 (dated May 12, 1970)
GN22-0361 (dated June 8, 1970)

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, refer to the latest System/360 SRL Newsletter, GN20-0360, for the editions that are applicable and current.

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

This manual has been prepared by the Systems Development Division, Product Publications, Dept. B98, PO Box 390, Poughkeepsie, New York 12602. A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be sent to the above address.

Contents

The IBM System/360 is a solid-state, program compatible, data processing system providing the speed, precision, and data manipulating versatility demanded by the challenge of commerce, science, and industry. System/360, with advanced logical design implemented by microminiature technology, provides a new dimension of performance, flexibility, and reliability. This dimension makes possible a new, more efficient systems approach to all areas of information processing, with economy of implementation and ease of use. System/360 is a single, coordinated set of new data processing equipment intended to replace old logical structures with an advanced creative design for present and future application.

The logical design of System/360 permits efficient use at several levels of performance with the preservation of upward and downward program compatibility. In addition, extremely high performance and reliability requirements may be met by using the multisystem feature to combine several models into one multisystem.

General-Purpose Design

System/360 is a general-purpose system that may be tailored readily for commercial, scientific, communications, or control applications. A *Standard* instruction set provides the basic computing function of the system. To this set a decimal feature may be added to provide a *Commercial* instruction set or a floating-point feature may be added to provide a *Scientific* instruction set. When the instructions associated with storage protection are added to the commercial and scientific features, a *Universal* instruction set is obtained. Timer and direct-control features may be used with systems to support time-sharing or real-time operations, and in teleprocessing applications.

System/360 is designed to accommodate large quantities of addressable storage. The markedly increased capacities over previous storage are provided by the combined use of high-speed storage of medium size and large-capacity storage of medium speed. Thus, the requirements for both performance and size are satisfied in one system by the availability of different types of storage units. Also, the design makes provision for development, in the future, of even greater storage capacities.

Another aspect of the general-purpose design of System/360 is its standard-interface method for attaching all input/output devices. Future input/output devices will also attach to this input/ouput interface which is common to all System/360 channels.

Models of System/360 differ in storage speed, storage width (the amount of data obtained in each storage access), register width, and capabilities for processing data concurrently with the operation of multiple input/output devices. Several CPU's permit a wide choice in internal performance. Yet none of these differences affect the logical appearance of these models to the programmer.

An individual System/360 is obtained by selecting the system components most suited to the applications from a wide variety of alternatives in internal performance, functional ability, and input/output (I/O).

Compatibility

All models of System/360 are upward and downward compatible; that is, any program gives identical results on any model. Compatibility allows for ease in systems growth, convenience in systems backup, and simplicity in education. The compatibility rule has three limitations.

1. The systems facilities used by a program should be the same in each case. For example, the optional CPU features and the storage capacity, as well as the quantity, type, and priority of I/O equipment, should be equivalent.

2. The program should be independent of the relation of instruction execution times and of I/O data rates, access times, and command execution times.

3. The compatibility rule does not apply to detail functions for which neither frequency of occurrence nor usefulness of result warrants identical action in all models. These functions, all explicitly identified in this manual, are concerned with the handling of invalid programs and machine malfunctions.

System Program

Interplay of equipment and program is an essential consideration in System/360. The system is designed to operate with a supervisory program that coordi-

nates and executes all I/O instructions, handles exceptional conditions, and supervises scheduling and execution of multiple programs. System/360 provides for efficient switching from one program to another, as well as for the relocation of programs in storage. To the problem programmer, the supervisory program and the equipment are indistinguishable.

IBM provides System/360 programs that control and schedule the use of CPU facilities, main storage, storage devices attached to channels, input/output devices, etc. These programs are designed to control all system resources, including programs supplied by the customer and by IBM.

System Alerts

The interruption system permits the CPU to respond automatically to conditions arising outside of the system, in I/O units, or in the CPU itself. Interruption switches the CPU from one program to another by changing not only the instruction address but all essential machine-status information.

Protection features permit one program to be preserved when another program erroneously attempts to gain access to information in the protected storage area. Protection does not cause any loss of performance. Storage operations initiated from the CPU, as well as those initiated from a channel, are subject to the protection procedure.

Programs are checked for correctness of instructions and data as the instructions are executed. This policing-action distinguishes and identifies program errors and machine errors. Thus, program errors cannot cause machine checks: each of these types of error causes a different type of interruption. When an interruption due to machine malfunction occurs, the information necessary to identify the error is recorded automatically in a predetermined storage area. This logging of pertinent information can be used to assist in the analysis of machine faults. Moreover, operator errors are reduced by minimizing the number of manual controls and the need for their use. To reduce accidental operator errors, operator consoles are basically I/O devices that function under control of the system program.

Multisystem Operation

Several models of System/360 can be combined into one multisystem configuration. Three types of communication between CPU's are available. Largest in capacity, and moderately fast in response, is communications by means of a shared I/O device, such as a disk file. Faster data transfer may be obtained by direct connection between the channels of two individual systems. Finally, some models permit sharing of storage between CPU's, making information exchange possible at storage speeds. These types of communication are supplemented by allowing one CPU to be interrupted by another CPU and by making status information directly available from one CPU to another.

Input/Output

Channels provide the data path and control for I/O devices as they communicate with the CPU. In general, channels operate asynchronously with the CPU and, in some cases, a single data path is made up of several subchannels. When this is the case, the single data path is shared by several low-speed devices, such as card readers, punches, printers, and terminals, each on a separate subchannel. This type of channel is called a *multiplexor* channel. Another type of channel, the *selector* channel accommodates higher data rates, but can be involved in only one data transfer operation at a time.

In every case, the amount of data that comes into the channel in parallel from an I/O device is a byte (i.e., eight bits). All channels or subchannels perform the same functions and respond to a common set of I/O instructions and commands.

Each I/O device is connected to one or more channels by an I/O interface. This I/O interface allows attachment of present and future I/O devices without alteration of the I/O instruction set or of channel functions. Control units are used where necessary to match the internal connections of the I/O device to the interface. Flexibility is enhanced by optional access to a control unit or device from either of two channels.

Technology

System/360 employs solid-logic integrated components, which in themselves provide advanced equipment reliability. These components are smaller than previous components, operate faster, and lend themselves to automated fabrication. The design of System/360, however, is not dependent upon, or limited to, any particular type of technology. System/360 is free to take continuing advantage of new advances in technology.

The basic structure of a System/360 consists of main storage, a central processing unit (CPU), the selector and multiplexor channels, and the input/output devices attached to the channels through control units. It is possible for systems to communicate with each other by means of shared I/O devices, a channel, or shared storage. Figure 1 shows the basic organization of a single system.

Main Storage

Storage units may be either physically integrated with the CPU or constructed as stand-alone units. The storage cycle speed is not directly related to the internal cycling of the CPU, thereby permitting an efficient relationship of CPU speed to storage width. The physical differences in the various main-storage units do not affect the logical structure of the system.

Main storage may be shared by CPU's. Fetching and storing of data by the CPU are not affected by any concurrent I/O data transfer or by reference to the same storage location by another CPU. If a CPU and a channel concurrently refer to the same storage location, the accesses normally are granted in a sequence that assigns higher priority to references by channels. If the first reference changes the contents of the location, any subsequent storage fetches obtain the new contents.

Instructions that involve fetching and subsequently storing of data do not necessarily take the storage cycles contiguously, and it is possible for a channel or another CPU to take one or more intervening cycles. When two CPU's concurrently cause the contents of the same location to be updated, such interleaving may cause the information stored in one of the accesses to be lost or the results to be meaningless.

For example, if two CPU's attempt to update information at the same location by an instruction that causes fetching and subsequently storing of the updated data at the same location, it is possible for both CPU's to fetch the data and subsequently for both CPU's to take the store cycles. The change made by the first CPU to store the result in such case is lost. Only the instruction TEST AND SET takes the fetch and store cycles without permitting a channel or another CPU to interleave a cycle.

The contents of main storage are preserved when power is turned on. Turning power off does not affect the contents of main storage if the CPU is in the stopped state. The contents of the keys in storage associated with the protection feature are not necessarily preserved when the main-storage power is turned off.

Information Formats

The system transmits information between main storage and the CPU in units of eight bits, or a multiple of eight bits at a time. Each eight-bit unit of information is called a *byte*, the basic building block of all formats. A ninth bit, the parity or check bit, is trans-

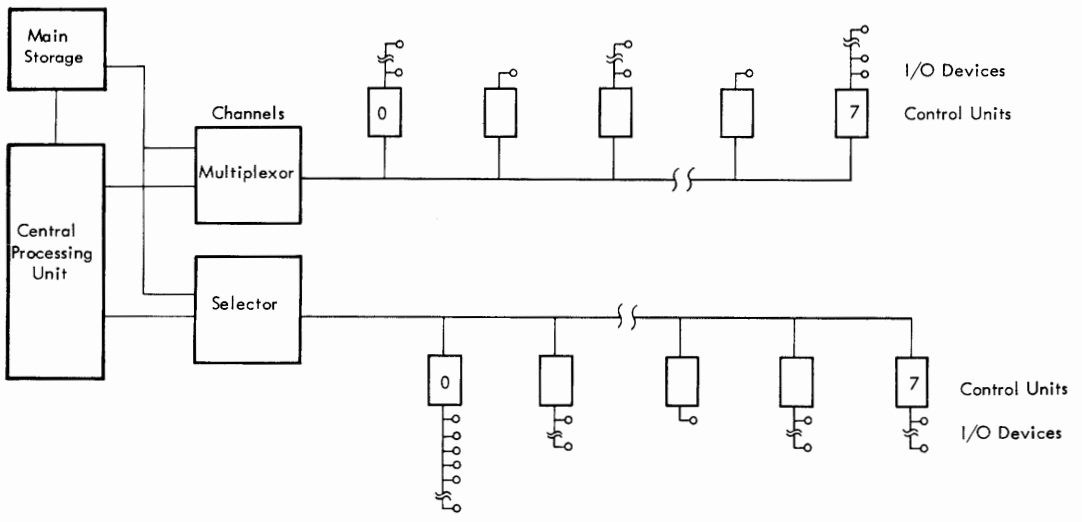


Figure 1. IBM System/360 Basic Logical Structure

mitted with each byte and carries odd parity on the byte. The parity bit cannot be affected by the program; its only purpose is to cause an interruption when a parity error is detected. References in this manual to the size of data fields and registers exclude the mention of the associated parity bits. All storage capacities are expressed in number of bytes provided, without regard to storage width.

Bytes may be handled separately or grouped together in fields. A *halfword* is a group of two consecutive bytes and is the basic building block of instructions. A *word* is a group of four consecutive bytes; a *double word* is a field consisting of two words (Figure 2). The location of any field or group of bytes is specified by the address of its leftmost byte.

The length of fields is either implied by the operation to be performed or stated explicitly as part of the instruction. When the length is implied, the information is said to have a fixed length, which can be either one, two, four, or eight bytes.

When the length of a field is not implied by the operation code, but is stated explicitly, the information is said to have variable field length. This length can be varied in one-byte increments.

Within any program format or any fixed-length operand format, the bits making up the format are consecutively numbered from left to right starting with the number 0.

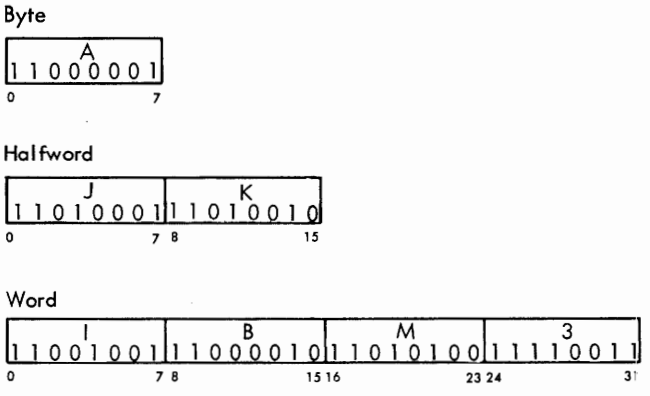


Figure 2. Sample Information Formats

Addressing

Byte locations in storage are consecutively numbered starting with 0; each number is considered the address of the corresponding byte. A group of bytes in storage is addressed by the leftmost byte of the group. The number of bytes in the group is either implied or explicitly defined by the operation. The addressing arrangement uses a 24-bit binary address to accommodate a maximum of 16,777,216 byte addresses. This set of main-storage addresses includes some locations reserved for special purposes.

Storage addressing wraps around from the maximum byte address, 16,777,215, to address 0. Variable-length operands may be located partially in the last and partially in the first location of storage, and are processed without any special indication of crossing the maximum address boundary.

When only a part of the maximum storage capacity is available in a given installation, the available storage is normally contiguously addressable, starting at address 0. An addressing exception is recognized when any part of an operand is located beyond the maximum available capacity of an installation. Except for a few instructions, the addressing exception is recognized only when the data are actually used and not when the operation is completed before using the data. The addressing exception causes a program interruption.

In some models main storage may be shared by more than one CPU. In that case, the address of a byte location is normally the same for each CPU.

Information Positioning

Fixed-length fields, such as halfwords and double words, must be located in main storage on an *integral boundary* for that unit of information. A boundary is called integral for a unit of information when its storage address is a multiple of the length of the unit in bytes. For example, words (four bytes) must be located in storage so that their address is a multiple of the number 4. A halfword (two bytes) must have an address that is a multiple of the number 2, and double words (eight bytes) must have an address that is a multiple of the number 8.

Storage addresses are expressed in binary form. In binary, integral boundaries for halfwords, words, and double words can be specified only by the binary addresses in which one, two, or three of the low-order bits, respectively, are zero (Figure 3). For example, the integral boundary for a word is a binary address in which the two low-order positions are zero.

Variable-length fields are not limited to integral boundaries, and may start on any byte location.

Note: When the byte-oriented operand feature is installed, certain boundary alignment restrictions do not apply. Refer to the description of the feature elsewhere in this section.

Central Processing Unit

The central processing unit (Figure 4) contains the facilities for addressing main storage, for fetching or storing information, for arithmetic and logical proc-

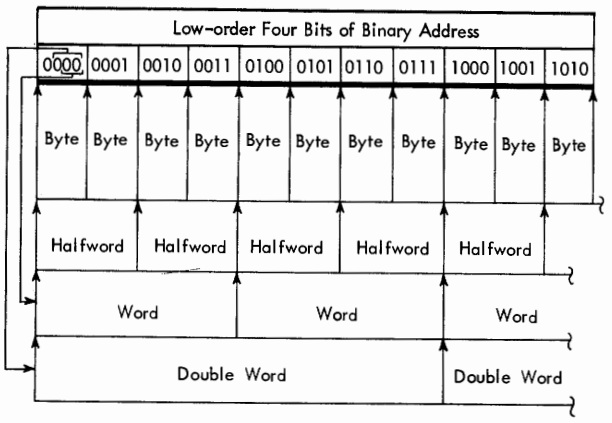


Figure 3. Integral Boundaries for Halfwords, Words, and Double Words

essing of data, for sequencing instructions in the desired order, and for initiating the communication between storage and external devices.

The system control section provides the normal CPU control that guides the CPU through the functions necessary to execute the instructions. While the physical make-up of the control section in the various models of the System/360 may be different, the logical function remains the same. The result of executing a valid instruction is the same for each model.

The CPU provides 16 *general registers* for fixed-point operands and four *floating-point registers* for floating-point operands. Implementation of these registers may be in special circuitry, in a local storage unit, or in a separate area of main storage. In each case, the address and functions of these registers are identical.

General Registers

The CPU can address information in 16 general registers. The general registers can be used as index registers, in address arithmetic and indexing, and as accumulators in fixed-point arithmetic and logical operations. The registers have a capacity of one word (32 bits). The general registers are identified by numbers 0-15 and are specified by a four-bit R field in an instruction (Figure 5). Some instructions provide for addressing multiple general registers by having several R fields.

For some operations, two adjacent general registers are coupled together, providing a two-word capacity. In these operations, the addressed register contains the high-order operand bits and must have an even address, and the implied register, containing the low-order operand bits, has the next higher address.

Floating-Point Registers

Four floating-point registers are available for floating-point operations. They are identified by the numbers 0, 2, 4, and 6 (Figure 5). These floating-point registers

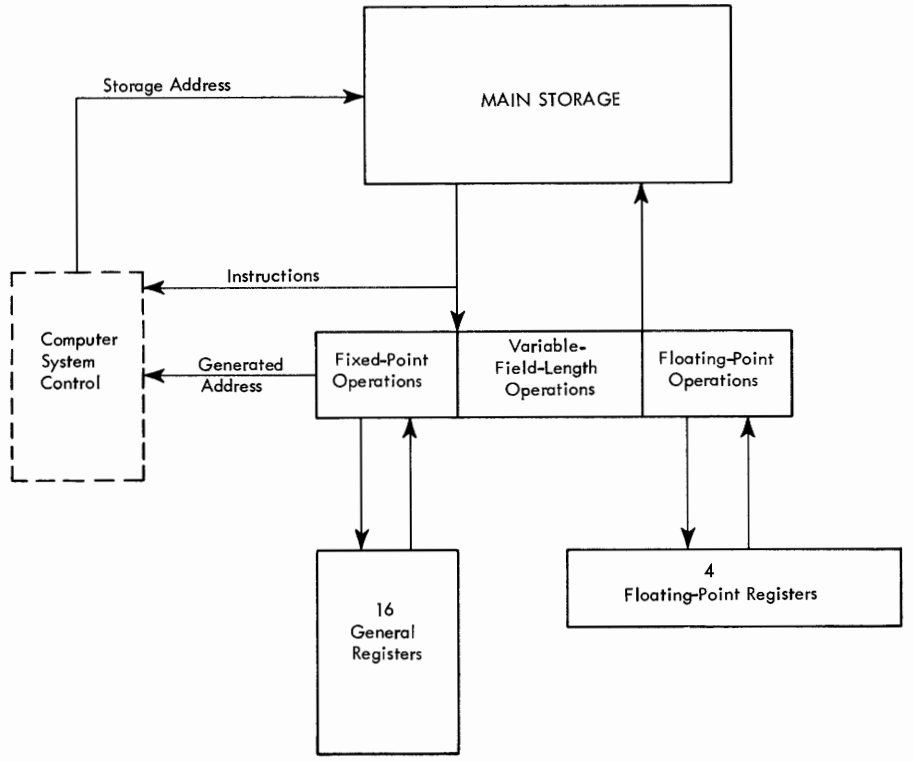


Figure 4. Basic Concept of Central Processing Unit Functions

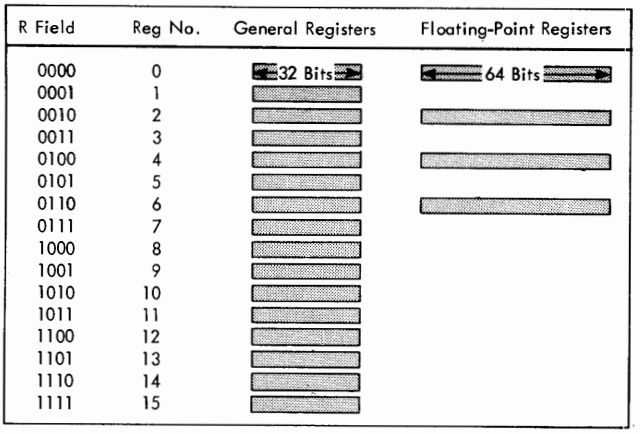


Figure 5. General and Floating-Point Registers

are two words (64 bits) in length and can contain either a short (one word) or a long (two words) floating-point operand. A short operand occupies the high-order bits of a floating-point register. The low-order portion of the register is ignored and remains unchanged in short-precision arithmetic. The instruction operation code determines which type of register (general or floating-point) is to be used in an operation.

Arithmetic and Logical Unit

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

Arithmetic and logical operations performed by the CPU fall into four classes: fixed-point arithmetic, decimal arithmetic, floating-point arithmetic, and logical operations. These classes differ in the data formats used, the registers involved, the operations provided, and the way the field length is stated.

Fixed-Point Arithmetic

The basic arithmetic operand is the 32-bit fixed-point binary word. Sixteen-bit halfword operands may be specified in most operations for improved performance or storage utilization. See Figure 6. To preserve

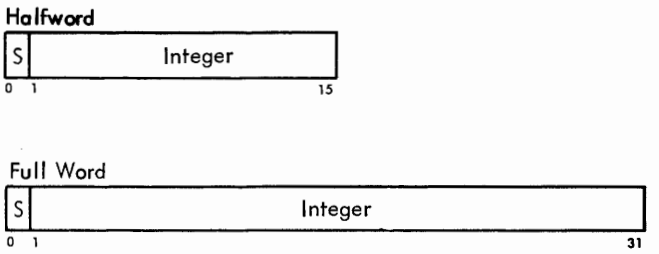


Figure 6. Fixed-Point Number Formats

precision, some products and all dividends are 64 bits long.

Because the 32-bit word size readily accommodates a 24-bit address, fixed-point arithmetic can be used both for integer operand arithmetic and for address arithmetic. This combined usage provides economy and permits the entire fixed-point instruction set and several logical operations to be used in address computation. Thus, multiplication, shifting, and logical manipulation of address components are possible.

Additions, subtractions, multiplications, divisions, and comparisons are performed upon one operand in a register and another operand either in a register or from storage. Multiple-precision operation is made convenient by the two's-complement notation and by recognition of the carry from one word to another. A word in one register or a double word in a pair of adjacent registers may be shifted left or right. A pair of conversion instructions — CONVERT TO BINARY and CONVERT TO DECIMAL — provides transition between decimal and binary radix (number base) without the use of tables. Multiple-register loading and storing instructions facilitate subroutine switching.

Decimal Arithmetic

Decimal arithmetic lends itself to data processing procedures that require few computational steps between the source input and the documented output. This type of processing is frequently found in commercial applications, particularly when use is made of problem-oriented languages. Because of the limited number of arithmetic operations performed on each item of data, radix conversion from decimal to binary and back to decimal is not justified, and the use of registers for intermediate results yields no advantage over storage-to-storage processing. Hence, decimal arithmetic is provided, and both operands and results are located in storage. Decimal arithmetic includes addition, subtraction, multiplication, division, and comparison.

Decimal numbers are treated as signed integers with a variable-field-length format from one to 16 bytes long. Negative numbers are carried in true form.

The decimal digits 0-9 are represented in the four-bit binary-coded-decimal form by 0000-1001, respectively (Figure 7). The codes 1010-1111 are not valid as digits and are reserved for sign codes; 1011 and 1101 represent a minus; the other four codes are interpreted as plus. The sign codes generated in decimal arithmetic depend upon the character set preferred (Figure 7). When the extended binary-coded-decimal interchange code (EBCDIC) is preferred, the codes are 1100 and 1101. When the USASCII set, expanded to eight bits, is preferred, the codes are 1010 and 1011. The choice between the two code sets is determined by a mode bit.

Decimal operands and results are represented by four-bit binary-coded-decimal digits packed two to a byte. They appear in fields of variable length and are accompanied by a sign in the rightmost four bits of the

Digit Code		Sign Code	
0	0000	+	1010
1	0001	-	1011
2	0010	+	1100
3	0011	-	1101
4	0100	+	1110
5	0101	+	1111
6	0110		
7	0111		
8	1000		
9	1001		

Figure 7. Bit Codes for Digits and Signs

low-order byte. Operand fields may be located on any byte boundary, and may have length up to 31 digits and sign. Operands participating in an operation may have different lengths. Packing of digits within a byte (Figure 8) and of variable-length fields within storage results in efficient use of storage, in increased arithmetic performance, and in an improved rate of data transmission between storage and files.

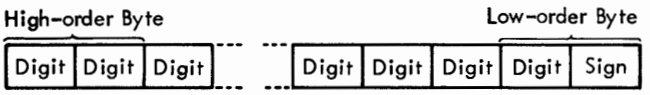


Figure 8. Packed Decimal Number Format

Decimal numbers may also appear in a zoned format as a subset of the eight-bit alphameric character set (Figure 9). This representation is required for character-set sensitive I/O devices. A zoned format number carries its sign in the leftmost four bits of the low-order byte. The zoned format is not used in decimal arithmetic operations. Instructions are provided for packing and unpacking decimal numbers so that they may be changed from the zoned to the packed format and vice versa.

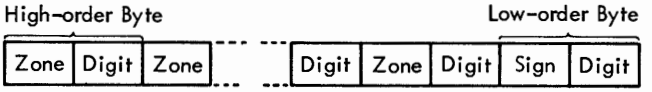


Figure 9. Zoned Decimal Number Format

Floating-Point Arithmetic

Floating-point numbers occur in either of two fixed-length formats — short or long. These formats differ only in the length of the fractions (Figure 10).

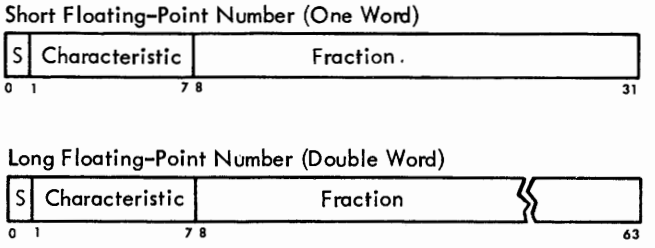


Figure 10. Short and Long Floating-Point Number Formats

Floating-point operands are either 32 or 64 bits long. The short length, equivalent to seven decimal places of precision, permits a maximum number of operands to be placed in storage and gives the shortest execution times. The long length, used when higher precision is desired, gives up to 17 decimal places of precision, thus eliminating most requirements for double-precision arithmetic.

The operand lengths, being powers of two, permit maximum efficiency in the use of binary addressing and in matching the physical word sizes of the different models. Floating-point arithmetic is designed to allow easy transition between the two formats.

The fraction of a floating-point number is expressed in hexadecimal (base 16) digits, each consisting of four binary bits and having the values 0-15. In the short format, the fraction consists of six hexadecimal digits occupying bits 8-31. In the long format the fraction has 14 hexadecimal digits occupying bits 8-63.

The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for the floating-point number, the fraction is considered to be multiplied by a power of 16. The characteristic portion, bits 1-7 of both formats, is used to indicate this power. The characteristic is treated as an excess 64 number

with a range from −64 through +63, and permits representation of decimal numbers with magnitudes in the range of 10^{-78} to 10^{75}.

Bit position 0 in either format is the sign (S) of the fraction. The fraction of negative numbers is carried in true form.

Four 64-bit floating-point registers are provided. Arithmetic operations are performed with one operand in a register and another either in a register or from storage. The result, developed in a register, is generally of the same length as the operands. The availability of several floating-point registers eliminates much storing and loading of intermediate results.

Logical Operations

Logical information is handled as fixed- or variable-length data. It is subject to such operations as comparison, translation, editing, bit testing, and bit setting.

When used as a fixed-length operand, logical information can consist of either one, four, or eight bytes and is processed in the general registers (Figure 11).

A large portion of logical information consists of alphabetic or numeric character codes, called *alphameric data,* and is used for communication with character-set sensitive I/O devices. This information has the variable-field-length format and can consist of up to 256 bytes (Figure 12). It is processed storage to storage, left to right, an eight-bit byte at a time.

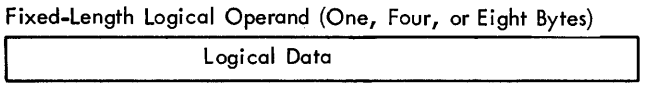


Figure 11. Fixed-Length Logical Information

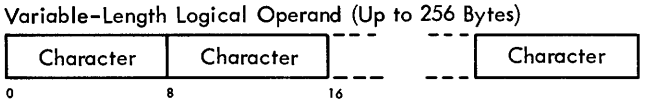


Figure 12. Variable-Length Logical Information

The CPU can handle any eight-bit character set, although certain restrictions are assumed in the decimal arithmetic and editing operations. However, all character-set sensitive I/O equipment will assume either the extended binary-coded-decimal interchange code (EBCDIC) or the USA Standard Code for Information Interchange (USASCII) extended to eight bits, referred to as USASCII-8 in this manual. The numbering convention for the bit positions within a character differ for each of the codes. The conventions are as follows:

	BIT POSITIONS
EBCDIC	01234567
USASCII-8	87654321

The preferred codes do not have a graphic defined for all 256 eight-bit codes. When it is desirable to represent all possible bit patterns, a hexadecimal representation may be used instead of the preferred eight-bit code. The hexadecimal representation uses one graphic for a four-bit code, and therefore, two graphics for an eight-bit byte. The graphics 0-9 are used for codes 0000-1001; the graphics A-F are used for codes 1010-1111. The code tables are in Appendix F.

Program Execution

The CPU program consists of instructions, index words, and control words specifying the operations to be performed. This information resides in main storage and general registers, and may be operated upon as data.

Instruction Format

The length of an instruction format can be one, two, or three halfwords. It is related to the number of storage addresses necessary for the operation. An instruction consisting of only one halfword causes no reference to main storage. A two-halfword instruction provides one storage-address specification; a three-halfword instruction provides two storage-address specifications. All instructions must be located in storage on integral boundaries for halfwords. Figure 13 shows five basic instruction formats.

The five basic instruction formats are denoted by the format codes RR, RX, RS, SI, and SS. The format codes express, in general terms, the operation to be performed. RR denotes a register-to-register operation; RX, a register-and-indexed-storage operation; RS, a register-and-storage operation; SI, a storage and immediate-operand operation; and SS, a storage-to-storage operation. An immediate operand is one contained within the instruction.

For purposes of describing the execution of instructions, operands are designated as first and second operands and, in the case of branch-on-index instructions, third operands. These names refer to the manner in which the operands participate. The operand to which a field in an instruction format applies is generally denoted by the number following the code name of the field, for example, R_1, B_1, L_2, D_2.

In each format, the first instruction halfword consists of two parts. The first byte contains the operation code (op code). The length and format of an instruction are specified by the first two bits of the operation code.

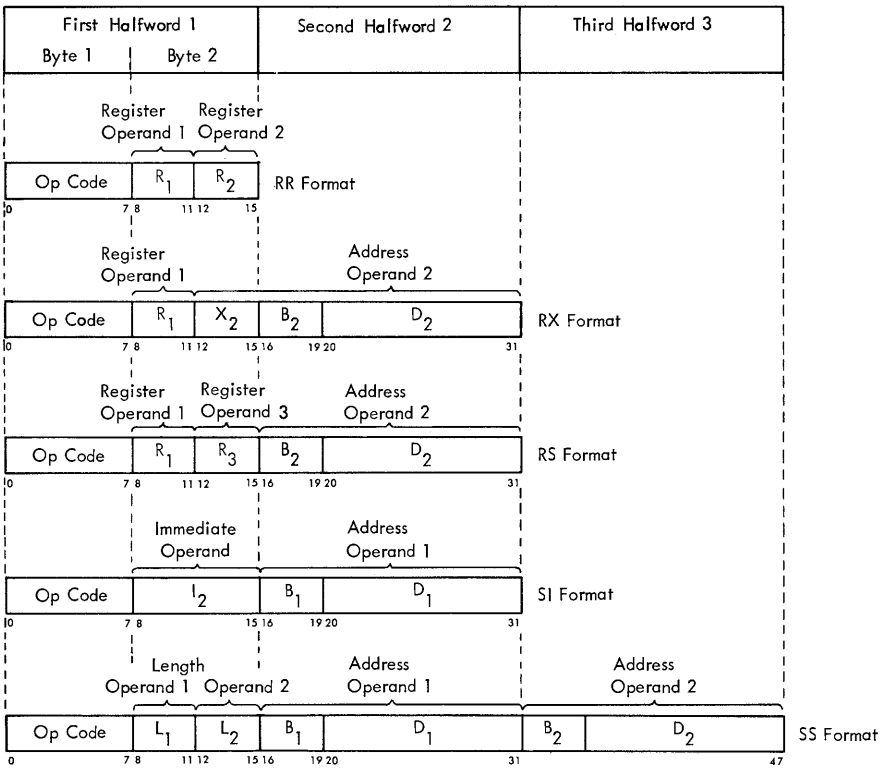


Figure 13. Five Basic Instruction Formats

INSTRUCTION LENGTH RECORDING

BIT POSITIONS (0-1)	INSTRUCTION LENGTH	INSTRUCTION FORMAT
00	One halfword	RR
01	Two halfwords	RX
10	Two halfwords	RS or SI
11	Three halfwords	SS

The second byte is used either as two 4-bit fields or as a single eight-bit field. This byte can contain the following information:

Four-bit operand register specification (R_1, R_2, or R_3)
Four-bit index register specification (X_2)
Four-bit mask (M_1)
Four-bit operand length specification (L_1 or L_2)
Eight-bit operand length specification (L)
Eight-bit byte of immediate data (I_2)

In some instructions a four-bit field or the whole second byte of the first halfword is ignored.

The second and third halfwords always have the same format:

Four-bit base register designator (B_1 or B_2), followed by a 12-bit displacement (D_1 or D_2).

Address Generation

For addressing purposes, operands can be grouped in three classes: explicitly addressed operands in main storage, immediate operands placed as part of the instruction stream in main storage, and operands located in the general or floating-point registers.

To permit the ready relocation of program segments and to provide for the flexible specifications of input, output, and working areas, all instructions referring to main storage have been given the capacity of employing a full address.

The address used to refer to main storage is generated from the following three binary numbers:

Base Address (B) is a 24-bit number contained in a general register specified by the program in the B field of the instruction. The B field is included in every address specification. The base address can be used as a means of static relocation of programs and data. In array-type calculations, it can specify the location of an array and, in record-type processing, it can identify the record. The base address provides for addressing the entire main storage. The base address may also be used for indexing purposes.

Index (X) is a 24-bit number contained in a general register specified by the program in the X field of the instruction. It is included only in the address specified by the RX instruction format. The RX format instructions permit double indexing; i.e., the index can be used to provide the address of an element within an array.

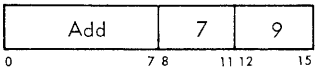
Displacement (D) is a 12-bit number contained in the instruction format. It is included in every address computation. The displacement provides for relative addressing up to 4095 bytes beyond the element or base address. In array-type calculations the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

In forming the address, the base address and index are treated as unsigned 24-bit positive binary integers. The displacement is similarly treated as a 12-bit positive binary integer. The three are added as 24-bit binary numbers, ignoring overflow. Since every address includes a base, the sum is always 24 bits long. The address bits are numbered 8-31 corresponding to the numbering of the base address and index bits in the general register.

The program may have zeros in the base address, index, or displacement fields. A zero is used to indicate the absence of the corresponding address component. A base or index of zero implies that a zero quantity is to be used in forming the address, regardless of the contents of general register 0. A displacement of zero has no special significance. Initialization, modification, and testing of base addresses and indexes can be carried out by fixed-point instructions, or by BRANCH AND LINK, BRANCH ON COUNT, or BRANCH-ON-INDEX instructions.

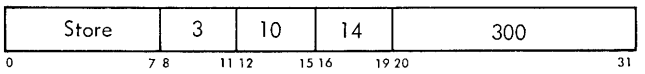
As an aid in describing the logic of the instruction format, examples of two instructions and their related instruction formats follow.

RR Format



Execution of the ADD instruction adds the contents of general register 9 to the contents of general register 7 and the sum of the addition is placed in general register 7.

RX Format



Execution of the 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.

Sequential Instruction Execution

Normally, the operation of the CPU is controlled by instructions taken in sequence. An instruction is fetched from a location specified by the instruction address in the current PSW. The instruction address is then increased by the number of bytes in the instruction fetched to address the next instruction in sequence. The instruction is then executed and the same steps are repeated using the new value of the instruction address.

Conceptually, all halfwords of an instruction are fetched from storage after the preceding operation is completed and before execution of the current operation, even though physical storage word size and overlap of instruction execution with storage access may cause actual instruction fetching to be different. Thus, it is possible to modify an instruction in storage by the immediately preceding instruction.

A change from sequential operation may be caused by branching, status switching, interruptions, or manual intervention.

Branching

The normal sequential execution of instructions is changed when reference is made to a subroutine, when a two-way choice is encountered, or when a segment of coding, such as a loop, is to be repeated. All these tasks can be accomplished with branching instructions. Provision is made for subroutine linkage, permitting not only the introduction of a new instruction address but also the preservation of the return address and associated information.

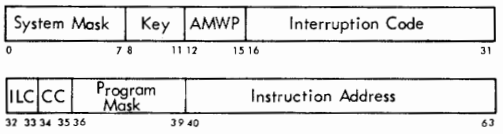
Decision-making is generally and symmetrically provided by the BRANCH ON CONDITION instruction. This instruction inspects a two-bit *condition code* that reflects the result of a majority of the arithmetic, logical, and I/O operations. Each of these operations can set the code in any one of four states, and the conditional branch can specify any selection of these four states as the criterion for branching. For example, the condition code reflects such conditions as nonzero, first operand high, equal, overflow, channel busy, zero, etc. Once set, the condition code remains unchanged until modified by an instruction that reflects a different condition code.

The two bits of the condition code provide for four possible condition code settings: 0, 1, 2, and 3. The specific meaning of any setting is significant only to the operation setting the condition code.

Loop control can be performed by the conditional branch when it tests the outcome of address arithmetic and counting operations. For some particularly frequent combinations of arithmetic and tests, the instructions BRANCH ON COUNT and BRANCH ON INDEX are provided. These branches, being specialized, provide increased performance for these tasks.

Program Status Word

A double word, the program status word (PSW), contains the information required for proper program execution. The PSW includes the instruction address, condition code, and other fields to be discussed. In general, the PSW is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently being executed. The active or controlling PSW is called the "current PSW." By storing the current PSW during an interruption, the status of the CPU can be preserved for subsequent inspection. By loading a new PSW or part of a PSW, the state of the CPU can be initialized or changed. Figure 14 shows the PSW format.



0-7	System mask	14	Wait state (W)
0	Channel 0 mask	15	Problem state (P)
1	Channel 1 mask	16-31	Interruption code
2	Channel 2 mask	32-33	Instruction Length code (ILC)
3	Channel 3 mask	34-35	Condition code (CC)
4	Channel 4 mask	36-39	Program mask
5	Channel 5 mask	36	Fixed-point overflow mask
6	Channel 6 mask	37	Decimal overflow mask
7	External mask	38	Exponent underflow mask
8-11	Protection key	39	Significance mask
12	ASCII(A)	40-63	Instruction address
13	Machine-check mask (M)		

Figure 14. Program Status Word Format

Interruption

The interruption system permits the CPU to change state as a result of conditions external to the system, in input/output (I/O) units, or in the CPU itself. Five classes of interruption conditions are possible: I/O, program, supervisor call, external, and machine check.

Each class has two related PSW's called "old" and "new" in unique main-storage locations (Figure 15). In all classes, an interruption involves merely storing the current PSW in its "old" position and making the PSW at the "new" position the current PSW. The "old" PSW holds all necessary status information of the system existing at the time of the interruption. If, at the conclusion of the interruption routine, there is an instruction to make the old PSW the current PSW, the system is restored to the state prior to the interruption and the interrupted routine continues.

	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 15. Permanent Storage Assignments

Interruptions are taken only when the CPU is interruptable for the interruption source. The system mask, program mask, and machine check mask bits in the PSW may be used to mask certain interruptions. When masked off, an interruption either remains pending or is ignored. The system mask may keep I/O and external interruptions pending, the program mask may cause four of the 15 program interruptions to be ignored, and the machine-check mask may cause machine-check interruptions to remain pending. Other interruptions cannot be masked off.

An interruption always takes place after one instruction execution is finished and before a new instruction execution is started. However, the occurrence of an interruption may affect the execution of the current instruction. To permit proper programmed action following an interruption, the cause of the interruption is identified and provision is made to locate the last executed instruction.

Input/Output Interruption

An I/O interruption provides a means by which the CPU responds to conditions in the channels and I/O units.

An I/O interruption can occur only when the mask bit associated with the channel is set to one. The address of the channel and I/O unit involved are recorded in bits 16-31 of the old PSW. Further information concerning the I/O action is preserved in the channel status word (CSW) that is stored during the interruption.

Program Interruption

Unusual conditions encountered in a program create program interruptions. These conditions include incorrect operands and operand specifications, as well as exceptional results. The interruption code identifies the interruption cause. Figure 16 shows the different causes that may occur.

Interruption Code	Program Interruption Cause
1 00000001	Operation
2 00000010	Privileged operation
3 00000011	Execute
4 00000100	Protection
5 00000101	Addressing
6 00000110	Specification
7 00000111	Data
8 00001000	Fixed-point overflow
9 00001001	Fixed-point divide
10 00001010	Decimal overflow
11 00001011	Decimal divide
12 00001100	Exponent overflow
13 00001101	Exponent underflow
14 00001110	Significance
15 00001111	Floating-point divide

Figure 16. Interruption Code for Program Interruption

Supervisor-Call Interruption

This interruption occurs as a result of execution of the instruction SUPERVISOR CALL. Eight bits from the instruction format are placed in the interruption code of the old PSW, permitting an identification to be associated with the interruptions. A major use for the instruction SUPERVISOR CALL is to switch from the problem-state to the supervisor state. This interruption may also be used for other modes of status-switching.

External Interruption

The external interruption provides the means by which the CPU responds to signals from the interruption key on the system control panel, the timer, and the external signals of the direct control feature.

An external interruption can occur only when system mask bit 7 in the PSW is one.

The source of the interruption is identified by the interruption code in bits 24-31 of the PSW (Figure 17). Bits 16-23 of the interruption code are made zero.

Interruption Code Bit	External Interruption Cause	Mask Bit
24	Timer	7
25	Interrupt key	7
26	External signal 2	7
27	External signal 3	7
28	External signal 4	7
29	External signal 5	7
30	External signal 6	7
31	External signal 7	7

Figure 17. Interruption Code for External Interruption

Machine-Check Interruption

The occurrence of a machine check (if not masked off) terminates the current instruction, initiates a diagnostic procedure, and subsequently causes the machine-check interruption. A machine check cannot be caused by invalid data or instructions. The diagnostic scan is performed into the scan area starting at location 128. Proper execution of these steps depends on the nature of the machine check.

Priority of Interruptions

During execution of an instruction, several interruption requests may occur. Simultaneous interruption requests are honored in the following predetermined order:

Machine Check
Program or Supervisor Call
External
Input/Output

The program and supervisor-call interruptions are mutually exclusive and cannot occur at the same time.

When more than one interruption cause requests service, the action consists of storing the old PSW and fetching the new PSW belonging to the interruption which is taken first. This new PSW subsequently is stored without any instruction execution and the next interruption PSW is fetched. This process continues until no more interruptions are to be serviced. When the last interruption request has been serviced, instruction execution is resumed using the PSW last fetched. The order of execution of the interruption subroutines is, therefore, the reverse of the order in which the PSW's are fetched.

Thus, the most important interruptions — I/O, external, program or supervisor call — are actually serviced first. Machine check, when it occurs, does not allow any other interruptions to be taken.

Program States

Over-all CPU status is determined by four types of program-state alternatives, each of which can be changed independently to its opposite and most of which are indicated by a bit or bits in the PSW. The program-state alternatives are named stopped or operating, running or waiting, masked or interruptible, and supervisor or problem state. These states differ in the way they affect the CPU functions and the manner in which their status is indicated and switched. All program states are independent of each other in their functions, indication, and status-switching.

Stopped or Operating States: The stopped state is entered and left by manual procedure. Instructions are not executed, interruptions are not accepted, and the timer is not updated. In the operating state, the CPU

is capable of executing instructions and being interrupted.

Running or Waiting State: In the running state, instruction fetching and execution proceed in the normal manner. The wait state is normally entered by the program to await an interruption, for example, an I/O interruption or operator intervention from the console. In the wait state, no instructions are processed, the timer is updated, and I/O and external interruptions are accepted, unless masked. Running or waiting state is determined by the setting of bit 14 in the PSW.

Masked or Interruptible State: The CPU may be interruptible or masked for I/O, external, machine-check, and some program interruptions. When the CPU is interruptible for a class of interruptions, these interruptions are accepted. When the CPU is masked, the I/O, external, and machine-check interruptions remain pending, whereas program interruptions are ignored. The interruptible states of the CPU are changed by changing the mask bits of the PSW.

Supervisor or Problem State: In the problem state, all I/O instructions and a group of control instructions are invalid. In the supervisor state, all instructions are valid. The choice of problem or supervisor state is determined by bit 15 of the PSW.

Byte-Oriented Operand Feature

When the byte-oriented operand feature is installed, the restriction that all halfword, word, and double-word operands in main storage must be located at addresses that are integral multiples of the operand length is changed to the extent that all storage operands of unprivileged operations can appear on any byte boundary. The change affects storage references made by the CPU with RX and RS format instructions and applies to fixed-point, floating-point, and logical operands.

The feature does not pertain to instruction addresses. Instructions still must appear on even-byte boundaries. The low-order bit of a branch address must be zero, and the instruction EXECUTE still must designate the subject instruction on an even byte address.

The feature does not apply to the operands designated by privileged instructions. The instruction LOAD PSW still must designate an operand located on a double-word boundary. Similarly, SET STORAGE KEY and INSERT STORAGE KEY must still designate operands that start at a quadruple-word boundary, and DIAGNOSE may, depending upon the model, require a number of low-order bit positions of the operand address to contain zeros.

The feature does not affect channel operation. A channel command word (CCW) still must be located on a double-word boundary, and the constraints on address resolution in the channel address word (CAW) and transfer in channel are maintained.

When the feature is installed, a number of instructions that ordinarily could cause a specification exception cannot cause this exception. Also, the halfword, word, and double-word store-type operations that ordinarily are suppressed upon a protection or addressing exception (CONVERT TO DECIMAL, STORE HALFWORD, and the three store instructions ST, STD, and STE) now are terminated when a protection or addressing exception is recognized.

Programming Note

Significant performance degradation is possible when storage operands are not positioned at addresses that are integral multiples of the operand length. To ensure optimum performance, storage operands should be aligned on integral boundaries, and use of unaligned operands should be reserved for exceptional cases.

Protection Features

Two protection features are available. These features make it possible to protect the contents of main storage from destruction or misuse. When the store-protection feature is installed, attempts to modify storage are monitored. The addition of the fetch-protection feature to the store-protection feature provides for monitoring of all accesses to storage.

Protection is achieved by dividing main storage into blocks of 2,048 bytes, and by associating a five-bit key with each block. Two instructions — SET STORAGE KEY and INSERT STORAGE KEY — are provided for assigning and inspecting the code in a key. The same code may be used in many keys.

A user's right of access to storage is identified by a four-bit *protection* key. For references caused by the CPU, the protection key in the current PSW is used; accesses by channels are controlled by the protection key assigned to the associated I/O operation.

When protection applies to a main-storage reference, the key in storage is compared with the protection key associated with the reference. Access to the location, for both operands and instructions, is granted only when the two keys match. The keys are said to match when the four high-order bits of the key in storage are equal to the protection key or when the protection key is zero. When store-and-fetch protection is installed, the low-order bit of the key in storage is used to specify whether or not fetching is to be monitored.

When a protection mismatch is detected, the content of the protected main-storage location remains unaltered. A protection violation due to a CPU reference causes the instruction to be suppressed or terminated and program execution to be altered by an interruption. A violation due to an I/O operation causes the I/O operation to be terminated, with the protection mismatch indicated in the channel status word stored at the end of an I/O operation.

Timer Feature

The timer is provided as an interval timer and may be programmed to maintain the time of day. The timer consists of a full word in main-storage location 80. The timer word is counted down at a rate of 50 or 60 cycles per second, depending on line frequency. The timer word is treated as a signed integer following the rules of fixed-point arithmetic. An external interruption condition is signaled when the value of the timer word goes from positive to negative. The full cycle time of the timer is 15.5 hours.

An updated timer value is available at the end of each instruction execution but is not updated in the stopped state. The timer is changed by addressing storage location 80. As an interval timer, the timer is used to measure elapsed time over relatively short intervals. It can be set to any value at any time.

Direct Control Feature

The direct control feature provides two instructions, READ DIRECT and WRITE DIRECT, and six external interruption lines. The read and write instructions provide for the transfer of a single byte of information between an external device and the main storage of the system. It is usually most desirable to use the data channels of the system to handle the transfer of any volume of information and use the direct data control feature to pass controlling and synchronizing information between the CPU and special external devices.

Each of the six external signal lines, when pulsed, sets up the conditions for an external interruption.

Multisystem Operation

The design of System/360 permits communication between individual CPU's at several transmission rates.

The communication is possible through shared control units, through a channel-to-channel adapter, and through shared storage. Interconnection of CPU's is further enhanced by the direct control feature (described in the previous section), which can be used to signal from one CPU to another, and by facilities for direct address relocation, malfunction indication, and external CPU initialization.

The relocation procedure applies to the first 4,096 bytes of storage. This area contains all permanent storage assignments and, generally, has special significance to supervisory programs. The relocation is accomplished by inserting a 12-bit prefix in each address which has the high-order 12 bits set to zero and hence, pertains to location 0-4095. Two manually set prefixes are available to permit the use of an alternative area when storage malfunction occurs. The choice between the prefixes is determined by a prefix trigger set during initial program loading.

To alert one CPU to the possible malfunction of another CPU, a machine check-out signal is provided, which can serve as an external interruption to another CPU.

Finally, provision is made for starting one CPU by a signal from another CPU.

Input and Output

The following information is introductory in nature. For thorough definition of the input/output system, see "Input/Output Operations."

Input/Output Devices and Control Units

Input/output operations involve the transfer of information to or from main storage and an I/O device. Input/output devices include such equipment as card readers and punches, magnetic tape units, disk storage, drum storage, typewriter-keyboard devices, printers, teleprocessing devices, and process control equipment.

Many I/O devices function with an external document, such as a punched card or a reel of magnetic tape. Some I/O devices handle only electrical signals, such as those found in process-control networks. In either case, I/O device operation is regulated by a control unit. The control-unit function may be housed with the I/O device, as is the case with a printer, or a separate control unit may be used. In all cases, the control-unit function provides the logical and buffering capabilities necessary to operate the associated I/O device. From the programming point of view, most control-unit functions merge with I/O device functions.

Input/Output Interface

All communication between the control unit and the channel takes place over a connection called the I/O interface. The I/O interface provides an information format and control signal sequences that are independent of the type of control unit and channel and provide a uniform means of attaching and controlling various types of I/O devices.

Channels

The channel controls transfer of data between I/O devices and main storage. It connects with the CPU and main storage and, via the I/O interface, with control units. The channel relieves the CPU of the burden of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O operations.

A channel may be an independent unit, complete with necessary logical and storage capabilities, or it may time-share CPU facilities and be physically integrated with the CPU. In either case, channel functions are identical. Channels may be implemented, however, to have different maximum data transfer capabilities.

The System/360 has two types of channels: multiplexor and selector. The channel facility necessary to sustain an operation with an I/O device is called a subchannel. The selector channel has one subchannel; the multiplexor channel has multiple subchannels.

Channels have two modes of operation: burst and multiplex.

In the burst mode, the data transfer facilities of the channel are monopolized for the duration of transfer of a burst of data. Other devices attached to the channel cannot transfer data until the burst ceases. The selector channel functions only in the burst mode.

The multiplexor channel functions in either the burst mode or in the multiplex mode. In the multiplex mode, the multiplexor channel can sustain concurrent I/O operations on several subchannels. Bytes of data associated with different I/O devices are interleaved and routed to or from the desired locations in main storage. The I/O interface is time-shared by a number of concurrently operating I/O devices, each of which uses its own subchannel.

Some I/O devices can operate only in burst mode. Other I/O devices have a manual switch in the control unit that may be set to a burst-mode or to a multiplex-mode position, when attached to a multiplexor chan-

nel. When attached to a selector channel, an I/O device can operate only in burst mode.

Input/Output Instructions

The System/360 uses only four I/O instructions:

START I/O
TEST I/O
HALT I/O
TEST CHANNEL

Input/output instructions can be executed only while the CPU is in the supervisor state.

Start I/O

The START I/O instruction is used to initiate an I/O operation. The address part of the instruction specifies the channel and I/O device.

Test I/O

Execution of the TEST I/O instruction sets the condition code in the PSW to indicate the state of the addressed channel, subchannel, and I/O device, and may cause a CSW to be stored. The instruction may be used to clear I/O interruption conditions, selectively by device.

Halt I/O

The HALT I/O instruction terminates a channel operation.

Test Channel

Execution of the TEST CHANNEL instruction sets the condition code in the PSW to indicate the state of the channel addressed by the instruction. The resulting condition code indicates one of the following: channel available, interruption condition in channel, channel working, or channel not operational.

Input/Output Operation Initiation

An I/O operation is initiated by a START I/O instruction. If the necessary channel and device facilities are available, START I/O is accepted and the CPU continues its program. The channel independently governs the I/O device specified by the instruction.

Channel Address Word

Successful execution of START I/O causes the channel to fetch a channel address word (CAW) from the main-storage location 72. The CAW specifies the byte location in main storage where the channel program begins.

Figure 18 shows the format for the CAW. Bits 0-3 specify the storage-protection key that will govern the I/O operation. Bits 4-7 must contain zeros. Bits 8-31 specify the location of the first channel command word (CCW).

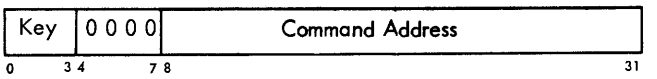


Figure 18. Channel Address Word Format

Channel Command Word

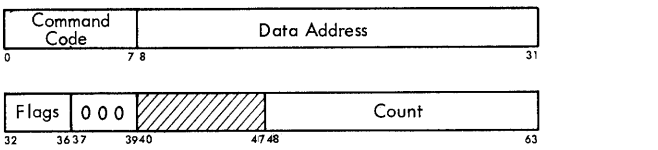
The byte location specified by the CAW is the first of eight bytes of information that the channel fetches from main storage. These 64 bits of information are called a channel command word (CCW). Only the START I/O instruction may cause the channel to fetch CCW's.

One or more CCW's make up the channel program that directs channel operations.

A channel command word can specify one of six commands:

Read
Write
Read Backward
Control
Sense
Transfer In Channel

If more than one CCW is to be fetched, the CCW's are to be fetched sequentially, except when transfer in channel is encountered. Figure 19 shows the format for CCW's.



The command code specifies the operation to be performed (read, write, rewind, etc.).

The data address specifies the first byte location in main storage for a data transfer type of operation.

The flag bits may specify chaining to another CCW, suppression of a possible incorrect-length indication, etc.

The count specifies the number of bytes for a data transfer operation.

Figure 19. Channel Command Word Format

Input/Output Commands

Read

The read command causes data to be read from the selected I/O device and defines the area in main storage to be used.

Write

The write command causes a write operation on the selected I/O device and defines the data in main storage to be written.

Read Backward

The read-backward command causes a read operation in which the characters are read from the external document in reverse order by the I/O device. Bytes read backward are placed in descending main storage locations.

Control

The control command contains information used to control the selected I/O device. This control information is called an order. Order information may be entirely contained in the command code, or the control command may provide a data address and byte count for additional order information in main storage to be fetched by the channel. Also, a control command may specify information in main storage such as the address of a particular disk storage track.

Orders are peculiar to the particular I/O device in use; orders can specify such functions as rewinding a tape unit, loading a tape cartridge, or line skipping on a printer. A control command may cause mechanical motion by an I/O device, or it may specify a function altogether electronic in nature, such as setting the recording density for a tape unit operation.

The general relationship of I/O instructions, commands, and orders is shown in Figure 20.

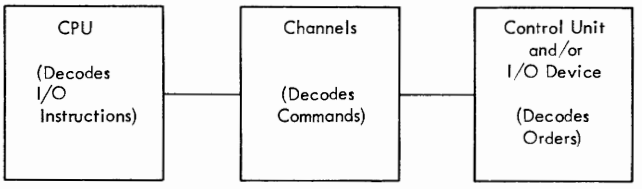


Figure 20. Relationship of I/O Instructions, Commands, and Orders

Sense

The sense command specifies the beginning main storage location to which sense information is transferred from the selected control unit. One or more bytes of sense data may be specified, depending upon the type of I/O device. The sense data provides detailed information concerning the selected I/O device, such as a stacker-full condition of a card reader or a file-protected condition of a reel of magnetic tape on a tape unit. Sense data have significance peculiar to the type of I/O device involved.

Transfer In Channel

The transfer-in-channel (TIC) command specifies the location of the next CCW to be fetched and used by the channel. The TIC command is used whenever the programmer wants to specify a CCW that is not located at the next higher double word location in main storage. The TIC command permits a programmer to cause execution of any CCW, including a CCW immediately preceding a TIC command, except that the channel will not permit a TIC command to specify execution of another TIC command. Also, the CAW may not address a TIC command.

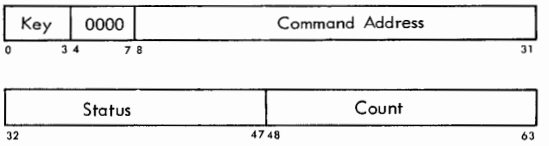
Input/Output Termination

Input/output operations terminate with the device and channel signaling end of operation and a request for an I/O interruption.

A command can be rejected during an attempt to execute a START I/O, however, by a busy condition, by a channel programming error, etc. The condition code set in the PSW by an unsuccessful START I/O instruction will indicate one of the following: that a channel status word (CSW) has been stored to detail the conditions that precluded initiation of the I/O operation, that the equipment is busy, or that the addressed equipment is not operational.

Channel Status Word

The channel status word (CSW) provides information about the termination of an I/O operation. It can be formed or reformed by START I/O, TEST I/O, HALT I/O, or by an I/O interruption. The instruction TEST CHANNEL does not affect the CSW. Figure 21 shows the CSW format.



The key field contains the protection key used in the last operation.

The command address specifies the location plus 8 of the last CCW used.

The status field contains a unit status byte and a channel status byte. The unit status byte may indicate one or more conditions. such as control unit end, device end, busy, etc. The channel status byte may indicate a channel programming error, a channel data check, etc.

The count field specifies the residual count of the last CCW used.

Figure 21. Channel Status Word Format

Input/Output Interruptions

Input/output interruptions are caused by termination of an I/O operation or by operator intervention at the I/O device. An I/O interruption stores the current PSW in the I/O old PSW location, and places the I/O new PSW in control of the system. The I/O new PSW, when

made current by an I/O interruption, may cause CPU interrogation of the channel status word, or take whatever action is considered appropriate by the programmer.

An I/O interruption request may be initiated by an I/O interruption condition in a device, a control unit, or a channel. When a channel has multiple I/O interruption requests pending, it establishes a priority sequence for them before initiating an I/O interruption request to the CPU. Conditions responsible for I/O interruption requests remain pending in the I/O devices or channels until they are accepted by the CPU.

Basic Procedure for a Data-Transfer Operation

A START I/O instruction is used to initiate data transfer to or from an I/O device. To perform such an operation, it is necessary for the programmer to:

1. Establish a channel command word (CCW) or a list of CCW's in main storage.
2. Load the channel address word (CAW) with the address of the first byte of the first CCW in the channel program.
3. Load the channel and device address in the START I/O instruction to be used for the operation.
4. Set the system mask to disable all channels for I/O interruptions.
5. Issue the START I/O instruction.
6. Test the condition code established in the current PSW by termination of the START I/O.

Condition code 0 indicates that the I/O operation has been initiated and that the channel is proceeding with its execution. If an I/O interruption is desired upon termination of the operation, the pertinent channel mask bit must be set to one (an appropriate I/O new PSW must have been established previously).

Condition code 1 indicates that a channel status word (CSW) has been stored; its status bytes should be examined to determine why the desired operation was not initiated.

Condition code 2 indicates that the channel or subchannel addressed by the START I/O instruction was found to be busy with a previously initiated operation. If an I/O interruption from the operation already in progress is desired, the channel mask bit must be set to one.

Condition code 3 indicates that the addressed equipment is not operational; a message to the operator may be initiated.

Between the time a START I/O instruction is decoded by the CPU, and the time the CPU is released by the channel with condition code 0 set in the current PSW, the channel performs many functions. The CAW must be fetched, the first CCW must be fetched, the CAW and CCW must be tested for validity, etc. After a START I/O results in condition code 0, the operation continues until terminated. Termination of an I/O operation causes a request for an I/O interruption. Some of the relationships of CPU and channel functions are illustrated in Figure 22. The example covers the time span from initation of a START I/O instruction to a resulting I/O interruption.

The example illustrates a simple read operation. The START I/O used in Figure 22 addresses an IBM 2403 Magnetic Tape Unit and Control. The CAW addresses a CCW specifying a read operation. The CCW does not, however, specify command chaining or data chaining. Therefore, the single read CCW constitutes the entire channel program for the example.

The example is limited to the START I/O considerations shown in Figure 22; a successful read operation is assumed, and many machine functions, such as channel testing of CAW and CCW for validity, device selection, etc., are not represented. Similarly, other system operations, such as concurrent I/O operations, multiprogramming, etc., are not considered.

The purpose of this part of the manual has been to illustrate a data transfer operation, with major emphasis given to the point-in-time relationships of CPU and channel functions. The START I/O example provided does not purport to show how data-transfer programming should be done; a programmer familiar with the I/O system may generate considerably more comprehensive I/O routines.

See the "Input/Output Operations" section of this manual for thorough, detailed description of I/O operations.

System Control Panel

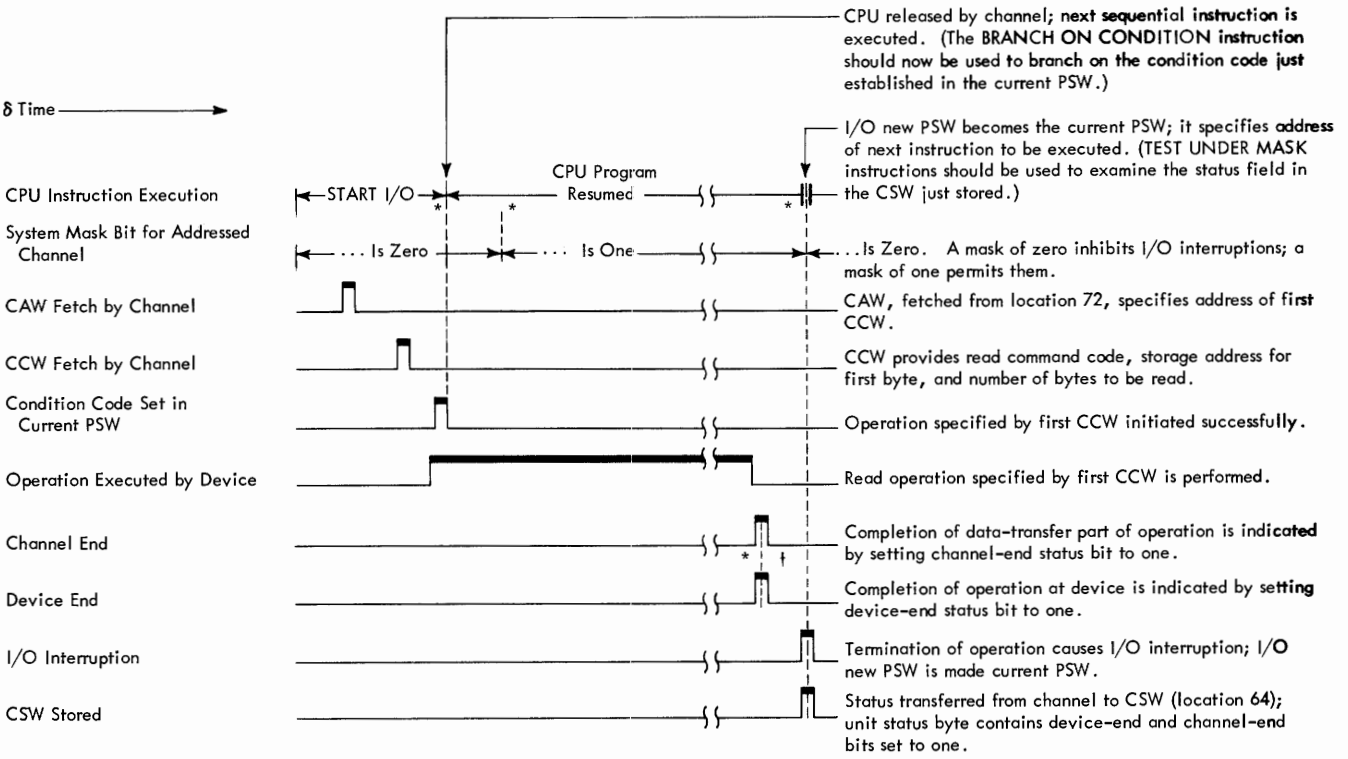
The system control panel provides the switches, keys, and lights necessary to operate and control the system. The need for operator manipulation of manual controls is held to a minimum by the system design and the governing supervisory program. The result is fewer and less serious operator errors.

System Control Panel Functions

The main functions provided by the system control panel are the ability to: reset the system; store and display information in main storage, in registers, and in the PSW; and load initial program information.

System Reset

The system-reset function resets the CPU, the channels, and on-line control units and I/O devices. In general, the system is placed in such a state that processing can be initiated without the occurrence of machine checks, except those caused by subsequent machine malfunction.



δ Timing relationships shown here are only relative. The purpose of this chart is limited to illustration of the relationship of CPU and channel actions; this chart is not intended to illustrate signal lines, timing pulses, voltage levels, etc.

* Dashed lines link concurrent actions.

† Channel end and device end are concurrent for tape read or write operations, but other devices or operations cause channel-end and device-end signals to be separated by time, which may result in two I/O interruptions.

SIO:	START I/O
CAW:	Channel Address Word
CCW:	Channel Command Word
CSW:	Channel Status Word
PSW:	Program Status Word

Figure 22. Basic Timing Chart for IBM 2403 Tape Read Operation

Store and Display

The store-and-display function permits manual intervention in the progress of a program. The function may be provided by a supervisory program in conjunction with proper I/O equipment and the interrupt key. Or, the system-control-panel facilities may be used to place the CPU in the stopped state, and then to store and display information in main storage, in general and floating-point registers, and in the instruction-address portion of the PSW.

Initial Program Loading

The initial-program-loading (IPL) procedure is used to begin or renew system operation. The load key is pressed after an input device is selected with the load-unit switches. This causes a read operation at the selected input device. Six words of information are read into main storage and may be used for reading more information into any part of main storage. Upon completion of the IPL read operation, the double word from location 0 is made the current PSW for subsequent control of the system.

The system controls are divided into three sections: operator control, operator intervention, and customer engineering control.

Operator Control Section

This section of the system control panel contains the operator controls required when the CPU is operating under supervisory program control.

The main functions provided are the control and indication of power, the indication of system status, and operator-to-machine communication. These include:

Emergency power-off pull switch
Power-on back-lighted key
Power-off key
Interrupt key
Wait light
Manual light
System light
Test light
Load light
Load-unit switches
Load key

Operator Intervention Section

This section of the system control panel provides controls required for operator intervention into normal programmed operation. These include:

System reset key
Stop key
Start key
Rate switch (single cycle or normal processing)
Storage-select switches
Address switches
Data switches
Store key
Display key
Set IC key
Address compare switches

Customer Engineering Section

This section of the system control panel provides the controls intended only for customer engineering use. Customer engineering controls are also available on some storage, channel, and control-unit equipment.

Fixed-Point Arithmetic

The fixed-point instruction set performs binary arithmetic on operands serving as addresses, index quantities, and counts, as well as fixed-point data. In general, both operands are signed and 32 bits long. Negative quantities are held in two's-complement form. One operand is always in one of the 16 general registers; the other operand may be in main storage or in a general register.

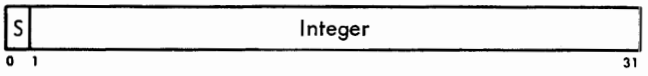
The instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, and storing, as well as for the sign control, radix conversion, and shifting of fixed-point operands. The entire instruction set is included in the standard instruction set.

The condition code is set as a result of all sign-control, add, subtract, compare, and shift operations.

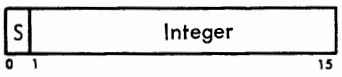
Data Format

Fixed-point numbers occupy a fixed-length format consisting of a one-bit sign followed by the integer field. When held in one of the general registers, a fixed-point quantity has a 31-bit integer field and occupies all 32 bits of the register. Some multiply, divide, and shift operations use an operand consisting of 64 bits with a 63-bit integer field. These operands are located in a pair of adjacent general registers and are addressed by an even address referring to the leftmost register of the pair. The sign-bit position of the rightmost register contains part of the integer. In register-to-register operations the same register may be specified for both operand locations.

Full Word Fixed-Point Number



Halfword Fixed-Point Number



Fixed-point data in main storage occupy a 32-bit word or a 16-bit halfword, with a binary integer field of 31 or 15 bits, respectively. The conversion instructions use a 64-bit decimal field. These data must be located on integral storage boundaries for these units of information, that is, double word, fullword, or halfword operands must be addressed with three, two, or one low-order address bit(s) set to zero.

A halfword operand in main storage is extended to a full word as the operand is fetched from storage. Subsequently, the operand participates as a full word operand.

In all discussions of fixed-point numbers in this publication, the expression "32-bit signed integer" denotes a 31-bit integer with a sign bit, and the expression "64-bit signed integer" denotes a 63-bit integer with a sign bit.

Number Representation

All fixed-point operands are treated as signed integers. Positive numbers are represented in true binary notation with the sign bit set to zero. Negative numbers are represented in two's-complement notation with a one in the sign bit. The two's-complement representation of a negative number may be considered the sum of the integer part of the field, taken as a positive number, and the maximum negative number. The two's complement of a number is obtained by inverting each bit of the number and adding a one in the low-order bit position.

This type of number representation can be considered the low-order portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number, including the sign bit, are zeros. When the number is negative, all these bits, including the sign bit, are ones. Therefore, when an operand must be extended with high-order bits, the expansion is achieved by prefixing a field in which each bit is set equal to the high-order bit of the operand.

Two's-complement notation does not include a negative zero. It has a number range in which the set of negative numbers is one larger than the set of positive numbers. The maximum positive number consists of an all-one integer field with a sign bit of zero, whereas the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a one-bit for sign.

The CPU cannot represent the complement of the maximum negative number. When an operation, such as a subtraction from zero, produces the complement of the maximum negative number, the number remains unchanged, and a fixed-point overflow exception is recognized. An overflow does not result, however, when the number is complemented and the final result is within the representable range. An example of this case is a subtraction from minus one. The product of two maximum negative numbers is representable as a double-length positive number.

The sign bit is leftmost in a number. In an arithmetic operation, a carry out of the integer field changes the sign. However, in algebraic left-shifting the sign bit does not change even if significant high-order bits are shifted out of the integer field.

Condition Code

The results of fixed-point sign-control, add, subtract, compare, and shift operations are used to set the condition code in the program status word (PSW). All other fixed-point operations leave this code undisturbed. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect three types of results for fixed-point arithmetic. For most operations, the states 0, 1, or 2 indicate a zero, less than zero, or greater than zero content of the result register, while the state 3 is used when the result overflows.

For a comparison, the states 0, 1, or 2 indicate that the first operand is equal, low, or high.

For ADD LOGICAL and SUBTRACT LOGICAL, the codes 0 and 1 indicate a zero or nonzero result register content in the absence of a logical carry out of the sign position; the codes 2 and 3 indicate a zero or nonzero result register content with a logical carry out of the sign position.

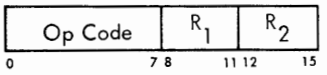
CONDITION CODE SETTINGS FOR FIXED-POINT ARITHMETIC

	0	1	2	3
Add H/F	zero	< zero	> zero	overflow
Add Logical	zero no carry	not zero, no carry	zero, carry	not zero, carry
Compare H/F	equal	low	high	--
Load and Test	zero	< zero	> zero	--
Load Complement	zero	< zero	> zero	overflow
Load Negative	zero	< zero	--	--
Load Positive	zero	--	> zero	overflow
Shift Left Double	zero	< zero	> zero	overflow
Shift Left Single	zero	< zero	> zero	overflow
Shift Right Double	zero	< zero	> zero	--
Shift Right Single	zero	< zero	> zero	--
Subtract H/F	zero	< zero	> zero	overflow
Subtract Logical	--	not zero, no carry	zero, carry	not zero, carry

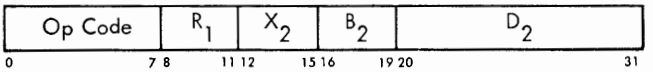
Instruction Format

Fixed-point instructions use the following three formats:

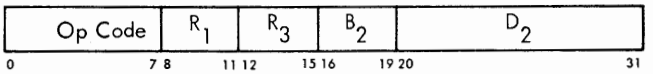
RR Format



RX Format



RS Format



In these formats, R_1 specifies the general 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 general register containing the second operand. The same register may be specified for the first and 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 content of the D_2 field to form an address designating the storage location of the second operand.

In the RS format, the content of the general register specified by the B_2 field is added to the content of the D_2 field. This sum designates the storage location of the second operand in LOAD MULTIPLE and STORE MULTIPLE. In the shift operations, the sum specifies the number of bits of the shift. The R_3 field specifies the address of a general register in LOAD MULTIPLE and STORE MULTIPLE and is ignored in the shift operations.

A zero in an X_2 or B_2 field indicates the absence of the corresponding address component.

An instruction can specify the same general register both for address modification and for operand location. Address modification is always completed before operation execution.

Results replace the first operand, except for STORE and CONVERT TO DECIMAL, where the result replaces the second operand.

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.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction. For LOAD AND TEST,

for example, LTR is the mnemonic and R_1, R_2 the operand designation.

Instructions

The fixed-point arithmetic instructions and their mnemonics, formats, and operation codes are listed in the following table. The table also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

NAME	MNEMONIC	TYPE		EXCEPTIONS		CODE
Load	LR	RR				18
Load	L	RX		P,A,S		58
Load Halfword	LH	RX		P,A,S		48
Load and Test	LTR	RR	C			12
Load Complement	LCR	RR	C		IF	13
Load Positive	LPR	RR	C		IF	10
Load Negative	LNR	RR	C			11
Load Multiple	LM	RS		P,A,S		98
Add	AR	RR	C		IF	1A
Add	A	RX	C	P,A,S,	IF	5A
Add Halfword	AH	RX	C	P,A,S,	IF	4A
Add Logical	ALR	RR	C			1E
Add Logical	AL	RX	C	P,A,S		5E
Subtract	SR	RR	C		IF	1B
Subtract	S	RX	C	P,A,S,	IF	5B
Subtract Halfword	SH	RX	C	P,A,S,	IF	4B
Subtract Logical	SLR	RR	C			1F
Subtract Logical	SL	RX	C	P,A,S		5F
Compare	CR	RR	C			19
Compare	C	RX	C	P,A,S		59
Compare Halfword	CH	RX	C	P,A,S		49
Multiply	MR	RR		S		1C
Multiply	M	RX		P,A,S		5C
Multiply Halfword	MH	RX		P,A,S		4C
Divide	DR	RR		S,	IK	1D
Divide	D	RX		P,A,S,	IK	5D
Convert to Binary	CVB	RX		P,A,S,D,	IK	4F
Convert to Decimal	CVD	RX		P,A,S		4E
Store	ST	RX		P,A,S		50
Store Halfword	STH	RX		P,A,S		40
Store Multiple	STM	RS		P,A,S		90
Shift Left Single	SLA	RS	C		IF	8B
Shift Right Single	SRA	RS	C			8A
Shift Left Double	SLDA	RS	C	S,	IF	8F
Shift Right Double	SRDA	RS	C	S		8E

NOTES

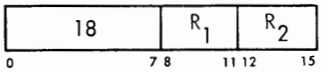
A	Addressing exception
C	Condition code is set
D	Data exception
IF	Fixed-Point overflow exception
IK	Fixed-point divide exception
P	Protection exception
S	Specification exception

Programming Note

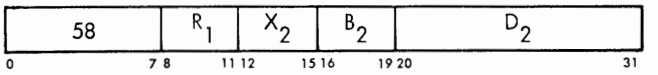
The logical comparisons, shifts, and connectives, as well as LOAD ADDRESS, BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL, also may be used in fixed-point calculations.

Load

LR R_1, R_2 [RR]



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



The second operand is placed in the first operand location. The second operand is not changed.

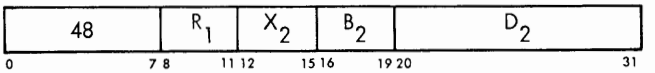
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (fetch violation by L only)
Addressing (L only)
Specification (L only)

Load Halfword

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



The halfword second operand is placed in the first operand location.

The halfword second operand is expanded to a fullword by propagating the sign-bit value through the 16 high-order bit positions. Expansion occurs after the operand is obtained from storage and before insertion in the register.

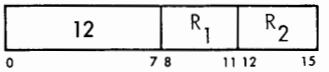
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (fetch violation)
Addressing
Specification

Load and Test

LTR R_1, R_2 [RR]



The second operand is placed in the first operand location, and the sign and magnitude of the second operand determine the condition code. The second operand is not changed.

Resulting Condition Code:

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

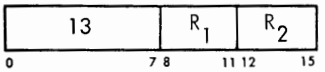
Program Interruptions: None.

Programming Note

When the same register is specified as first and second operand location, the operation is equivalent to a test without data movement.

Load Complement

LCR R_1, R_2 [RR]



The two's complement of the second operand is placed in the first operand location.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

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

Program Interruptions:

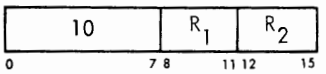
Fixed-point overflow

Programming Note

Zero remains invariant under complementation.

Load Positive

LPR R_1, R_2 [RR]



The absolute value of the second operand is placed in the first operand location.

The operation includes complementation of negative numbers; positive numbers remain unchanged.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

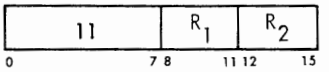
0 Result is zero
1 --
2 Result is greater than zero
3 Overflow

Program Interruptions:

Fixed-point overflow

Load Negative

LNR R_1, R_2 [RR]



The two's complement of the absolute value of the second operand is placed in the first operand location. The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged with positive sign.

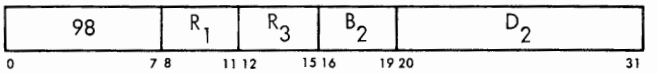
Resulting Condition Code:

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

Program Interruptions: None.

Load Multiple

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



The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is loaded from 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, with register 0 following register 15.

The second operand remains unchanged.

Condition Code: The code remains unchanged.

Program Interruptions:

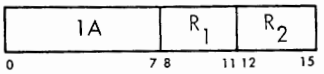
Protection (fetch violation)
Addressing
Specification

Programming Note

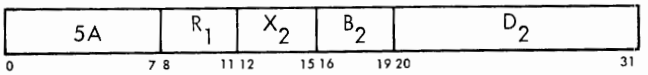
All combinations of register addresses specified by R_1 and R_3 are valid. When the register addresses are equal, only one word is transmitted. When the address specified by R_3 is less than the address specified by R_1, the register addresses wrap around from 15 to 0.

Add

AR R_1, R_2 [RR]



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



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

Addition is performed by adding all 32 bits of both operands. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

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

Program Interruptions:

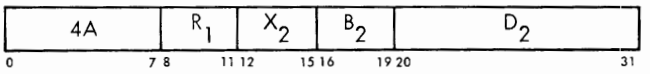
Protection (fetch violation by A only)
Addressing (A only)
Specification (A only)
Fixed-point overflow

Programming Note

In two's-complement notation, a zero result is always positive.

Add Halfword

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



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

The halfword second operand is expanded to a full-word before the addition by propagating the sign-bit value through the 16 high-order bit positions.

Addition is performed by adding all 32 bits of both operands. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

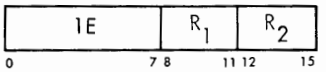
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Interruptions:

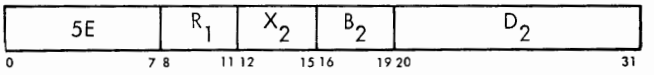
Protection (fetch violation)
Addressing
Specification
Fixed-point overflow

Add Logical

ALR R_1, R_2 [RR]



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



The second operand is added to the first operand, and the sum is placed in the first operand location. The occurrence of a carry out of the sign position is recorded in the condition code.

Logical addition is performed by adding all 32 bits of both operands without further change to the resulting sign bit. The instruction differs from ADD in the meaning of the condition code and in the absence of the interruption for overflow.

If a carry out of the sign position occurs, the leftmost bit of the condition code (PSW bit 34) is made one. In the absence of a carry, bit 34 is made zero. When the sum is zero, the rightmost bit of the condition code (PSW bit 35) is made zero. A nonzero sum is indicated by a one in bit 35.

Resulting Condition Code:

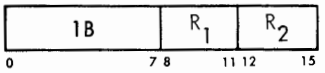
0 Sum is zero (no carry)
1 Sum is not zero (no carry)
2 Sum is zero (carry)
3 Sum is not zero (carry)

Program Interruptions:

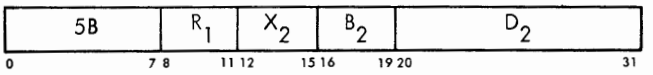
Protection (fetch violation by AL only)
Addressing (AL only)
Specification (AL only)

Subtract

SR R_1, R_2 [RR]



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



The second operand is subtracted from the first operand, and the difference is placed in the first operand location.

Subtraction is considered to be performed by adding the one's complement of the second operand and a low-order one to the first operand. All 32 bits of both operands participate, as in ADD. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow is recognized. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

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

Program Interruptions:

Protection (fetch violation by S only)
Addressing (S only)
Specifications (S only)
Fixed-point overflow

Programming Note

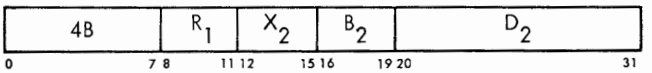
The use of the one's complement and the low-order one instead of the two's complement of the second operand is necessary for proper recognition of overflow.

When the same register is specified as first and second operand location, subtracting is equivalent to clearing the register.

Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

Subtract Halfword

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



The halfword second operand is subtracted from the first operand, and the difference is placed in the first operand location.

The halfword second operand is expanded to a fullword before the subtraction by propagating the sign-bit value through 16 high-order bit positions.

Subtraction is considered to be performed by adding the one's complement of the expanded second operand and a low-order one to the first operand. All 32 bits of both operands participate, as in ADD. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow is recognized. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

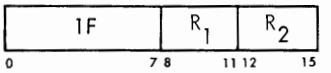
0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Interruptions:

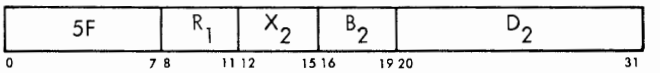
Protection (fetch violation)
Addressing
Specification
Fixed-point overflow

Subtract Logical

SLR R_1, R_2 [RR]



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



The second operand is subtracted from the first operand, and the difference is placed in the first operand location. The occurrence of a carry out of the sign position is recorded in the condition code.

Logical subtraction is considered to be performed by adding the one's complement of the second operand and a low-order one to the first operand. All 32 bits of both operands participate, without further change to the resulting sign bit. The instruction differs from SUBTRACT in the meaning of the condition code and in the absence of the interruption for overflow.

If a carry out of the sign position occurs, the leftmost bit of the condition code (PSW bit 34) is made one. In the absence of a carry, bit 34 is made zero. When the sum is zero, the rightmost bit of the condition code (PSW bit 35) is made zero. A nonzero sum is indicated by a one in bit 35.

Resulting Condition Code:

0 --
1 Difference is not zero (no carry)
2 Difference is zero (carry)
3 Difference is not zero (carry)

Program Interruptions:

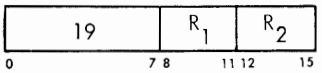
Protection (fetch violation by SL only)
Addressing (SL only)
Specification (SL only)

Programming Note

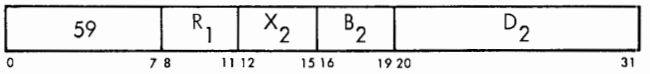
A zero difference cannot be obtained without a carry out of the sign position.

Compare

CR R_1, R_2 [*RR*]



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



The first operand is compared with the second operand, and the result determines the setting of the condition code.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:

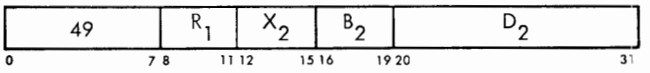
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Interruptions:

Protection (fetch violation by C only)
Addressing (C only)
Specification (C only)

Compare Halfword

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



The first operand is compared with the halfword second operand, and the result determines the setting of the condition code.

The halfword second operand is expanded to a fullword before the comparison by propagating the sign-bit value through the 16 high-order bit positions.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:

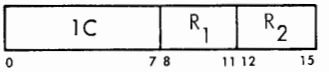
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Interruptions:

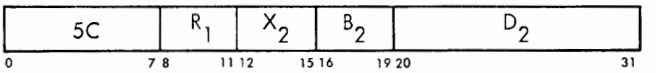
Protection (fetch violation)
Addressing
Specification

Multiply

MR R_1, R_2 [*RR*]



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



The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

Both multiplier and multiplicand are 32-bit signed integers. The product is always a 64-bit signed integer and occupies an even/odd register pair. Because the multiplicand is replaced by the product, the R_1 field of the instruction must refer to an even-numbered register. A specification exception occurs when R_1 is odd. The multiplicand is taken from the odd register of the pair. The content of the even-numbered register replaced by the product is ignored, unless the register contains the multiplier. An overflow cannot occur.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code: The code remains unchanged.

Program Interruptions:

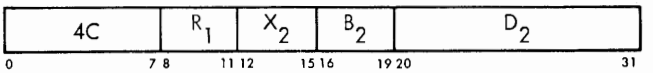
Protection (fetch violation by M only)
Addressing (M only)
Specification

Programming Note

The significant part of the product usually occupies 62 bits or fewer. Only when two maximum negative numbers are multiplied are 63 significant product bits formed. Since two's-complement notation is used, the sign bit is extended right until the first significant product digit is encountered.

Multiply Halfword

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



The product of the halfword multiplier (second operand) and multiplicand (first operand) replaces the multiplicand.

Both multiplicand and product are 32-bit signed integers and may be located in any general register. The halfword multiplier is expanded to a fullword

before multiplication by propagating the sign-bit value through the 16 high-order bit positions. The multiplicand is replaced by the low-order part of the product. The bits to the left of the 32 low-order bits are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code: The code remains unchanged.

Program Interruptions:

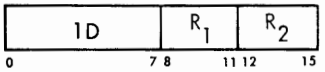
Protection (fetch violation)
Addressing
Specification

Programming Note

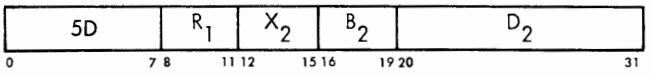
The significant part of the product usually occupies 46 bits or fewer, the exception being 47 bits when both operands are maximum negative. Since the low-order 32 bits of the product are stored unchanged, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow.

Divide

DR ***R_1, R_2*** **[*RR*]**



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



The dividend (first operand) is divided by the divisor (second operand) and replaced by the remainder and the quotient.

The dividend is a 64-bit signed integer and occupies the even/odd pair of registers specified by the R_1 field of the instruction. A specification exception occurs when R_1 is odd. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and odd-numbered registers, respectively. The divisor is a 32-bit signed integer.

The sign of the quotient is determined by the rules of algebra. The remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive. All operands and results are treated as signed integers. When the relative magnitude of dividend and divisor is such that the quotient cannot be expressed by a 32-bit signed integer, a fixed-point divide exception is recognized (a program interruption occurs, no division takes place, and the dividend remains unchanged in the general registers).

Condition Code: The code remains unchanged.

Program Interruptions:

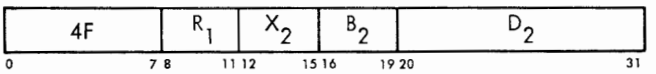
Protection (fetch violation by D only)
Addressing (D only)
Specification
Fixed-point divide

Programming Note

Division applies to fullword operands in storage only.

Convert to Binary

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



The radix of the second operand is changed from decimal to binary, and the result is placed in the first operand location. The number is treated as a right-aligned signed integer both before and after conversion.

The second operand has the packed decimal data format and is checked for valid sign and digit codes. Improper codes are a data exception and cause a program interruption. The decimal operand occupies a double-word storage field, which must be located on an integral boundary. The low-order four bits of the field represent the sign. The remaining 60 bits contain 15 binary-coded-decimal digits in true notation. The packed decimal data format is described under "Decimal Arithmetic."

The result of the conversion is placed in the general register specified by R_1. The maximum number that can be converted and still be contained in a 32-bit register is 2,147,483,647; the minimum number is −2,147,483,648. For any decimal number outside this range, the operation is completed by placing the 32 low-order binary bits in the register; a fixed-point divide exception exists, and a program interruption follows. In the case of a negative second operand, the low-order part is in two's-complement notation.

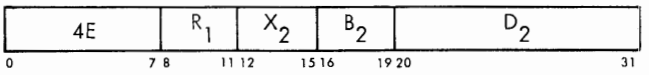
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (fetch violation)
Addressing
Specification
Data
Fixed-point divide

Convert to Decimal

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



The radix of the first operand is changed from binary to decimal, and the result is stored in the second operand location. The number is treated as a right-aligned signed integer both before and after conversion.

The result is placed in the storage location designated by the second operand and has the packed decimal format, as described in "Decimal Arithmetic." The result occupies a double-word in storage and must be located on an integral boundary. The low-order four bits of the field represent the sign. A positive sign is encoded as 1100 or 1010; a negative sign is encoded as 1101 or 1011. The choice between the two sign representations is determined by the state of PSW bit 12. The remaining 60 bits contain 15 binary-coded-decimal digits in true notation.

The number to be converted is obtained as a 32-bit signed integer from a general register. Since 15 decimal digits are available for the decimal equivalent of 31 bits, an overflow cannot occur.

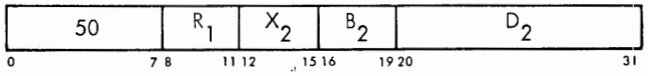
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store violation)
Addressing
Specification

Store

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



The first operand is stored at the second operand location.

The 32 bits in the general register are placed unchanged at the second operand location.

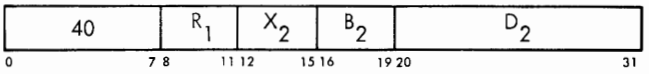
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store violation)
Addressing
Specification

Store Halfword

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



The first operand is stored at the halfword second operand location.

The 16 low-order bits in the general register are placed unchanged at the second operand location. The 16 high-order bits of the first operand do not participate and are not tested.

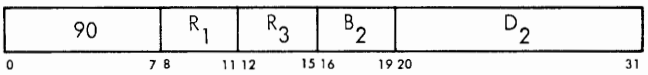
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store violation)
Addressing
Specification

Store Multiple

STM $R_1, R_3, D_2(B_2)$ [*RS*]



The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is stored at the locations designated by the second operand address.

The storage area where the contents of the general registers are placed starts at the location designated by the second operand address and continues through as many words as needed. The general registers are stored 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, with register 0 following register 15. The contents of the general registers remain unchanged.

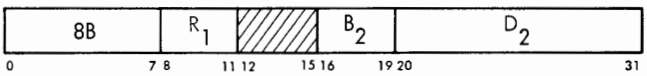
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store violation)
Addressing
Specification

Shift Left Single

SLA $R_1, D_2(B_2)$ [*RS*]



The integer part of the first operand is shifted left the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the left shift. Zeros are supplied to the vacated low-order register positions.

If a bit unlike the sign bit is shifted out of position 1, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

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

Program Interruptions:

Fixed-point overflow

Programming Note

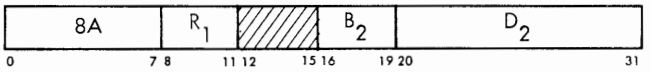
For numbers with an absolute value of less than 2^{30}, a left shift of one bit position is equivalent to multiplying the number by 2.

Shift amounts from 31-63 cause the entire integer to be shifted out of the register. When the entire integer field for a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -2^{31}.

The base register participating in the generation of the second operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

Shift Right Single

SRA ***R_1, $D_2(B_2)$*** ***[RS]***



The integer part of the first operand is shifted right the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the right shift. Bits equal to the sign are supplied to the vacated high-order bit positions. Low-order bits are shifted out without inspection and are lost.

Resulting Condition Code:

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

Program Interruptions:

None.

Programming Note

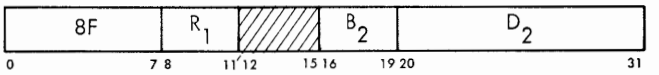
A right shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the value of the field is that obtained by dividing the value by 2. When an odd number is shifted right one position, the value of the field is that obtained by dividing the *next lower* number by 2. For example, +5 shifted right by one bit position yields +2, whereas −5 yields −3.

Shift amounts from 31-63 cause the entire integer to be shifted out of the register. When the entire integer field of a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of −1.

The base register participating in the generation of the second operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

Shift Left Double

SLDA ***R_1, $D_2(B_2)$*** ***[RS]***



The double-length integer part of the first operand is shifted left the number of bits specified by the second operand address.

The R_1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R_1 is odd.

The second operand address is not used to address data; its low-order 6-bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The operand is treated as a number with 63 integer bits and a sign in the sign position of the even register. The sign remains unchanged. The high-order position of the odd register contains an integer bit, and the content of the odd register participates in the shift in the same manner as the other integer bits. Zeros are supplied to the vacated positions of the registers.

If a bit unlike the sign bit is shifted out of bit position 1 of the even register, an overflow occurs. The

overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:

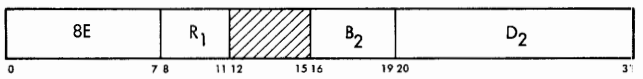
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

Specification
Fixed-point overflow

Shift Right Double

SRDA $R_1, D_2(B_2)$ [RS]



The double-length integer part of the first operand is shifted right the number of places specified by the second operand address.

The R_1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R_1 is odd.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The operand is treated as a number with 63 integer bits and a sign in the sign position of the even register. The sign remains unchanged. The high-order position of the odd register contains an integer bit, and the content of the odd register participates in the shift in the same manner as the other integer bits. The low-order bits are shifted out without inspection and are lost. Bits equal to the sign are supplied to the vacated positions of the registers.

Resulting Condition Code:

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

Program Interruptions:

Specification

Programming Note

A zero shift amount in the double-shift operations provides a double-length sign and magnitude test.

Fixed-Point Arithmetic Exceptions

Exceptional operand designations, data, or results cause a program interruption. When a program interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in fixed-point arithmetic.

Protection: The key of an operand in storage does not match the protection key in the PSW. The operation is suppressed for a store violation. Therefore, the condition code and data in registers and storage remain unchanged. The only exception is STORE MULTIPLE, which is terminated; the amount of data stored is unpredictable and should not be used for further computation. The operation is terminated on any fetch violation.

Addressing: An address designates an operand location outside the available storage for a particular installation. In most cases, the operation is terminated. Therefore, the result data are unpredictable and should not be used for further computation. The exceptions are STORE, STORE HALFWORD, and CONVERT TO DECIMAL, which are suppressed. Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. The address restrictions do not apply to the components from which an address is generated – the content of the D_2 field and the contents of the registers specified by X_2 and B_2.

Specification: A double-word operand is not located on a 64-bit boundary, a fullword operand is not located on a 32-bit boundary, a halfword operand is not located on a 16-bit boundary, or an instruction specifies an odd register address for a pair of general registers containing a 64-bit operand. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Data: A sign or a digit code of the decimal operand in CONVERT TO BINARY is incorrect. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Fixed-Point Overflow: The result of a sign-control, add, subtract, or shift operation overflows. The interruption occurs only when the fixed-point overflow mask bit is one. The operation is completed by placing the truncated low-order result in the register and setting the condition code to 3. The overflow bits are lost. In add-type operations the sign stored in the register is the opposite of the sign of the sum or difference. In shift operations the sign of the shifted number remains unchanged. The state of the mask bit does not affect the result.

Fixed-Point Divide: The quotient of a division exceeds the register size, including division by zero, or the result in CONVERT TO BINARY exceeds 31 bits. Division is suppressed. Therefore, data in the registers remain unchanged. The conversion is completed by recording the truncated low-order result in the register.

Decimal arithmetic operates on data in the packed format. In this format, two decimal digits are placed in one eight-bit byte. Since data are often communicated to or from external devices in the zoned format (which has one digit in an eight-bit byte), the necessary format-conversion operations are also provided in this instruction group.

Data are interpreted as integers, right-aligned in their fields. They are kept in true notation with a sign in the low-order eight-bit byte.

Processing takes place right to left between main-storage locations. All decimal arithmetic instructions use a two-address format. Each address specifies the leftmost byte of an operand. Associated with this address is a length field, indicating the number of additional bytes that the operand extends beyond the first byte.

The decimal arithmetic instruction set provides for adding, subtracting, comparing, multiplying, and dividing, as well as the format conversion of variable-length operands. All of the instructions discussed in this section except PACK, UNPACK, and MOVE WITH OFFSET are part of the decimal feature.

The condition code is set as a result of all add-type and comparison operations.

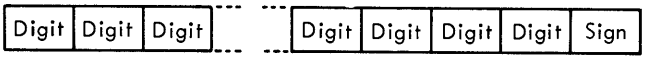
Data Format

Decimal operands reside in main storage only. They occupy fields that may start at any byte address and are composed of one to 16 eight-bit bytes.

Lengths of the two operands specified in an instruction need not be the same. If necessary they are considered to be extended with zeros to the left of the high-order digits. Results never exceed the limits set by address and length specification. Lost carries or lost digits from arithmetic operations are signaled as a decimal overflow exception.

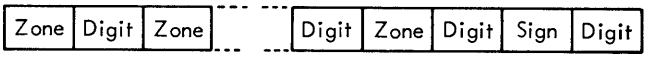
Although decimal arithmetic is performed on data in the packed format, decimal operands may be either in the packed or zoned format.

Packed Decimal Number



In the packed format, two decimal digits normally are placed adjacent in a byte, except for the rightmost byte of the field. In the rightmost byte a sign is placed to the right of decimal digit. Both digits and a sign are encoded and occupy four bits each.

Zoned Decimal Number



In the zoned format the low-order four bits of a byte, the *numeric*, are normally occupied by a decimal digit. The four high-order bits of a byte are called the *zone*, except for the rightmost byte of the field, where normally the sign occupies the zone position.

Arithmetic is performed with operands in the packed format and results in the packed format. In the zoned format, the digits are represented as part of an alphameric character set. A PACK instruction is provided to transform zoned data into packed data, and an UNPACK instruction performs the reverse transformation. Moreover, the editing instructions may be used to change data from packed to zoned.

The fields specified in decimal arithmetic other than in PACK, UNPACK, and MOVE WITH OFFSET either should not overlap at all or should have coincident rightmost bytes. In ZERO AND ADD, the destination field may also overlap to the right of the source field. Because the code configurations for digits and sign are verified during arithmetic, improper overlapping fields are recognized as data exceptions. In move-type operations, the operand digits and signs are not checked, and the operand fields may overlap without any restrictions.

The rules for overlapped fields are established for the case where operands are fetched right to left from storage, eight bits at a time, just before they are processed. Similarly, the results are placed in storage, eight bits at a time, as soon as they are generated. Actual processing procedure may be considerably different because of the use of preferred storage for intermediate results. Nevertheless, the same rules are observed.

Number Representation

Numbers are represented as right-aligned true integers with a plus or minus sign.

The digits 0-9 have the binary encoding 0000-1001. The codes 1010-1111 are invalid as digits. This set of

codes is interpreted as sign codes, with 1010, 1100, 1110, and 1111 recognized as plus and with 1011 and 1101 recognized as minus. The codes 0000-1001 are invalid as sign codes. The zones are not tested for valid codes inasmuch as they are eliminated in changing data from the zoned to the packed format.

The sign and zone codes generated for all decimal arithmetic results differ for the extended binary-coded-decimal interchange code (EBCDIC) and the USA Standard Code for Information Interchange (USASCII-8). The choice between the two codes is determined by bit 12 of the PSW. When bit 12 is zero, the preferred EBCDIC codes are generated; these are plus, 1100; minus, 1101; and zone, 1111. When bit 12 is one, the preferred USASCII-8 codes are generated; these are plus, 1010; minus, 1011; and zone, 0101.

Condition Code

The results of all add-type and comparison operations are used to set the condition code. All other decimal arithmetic operations leave the code unchanged. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect two types of results for decimal arithmetic. For most operations the states 0, 1, and 2 indicate a zero, less than zero, and greater than zero content of the result field; the state 3 is used when the result of the operation overflows.

For the comparison operation, the states 0, 1, and 2 indicate that the first operand compared equal, low, or high.

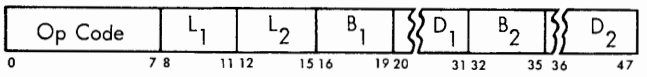
CONDITION CODE SETTING FOR DECIMAL ARITHMETIC

	0	1	2	3
Add Decimal	zero	< zero	> zero	overflow
Compare Decimal	equal	low	high	--
Subtract Decimal	zero	< zero	> zero	overflow
Zero and Add	zero	< zero	> zero	overflow

Instruction Format

Decimal instructions use the following format:

SS Format



For this format, the content of the general register specified by B_1 is added to the content of the D_1 field to form an address. This address specifies the leftmost byte of the first operand field. The number of operand bytes to the right of this byte is specified by the L_1 field of the instruction. Therefore, the length in bytes of the first operand field is 1-16, corresponding to a length code in L_1 of 0000-1111. The second operand field is specified similarly by the L_2, B_2, and D_2 instruction fields.

A zero in the B_1 or B_2 field indicates the absence of the corresponding address component.

Results of operations are always placed in the first operand field. The result is never stored outside the field specified by the address and length. In the event the first operand is longer than the second, the second operand is extended with high-order zeros up to the length of the first operand. Such extension never modifies storage. The second operand field and the contents of all general registers remain unchanged.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction. For ADD DECIMAL, for example, AP is the mnemonic and D_1 (L_1, B_1), D_2 (L_2, B_2) the operand designation.

Instructions

The decimal arithmetic instructions and their mnemonics and operation codes follow. All instructions use the ss format and assume packed operands and results. The only exceptions are PACK, which has a zoned operand, and UNPACK, which has a zoned result. The table indicates when the condition code is set and the exceptions in operand designations, data, or results that cause a program interruption.

NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Add Decimal	AP	SS T,C	P,A, D,DF	FA
Subtract Decimal	SP	SS T,C	P,A, D,DF	FB
Zero and Add	ZAP	SS T,C	P,A, D,DF	F8
Compare Decimal	CP	SS T,C	P,A, D	F9
Multiply Decimal	MP	SS T	P,A,S,D	FC
Divide Decimal	DP	SS T	P,A,S,D,DK	FD
Pack	PACK	SS	P,A	F2
Unpack	UNPK	SS	P,A	F3
Move with Offset	MVO	SS	P,A	F1

NOTES

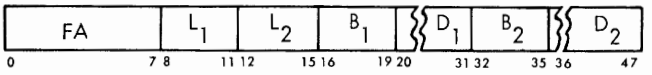
A	Addressing exception
C	Condition code is set
D	Data exception
DF	Decimal-overflow exception
DK	Decimal-divide exception
P	Protection exception
S	Specification exception
T	Decimal feature

Programming Note

The moving, editing, and logical comparing instructions may also be used in decimal calculations.

Add Decimal

AP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



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

Addition is algebraic, taking into account sign and all digits of both operands. All signs and digits are checked for validity. If necessary, high-order zeros are supplied for either operand. When the first operand field is too short to contain all significant digits of the sum, a decimal overflow occurs, and a program interruption is taken provided that the corresponding mask bit is one.

Overflow has two possible causes. The first is the loss of a carry out of the high-order digit position of the result field. The second cause is an oversized result, which occurs when the second operand field is larger than the first operand field and significant result digits are lost. The field sizes alone are not an indication of overflow.

The first and second operand fields may overlap when their low-order bytes coincide; therefore, it is possible to add a number to itself.

The sign of the sum is determined by the rules of algebra. When the operation is completed without an overflow, a zero sum has a positive sign, but when high-order digits are lost because of an overflow, a zero sum may be either positive or negative, as determined by what the sign of the correct sum would have been.

Resulting Condition Code:

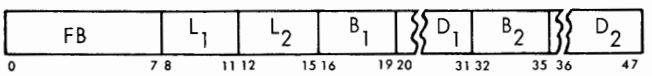
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Interruptions:

Operation (if decimal feature is not installed)
Protection (store or fetch violation)
Addressing
Data
Decimal overflow

Subtract Decimal

SP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The second operand is subtracted from the first operand, and the difference is placed in the first operand location.

Subtraction is algebraic, taking into account sign and all digits of both operands. The SUBTRACT DECIMAL is similar to ADD DECIMAL, except that the sign of the second operand is changed from positive to negative or from negative to positive after the operand is obtained from storage and before the arithmetic.

The sign of the difference is determined by the rules of algebra. When the operation is completed without an overflow, a zero difference has a positive sign, but when high-order digits are lost because of an overflow, a zero difference may be either positive or negative, as determined by what the sign of the correct difference would have been.

Resulting Condition Code:

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

Program Interruptions:

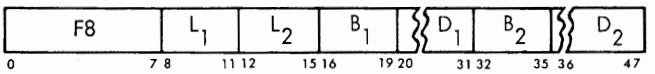
Operation (if decimal feature is not installed)
Protection (store or fetch violation)
Addressing
Data
Decimal overflow

Programming Note

The operands of SUBTRACT DECIMAL may overlap when their low-order bytes coincide, even when their lengths are unequal. This property may be used to set to zero an entire field or the low-order part of a field.

Zero and Add

ZAP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The second operand is placed in the first operand location.

The operation is equivalent to an addition to zero. A zero result is positive. When high-order digits are lost because of overflow, a zero result has the sign of the second operand.

Only the second operand is checked for valid sign and digit codes. Extra high-order zeros are supplied if needed. When the first operand field is too short to contain all significant digits of the second operand, a decimal overflow occurs and results in a program interruption, provided that the decimal overflow mask bit is one. The first and second operand fields may overlap when the rightmost byte of the first operand field is coincident with or to the right of the rightmost byte of the second operand.

Resulting Condition Code:

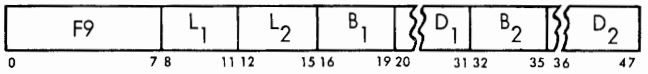
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

Operation (if decimal feature is not installed)
Protection (store or fetch violation)
Addressing
Data
Decimal overflow

Compare Decimal

CP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The first operand is compared with the second, and the condition code indicates the comparison result.

Comparison is right to left, taking into account the sign and all digits of both operands. All signs and digits are checked for validity, and any valid plus or minus sign is considered equal to any other valid plus or minus sign, respectively. If the fields are unequal in length, the shorter is extended with high-order zeros. A field with a zero value and positive sign is considered equal to a field with a zero value but negative sign. Neither operand is changed as a result of the operation. Overflow cannot occur in this operation.

The first and second fields may overlap when their low-order bytes coincide. It is possible, therefore, to compare a number to itself.

Resulting Condition Code:

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

Program Interruptions:

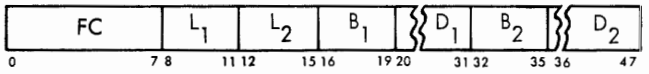
Operation (if decimal feature is not installed)
Protection (fetch violation)
Addressing
Data

Programming Note

The COMPARE DECIMAL is unique in processing from right to left; taking signs, zeros, and invalid characters into account; and extending variable-length fields when they are unequal in length.

Multiply Decimal

MP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

The multiplier size is limited to 15 digits and sign and must be less than the multiplicand size. Length code L_2, larger than seven, or larger than or equal to the length code L_1, is recognized as a specification exception. The operation is suppressed and a program interruption occurs.

Since the number of digits in the product is the sum of the number of digits in the operands, the multiplicand must have high-order zero digits for at least a field size that equals the multiplier field size; otherwise, a data exception is recognized, and a program interruption occurs. This definition of the multiplicand field insures that no product overflow can occur. The maximum product size is 31 digits. At least one high-order digit of the product field is zero.

All operands and results are treated as signed integers, right-aligned in their field. The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs, even if one or both operands are zero.

The multiplier and product fields may overlap when their low-order bytes coincide.

Condition Code: The code remains unchanged.

Program Interruptions:

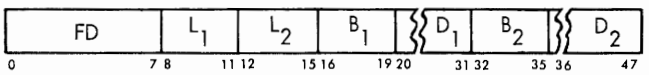
Operation (if decimal feature is not installed)
Protection (store or fetch violation)
Addressing
Specification
Data

Programming Note

When the multiplicand does not have the desired number of leading zeros, multiplication may be preceded by a ZERO AND ADD into a larger field.

Divide Decimal

DP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The dividend (the first operand) is divided by the divisor (the second operand) and replaced by the quotient and remainder.

The quotient field is placed leftmost in the first operand field. The remainder field is placed rightmost in the first operand field and has a size equal to the divisor size. Together, the quotient and remainder occupy the entire dividend field; therefore, the address of the quotient field is the address of the first oper-

and. The size of the quotient field in eight-bit bytes is $L_1 - L_2$, and the length code for this field is one less ($L_1 - L_2 - 1$). When the divisor length code is larger than seven (15 digits and sign) or larger than or equal to the dividend length code, a specification exception is recognized. The operation is suppressed, and a program interruption occurs.

The dividend, divisor, quotient, and remainder are all signed integers, right-aligned in their fields. The sign of the quotient is determined by the rules of algebra from dividend and divisor signs. The sign of the remainder has the same value as the dividend sign. These rules are true even when quotient or remainder is zero.

Overflow cannot occur. A quotient larger than the number of digits allowed is recognized as a decimal-divide exception. The operation is suppressed, and a program interruption occurs. Divisor and dividend remain unchanged in their storage locations.

The divisor and dividend fields may overlap only if their low-order bytes coincide.

Condition Code: The code remains unchanged.

Program Interruptions:

Operation (if decimal feature is not installed)
Protection (store or fetch violation)
Addressing
Specification
Data
Decimal divide

Programming Note

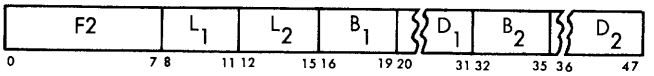
The maximum dividend size is 31 digits and sign. Since the smallest remainder size is one digit and sign, the maximum quotient size is 29 digits and sign.

The condition for a divide exception can be determined by a trial subtraction. The leftmost digit of the divisor field is aligned with the leftmost-less-one digit of the dividend field. When the divisor, so aligned, is less than or equal to the dividend, a divide exception is indicated.

A decimal-divide exception occurs if the dividend does not have at least one leading zero.

Pack

PACK $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The format of the second operand is changed from zoned to packed, and the result is placed in the first operand location.

The second operand is assumed to have the zoned format. All zones are ignored, except the zone over the low-order digit, which is assumed to represent a sign. The sign is placed in the right four bits of the low-order byte, and the digits are placed adjacent to the sign and to each other in the remainder of the result field. The sign and digits are moved unchanged to the first operand field and are not checked for valid codes.

The fields are processed right to left. If necessary, the second operand is extended with high-order zeros. If the first operand field is too short to contain all significant digits of the second operand field, the remaining high-order digits are ignored. Overlapping fields may occur and are processed by storing one result byte immediately after the necessary operand bytes are fetched. Except for the rightmost byte of the result field, which is stored immediately upon fetching the first operand byte, two operand bytes are needed for each result byte.

Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store or fetch violation)
Addressing

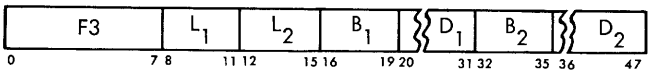
Programming Notes

The PACK instruction may be used to switch the two digits in one byte by specifying a zero in the L_1 and L_2 fields and the same address for both operands.

To remove the zones of all bytes of a field, including the low-order byte, both operands must be extended with a dummy byte in the low-order position, which subsequently is ignored in the result field.

Unpack

UNPK $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The format of the second operand is changed from packed to zoned, and the result is placed in the first operand location.

The digits and sign of the packed operand are placed unchanged in the first operand location, using the zoned format. Zones with coding 1111 in EBCDIC and coding 0101 in USASCII-8 are supplied for all bytes, except the low-order byte, which receives the sign of the packed operand. The operand sign and digits are not checked for valid codes.

The fields are processed right to left. The second operand is extended with high-order zero digits before unpacking, if necessary. If the first operand field is too short to contain all significant digits of the second

operand, the remaining high-order digits are ignored. The first and second operand fields may overlap and are processed by storing the first result byte immediately after the rightmost operand byte is fetched; for the remaining operand bytes, two result bytes are stored immediately after one byte is fetched.

Condition Code: The code remains unchanged.

Program Interruptions:

Addressing

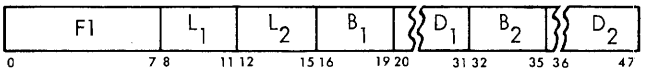
Protection (store or fetch violation)

Programming Note

A field that is to be unpacked can be destroyed by improper overlapping. If it is desired to save storage space for unpacking by overlapping the operand fields, the low-order position of the first operand must be to the right of the low-order position of the second operand by the number of bytes in the second operand minus two. If only one or two bytes are to be unpacked, the low-order positions of the two operands may coincide.

Move with Offset

MVO $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The second operand is placed to the left of and adjacent to the low-order four bits of the first operand.

The low-order four bits of the first operand are attached as low-order bits to the second operand, the second operand bits are offset by four bit positions, and the result is placed in the first operand location. The first and second operand bytes are not checked for valid codes.

The fields are processed right to left. If necessary, the second operand is extended with high-order zeros. If the first operand field is too short to contain all bytes of the second operand, the remaining information is ignored. Overlapping fields may occur and are processed by storing a result byte as soon as the necessary operand bytes are fetched.

Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store or fetch violation)

Addressing

Programming Note

The instruction set for decimal arithmetic includes no shift instructions since the equivalent of a shift can be obtained by programming. Programs for right or left shift and for an even or odd shift amount may be written with MOVE WITH OFFSET and the logical move instructions.

Decimal Arithmetic Exceptions

Exceptional operation codes, operand designations, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in decimal arithmetic.

Operation: The decimal feature is not installed and the instruction is ADD DECIMAL, SUBTRACT DECIMAL, ZERO AND ADD, COMPARE DECIMAL, MULTIPLY DECIMAL, or DIVIDE DECIMAL. The instruction is suppressed. Therefore, the condition code and data in storage and registers remain unchanged.

Protection: The key of an operand in storage does not match the protection key in the PSW.

The operation is terminated for either a store or a fetch violation by a decimal instruction; the result data and condition code are unpredictable.

Addressing: An address designates an operand location outside the available storage for the installation.

The operation is terminated. The result data and the condition code are unpredictable and should not be used for further computation.

These address exceptions do not apply to the components from which an address is generated — the contents of the D_1 and D_2 fields and the contents of the registers specified by B_1 and B_2.

Specification: A multiplier or a divisor size exceeds 15 digits and sign or is equal to or exceeds the multiplicand or dividend size. The instruction is suppressed; therefore, the condition code and data in storage and registers remain unchanged.

Data: A sign or digit code of an operand in ADD DECIMAL, SUBTRACT DECIMAL, ZERO AND ADD, COMPARE DECIMAL, MULTIPLY DECIMAL, or DIVIDE DECIMAL is incorrect, a multiplicand has insufficient high-order zeros, or the operand fields in these operations overlap incorrectly. The operation is terminated. The result data and the condition code are unpredictable and should not be used for further computation.

Decimal Overflow: The result of ADD DECIMAL, SUBTRACT DECIMAL, or ZERO AND ADD overflows. The program interruption occurs only when the decimal-overflow mask bit is one. The operation is completed by placing the truncated low-order result in the result field and setting the condition code to 3. The sign and low-order digits contained in the result field are the same as they would have been for an infinitely long result field.

Decimal Divide: The quotient exceeds the specified data field, including division by zero. Division is suppressed. Therefore, the dividend and divisor remain unchanged in storage.

The floating-point instruction set is used to perform calculations on operands with a wide range of magnitude and yielding results scaled to preserve precision.

A floating-point number consists of a signed exponent and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess 64 binary notation; the fraction is expressed as a hexadecimal number having a radix point to the left of the high-order digit.

To avoid unnecessary storing and loading operations for results and operands, four floating-point registers are provided. The floating-point instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, and storing, as well as the sign control of short or long operands. Short operands generally provide faster processing and require less storage than long operands. On the other hand, long operands provide greater preciseness in computation. Operations may be either register to register or storage to register. All floating-point instructions and registers are part of the floating-point feature.

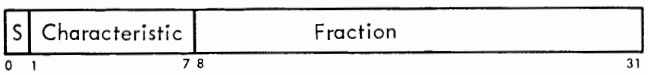
Maximum precision is preserved in addition, subtraction, multiplication, and division by producing normalized results. For addition and subtraction, instructions are also provided that generate unnormalized results. Normalized and unnormalized operands may be used in any floating-point operation.

The condition code is set as a result of all sign control, add, subtract, and compare operations.

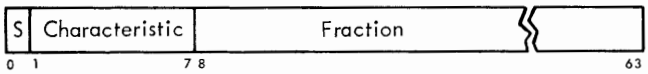
Data Format

Floating-point data occupy a fixed-length format, which may be either a fullword short format or a double-word long format. Both formats may be used in main storage and in the floating-point registers. The floating-point registers are numbered 0, 2, 4, and 6.

Short Floating-Point Number



Long Floating-Point Number



The first bit in either format is the sign bit (S). The subsequent seven bit positions are occupied by the characteristic. The fraction field may have either six or 14 hexadecimal digits.

The entire set of floating-point instructions is available for both short and long operands. When short-precision is specified, all operands and results are 32-bit floating-point words, and the rightmost 32 bits of the floating-point registers do not participate in the operations and remain unchanged. An exception is the product in MULTIPLY, which is a 64-bit word and occupies a full register. When long-precision is specified, all operands and results are 64-bit floating-point words.

Although final results have six fraction digits in short-precision and 14 fraction digits in long-precision, intermediate results in ADD NORMALIZED, SUBTRACT NORMALIZED, ADD UNNORMALIZED, SUBTRACT UNNORMALIZED, COMPARE, HALVE, and MULTIPLY may have one additional low-order digit. This low-order digit, the guard digit, increases the precision of the final result.

Number Representation

The fraction of a floating-point number is expressed in hexadecimal digits. The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for the floating-point number, the fraction is considered to be multiplied by a power of 16. The characteristic portion, bits 1-7 of both floating-point formats, indicates this power. The bits within the characteristic field can represent numbers from 0 through 127. To accommodate large and small magnitudes, the characteristic is formed by adding 64 to the actual exponent. The range of the exponent is thus -64 through $+63$. This technique produces a characteristic in excess 64 notation.

Both positive and negative quantities have a true fraction, the difference in sign being indicated by the sign bit. The number is positive or negative accordingly as the sign bit is zero or one.

The range covered by the magnitude (M) of a normalized floating-point number is
in short precision $16^{-65} \leq M \leq (1 - 16^{-6}) \cdot 16^{63}$, and
in long precision $16^{-65} \leq M \leq (1 - 16^{-14}) \cdot 16^{63}$,
or approximately $5.4 \cdot 10^{-79} \leq M \leq 7.2 \cdot 10^{75}$
in either precision.

A number with zero characteristic, zero fraction, and plus sign is called a true zero. A true zero may arise as the result of an arithmetic operation because of the particular magnitude of the operands. A result is forced to be true zero when (1) an exponent underflow occurs and the exponent-underflow mask (PSW bit 38) is zero, (2) a result fraction of an addition or subtraction operation is zero and the significance mask (PSW bit 39) is zero, or (3) the operand of HALVE, one or both operands of MULTIPLY, or the dividend in DIVIDE has a zero fraction. When a program interruption due to exponent underflow occurs, a true zero fraction is not forced; instead, the fraction and sign remain correct, and the characteristic is 128 too large. When a program interruption due to lost significance occurs, the fraction remains zero, and the fraction sign and characteristic remain correct. Whenever a result has a zero fraction, the exponent overflow and underflow exceptions do not cause a program interruption. When a divisor has a zero fraction, division is omitted, a floating-point divide exception exists, and a program interruption occurs. In addition and subtraction, an operand with a zero fraction or characteristic participates as a normal number.

The sign of a sum, difference, product, or quotient with zero fraction is positive. The sign of a zero fraction resulting from other operations is established by the rules of algebra from the operand signs.

Normalization

A quantity can be represented with the greatest precision by a floating-point number of given fraction length when that number is normalized. A normalized floating-point number has a nonzero high-order hexadecimal fraction digit. If one or more high-order fraction digits are zero, the number is said to be unnormalized. The process of normalization consists of shifting the fraction left until the high-order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A zero fraction can not be normalized, and its associated characteristic therefore remains unchanged when normalization is called for.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result. This function is called *postnormalization*. In performing multiplication and division, the operands are normalized prior to the arithmetic process. This function is called *prenormalization*.

Floating-point operations may be performed with or without normalization. Most operations are performed in only one of these two ways. Addition and subtraction may be specified either way.

When an operation is performed without normalization, high-order zeros in the result fraction are not eliminated. The result may or may not be normalized, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. Also, intermediate fraction results are shifted right when an overflow occurs, and the intermediate fraction result is truncated to the final result length after the shifting, if any.

Programming Note

Since normalization applies to hexadecimal digits, the three high-order bits of a normalized number may be zero.

Condition Code

The results of floating-point sign-control, add, subtract, and compare operations are used to set the condition code. Multiplication, division, loading, and storing leave the code unchanged. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect two types of results for floating-point arithmetic. For most operations, the states 0, 1, or 2 indicate that the result is zero, less than zero, or greater than zero. A zero result is indicated whenever the result fraction is zero, including a forced zero. State 3 is never set by floating-point operations.

For comparison, the states 0, 1, or 2 indicate that the first operand is equal, low, or high.

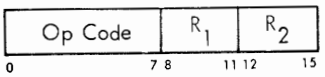
CONDITION CODE SETTING FOR FLOATING-POINT ARITHMETIC

	0	1	2	3
Add Normalized s/L	zero	< zero	> zero	--
Add Unnormalized s/L	zero	< zero	> zero	--
Compare s/L	equal	low	high	--
Load and Test s/L	zero	< zero	> zero	--
Load Complement s/L	zero	< zero	> zero	--
Load Negative s/L	zero	< zero	--	--
Load Positive s/L	zero	--	> zero	--
Subtract Normalized s/L	zero	< zero	> zero	--
Subtract Unnormalized s/L	zero	< zero	> zero	--

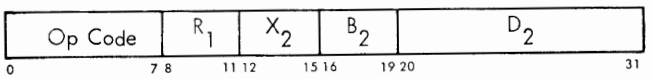
Instruction Format

Floating-point instructions use the following two formats:

RR Format



RX Format



In these formats, R_1 designates the address of a floating-point register. The contents of this register will be

called the first operand. The second operand location is defined differently for two formats.

In the RR format, the R_2 field specifies the address of a floating-point register containing the second operand. The same register may be specified for the first and second operand.

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

A zero in an X_2 or B_2 field indicates the absence of the corresponding address component.

The register address specified by the R_1 and R_2 fields should be 0, 2, 4 or 6. Otherwise, a specification exeception is recognized, and a program interruption is caused.

The storage address of the second operand should designate word boundaries for short operands and double-word boundaries for long operands. Otherwise, a specification exception is recognized, and a program interruption is caused.

Results replace the first operand, except for the storing operations, where the second operand is replaced.

Except for the storing of the final result, the contents of all floating-point or general registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

The floating-point instructions are the only instructions using the floating-point registers.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction. For a register-to-register operation using LOAD (short), for example, LER is the mnemonic and R_1, R_2 the operand designation.

Instructions

The floating-point arithmetic instructions and their mnemonics, formats, and operation codes follow. All operations can be specified in short and long precision and are part of the floating-point feature. The following table indicates when the condition code is set and the exceptions in operand designations, data, or results that cause a program interruption.

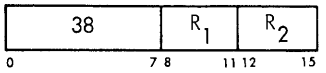
NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Load (Long)	LDR	RR F	S	28
Load (Long)	LD	RX F	P,A,S	68
Load (Short)	LER	RR F	S	38
Load (Short)	LE	RX F	P,A,S	78
Load and Test (Long)	LTDR	RR F,C	S	22
Load and Test (Short)	LTER	RR F,C	S	32
Load Complement (Long)	LCDR	RR F,C	S	23
Load Complement (Short)	LCER	RR F,C	S	33
Load Positive (Long)	LPDR	RR F,C	S	20
Load Positive (Short)	LPER	RR F,C	S	30
Load Negative (Long)	LNDR	RR F,C	S	21
Load Negative (Short)	LNER	RR F,C	S	31
Add Normalized (Long)	ADR	RR F,C	S,U,E,LS	2A
Add Normalized (Long)	AD	RX F,C	P,A,S,U,E,LS	6A
Add Normalized (Short)	AER	RR F,C	S,U,E,LS	3A
Add Normalized (Short)	AE	RX F,C	P,A,S,U,E,LS	7A
Add Unnormalized (Long)	AWR	RR F,C	S, E,LS	2E
Add Unnormalized (Long)	AW	RX F,C	P,A,S, E,LS	6E
Add Unnormalized (Short)	AUR	RR F,C	S, E,LS	3E
Add Unnormalized (Short)	AU	RX F,C	P,A,S, E,LS	7E
Subtract Normalized (Long)	SDR	RR F,C	S,U,E,LS	2B
Subtract Normalized (Long)	SD	RX F,C	P,A,S,U,E,LS	6B
Subtract Normalized (Short)	SER	RR F,C	S,U,E,LS	3B
Subtract Normalized (Short)	SE	RX F,C	P,A,S,U,E,LS	7B
Subtract Unnormalized (Long)	SWR	RR F,C	S, E,LS	2F
Subtract Unnormalized (Long)	SW	RX F,C	P,A,S, E,LS	6F
Subtract Unnormalized (Short)	SUR	RR F,C	S, E,LS	3F
Subtract Unnormalized (Short)	SU	RX F,C	P,A,S, E,LS	7F
Compare (Long)	CDR	RR F,C	S	29
Compare (Long)	CD	RX F,C	P,A,S	69
Compare (Short)	CER	RR F,C	S	39
Compare (Short)	CE	RX F,C	P,A,S	79
Halve (Long)	HDR	RR F	S,U	24
Halve (Short)	HER	RR F	S,U	34
Multiply (Long)	MDR	RR F	S,U,E	2C
Multiply (Long)	MD	RX F	P,A,S,U,E	6C
Multiply (Short)	MER	RR F	S,U,E	3C
Multiply (Short)	ME	RX F	P,A,S,U,E	7C
Divide (Long)	DDR	RR F	S,U,E,FK	2D
Divide (Long)	DD	RX F	P,A,S,U,E,FK	6D
Divide (Short)	DER	RR F	S,U,E,FK	3D
Divide (Short)	DE	RX F	P,A,S,U,E,FK	7D
Store (Long)	STD	RX F	P,A,S	60
Store (Short)	STE	RX F	P,A,S	70

NOTES

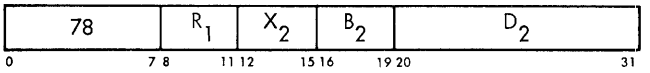
A	Addressing exception
C	Condition code is set
E	Exponent-overflow exception
F	Floating-point feature
FK	Floating-point divide exception
LS	Significance exception
P	Protection exception
S	Specification exception
U	Exponent-underflow exception

Load

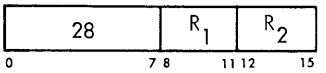
LER R_1, R_2 **[*RR, Short Operands*]**



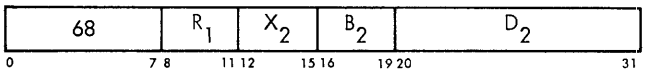
LE $R_1, D_2(X_2, B_2)$ **[*RX, Short Operands*]**



LDR R_1, R_2 **[*RR, Long Operands*]**



LD $R_1, D_2(X_2, B_2)$ **[*RX, Long Operands*]**



The second operand is placed in the first operand location.

The second operand is not changed. In short-precision the low-order half of the result register remains unchanged. Exponent overflow, exponent underflow, or lost significance cannot occur.

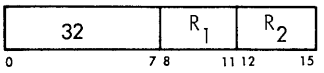
Condition Code: The code remains unchanged.

Program Interruptions:

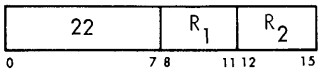
Operation (if floating-point feature is not installed)
Protection (fetch violation by LE and LD only)
Addressing (LE, LD only)
Specification

Load and Test

LTER R_1, R_2 **[*RR, Short Operands*]**



LTDR R_1, R_2 **[*RR, Long Operands*]**



The second operand is placed in the first operand location, and its sign and magnitude determine the condition code.

The second operand is not changed. In short-precision the low-order half of the result register remains unchanged and is not tested.

Resulting Condition Code:

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

Program Interruptions:

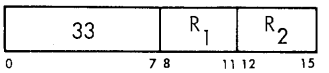
Operation (if floating-point feature is not installed)
Specification

Programming Note

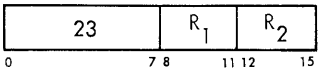
When the same register is specified as first and second operand location, the operation is equivalent to a test without data movement.

Load Complement

LCER R_1, R_2 **[*RR, Short Operands*]**



LCDR R_1, R_2 **[*RR, Long Operands*]**



The second operand is placed in the first operand location with the sign changed to the opposite value.

The sign bit of the second operand is inverted, while characteristic and fraction are not changed. In short-precision the low-order half of the result register remains unchanged and is not tested.

Resulting Condition Code:

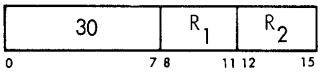
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

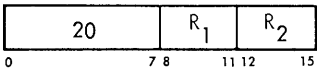
Operation (if floating-point feature is not installed)
Specification

Load Positive

LPER R_1, R_2 **[*RR, Short Operands*]**



LPDR R_1, R_2 **[*RR, Long Operands*]**



The second operand is placed in the first operand location with the sign made plus.

The sign bit of the second operand is made zero, while characteristic and fraction are not changed. In short-precision, the low-order half of the result register remains unchanged and is not tested.

Resulting Condition Code:

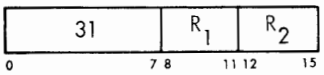
0 Result fraction is zero
1 --
2 Result is greater than zero
3 --

Program Interruptions:

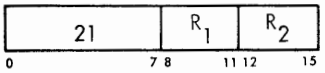
Operation (if floating-point feature is not installed)
Specification

Load Negative

LNER ***R_1, R_2*** **[*RR, Short Operands*]**



LNDR ***R_1, R_2*** **[*RR, Long Operands*]**



The second operand is placed in the first operand location with the sign made minus.

The sign bit of the second operand is made one, even if the fraction is zero. Characteristic and fraction are not changed. In short-precision, the low-order half of the result register remains unchanged and is not tested.

Resulting Condition Code:

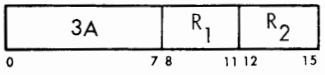
0 Result fraction is zero
1 Result is less than zero
2 --
3 --

Program Interruptions:

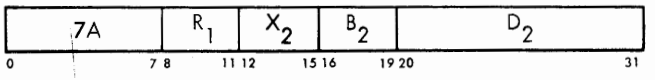
Operation (if floating-point feature is not installed)
Specification

Add Normalized

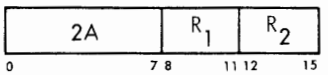
AER ***R_1, R_2*** **[*RR, Short Operands*]**



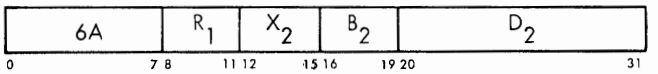
AE ***R_1, $D_2(X_2, B_2)$*** **[*RX, Short Operands*]**



ADR ***R_1, R_2*** **[*RR, Long Operands*]**



AD ***R_1, $D_2(X_2, B_2)$*** **[*RX, Long Operands*]**



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

In short-precision the low-order halves of the floating-point registers are ignored and remain unchanged.

Addition of two floating-point numbers consists of a characteristic comparison and a fraction addition. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted; its characteristic is increased by one for each hexadecimal digit of shift, until the two characteristics agree. The fractions are then added algebraically to form an intermediate sum. If an overflow carry occurs, the intermediate sum is right-shifted one digit, and the characteristic is increased by one. If this increase causes a characteristic overflow, an exponent-overflow exception is signaled, and a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is 128 smaller than the correct characteristic.

The short intermediate sum consists of 7 hexadecimal digits and a possible carry. The long intermediate sum consists of 15 hexadecimal digits and a possible carry. The low-order digit is a guard digit obtained from the fraction which is shifted right. Only one guard digit position participates in the fraction addition. The guard digit is zero if no shift occurs.

After the addition, the intermediate sum is left-shifted as necessary to form a normalized fraction; vacated low-order digit positions are filled with zeros; the characteristic is reduced by the amount of shift.

If normalization causes the characteristic to underflow and if the corresponding mask bit is one, a program interruption occurs. The fraction is correct and normalized, the sign is correct, and the characteristic is 128 larger than the correct one. If the corresponding mask bit is zero, the result is made a true zero. If no left shift takes place, the intermediate sum is truncated to the proper fraction length.

When the intermediate sum is zero and the significance mask bit is one, a significance exception exists, and a program interruption takes place. No normalization occurs; the intermediate sum characteristic remains unchanged. When the intermediate sum is zero and the significance mask bit is zero, the program

interruption for the significance exception does not occur; rather, the characteristic is made zero, yielding a true zero result. Exponent underflow does not occur for a zero fraction.

The sign of the sum is derived by the rules of algebra. The sign of a sum with zero result fraction is always positive.

Resulting Condition Code:

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

Program Interruptions:

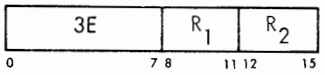
Operation (if floating-point feature is not installed)
Protection (fetch violation by AE and AD only)
Addressing (AE and AD only)
Specification
Significance
Exponent overflow
Exponent underflow

Programming Note

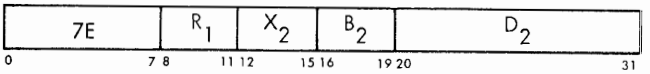
Interchanging the two operands in a floating-point addition does not affect the value of the sum.

Add Unnormalized

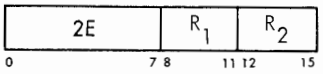
AUR R_1, R_2 **[*RR, Short Operands*]**



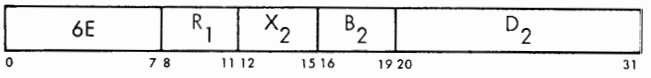
AU $R_1, D_2(X_2, B_2)$ **[*RX, Short Operands*]**



AWR R_1, R_2 **[*RR, Long Operands*]**



AW $R_1, D_2(X_2, B_2)$ **[*RX, Long Operands*]**



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

In short-precision, the low-order halves of the floating-point registers are ignored and remain unchanged.

After the addition the intermediate sum is truncated to the proper fraction length.

When the resulting fraction is zero and the significance mask bit is one, a significance exception exists and a program interruption takes place. When the resulting fraction is zero and the significance mask bit is zero, the program interruption for the significance exception does not occur; rather, the characteristic is made zero, yielding a true zero result.

Leading zeros in the result are not eliminated by normalization, and an exponent underflow cannot occur.

The sign of the sum is derived by the rules of algebra. The sign of a sum with zero result fraction is always positive.

Resulting Condition Code:

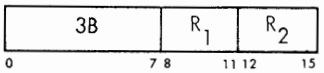
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

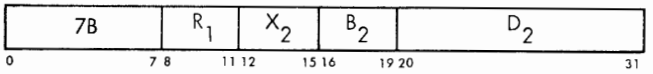
Operation (if floating-point feature is not installed)
Protection (fetch violation by AU and AW only)
Addressing (AU and AW only)
Specification
Significance
Exponent overflow

Subtract Normalized

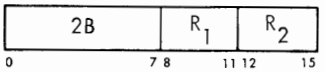
SER R_1, R_2 **[*RR, Short Operands*]**



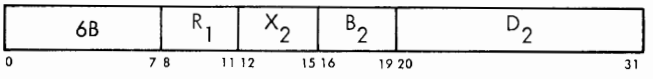
SE $R_1, D_2(X_2, B_2)$ **[*RX, Short Operands*]**



SDR R_1, R_2 **[*RR, Long Operands*]**



SD $R_1, D_2(X_2, B_2)$ ***RX, Long Operands*]**



The second operand is subtracted from the first operand, and the normalized difference is placed in the first operand location.

In short-precision, the low-order halves of the floating-point registers are ignored and remain unchanged.

The SUBTRACT NORMALIZED is similar to ADD NORMALIZED, except that the sign of the second operand is inverted before addition.

The sign of the difference is derived by the rules of algebra. The sign of a difference with zero result fraction is always positive.

Resulting Condition Code:

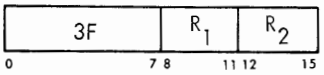
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

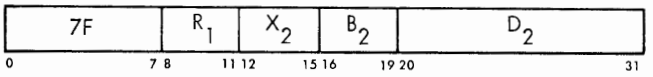
Operation (if floating-point feature is not installed)
Protection (fetch violation by SE and SD only)
Addressing (SD and SE only)
Specification
Significance
Exponent overflow
Exponent underflow

Subtract Unnormalized

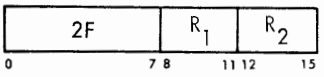
SUR R_1, R_2 **[*RR, Short Operands*]**



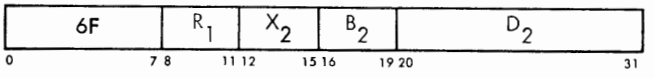
SU $R_1, D_2(X_2, B_2)$ **[*RX, Short Operands*]**



SWR R_1, R_2 **[*RR, Long Operands*]**



SW $R_1, D_2(X_2, B_2)$ **[*RX, Long Operands*]**



The second operand is subtracted from the first operand, and the unnormalized difference is placed in the first operand location.

In short-precision, the low-order halves of the floating-point register are ignored and remain unchanged.

The SUBTRACT UNNORMALIZED is similar to ADD UNNORMALIZED, except for the inversion of the sign of the second operand before addition.

The sign of the difference is derived by the rules of algebra. The sign of a difference with zero result fraction is always positive.

Resulting Condition Code:

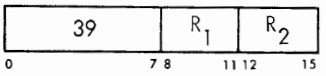
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 --

Program Interruptions:

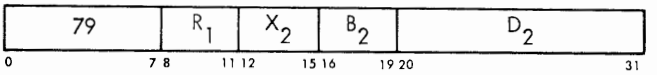
Operation (if floating-point feature is not installed)
Protection (fetch violation by SU and SW only)
Addressing (SW and SU only)
Specification
Significance
Exponent overflow

Compare

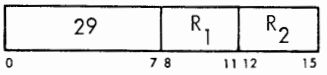
CER R_1, R_2 **[*RR, Short Operands*]**



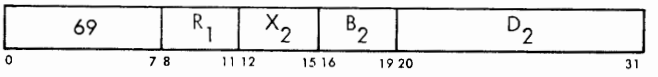
CE $R_1, D_2(X_2, B_2)$ **[*RX, Short Operands*]**



CDR R_1, R_2 **[*RR, Long Operands*]**



CD $R_1, D_2(X_2, B_2)$ **[*RX, Long Operands*]**



The first operand is compared with the second operand, and the condition code indicates the result.

In short-precision, the low-order halves of the floating-point registers are ignored.

Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. An exponent inequality is not decisive for magnitude determination since the fractions may have different numbers of leading zeros. An equality is established by following the rules for normalized floating-point subtraction. When the intermediate sum, including the guard digit,

is zero, the operands are equal. Neither operand is changed as a result of the operation.

An exponent-overflow, exponent-underflow, or lost-significance exception cannot occur.

Resulting Condition Code:

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

Program Interruptions:

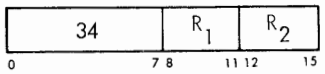
Operation (if floating-point feature is not installed)
Protection (fetch violation by CE and CD only)
Addressing (CD and CE only)
Specification

Programming Note

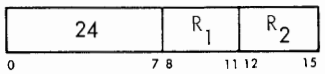
Numbers with zero fraction compare equal even when they differ in sign or characteristic.

Halve

HER R1, R2 **[RR, Short Operands]**



HDR R1, R2 **[RR, Long Operands]**



The second operand is divided by two, and the normalized quotient is placed in the first-operand location.

The second operand remains unchanged. In short-precision, the low-order halves of the floating-point registers remain unchanged.

The fraction of the second operand is shifted right one bit position, placing the contents of the low-order bit position into the high-order bit position of the guard digit and introducing a zero into the high-order bit position of the fraction. The intermediate result is subsequently normalized, and the normalized quotient is placed in the first-operand location. The guard digit participates in the normalization.

When normalization causes the characteristic to become less than zero, exponent underflow occurs. If the exponent-underflow mask is zero, the sign, characteristic, and fraction are set to zero, thus making the result a true zero. If the exponent-underflow mask is one, a program interruption occurs. The result is normalized, its sign and fraction remain correct, and the characteristic is made 128 larger than the correct characteristic.

When the fraction of the second operand is zero, the sign, characteristic, and fraction of the result are made zero. No normalization is attempted, and a significance exception is not recognized.

Condition Code: The code remains unchanged.

Program Interruptions:

Operation (if floating-point feature is not installed)
Specification
Exponent Underflow

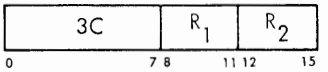
Programming Notes

In short- and long-precision, the halve operation is identical to a divide operation with the number two as divisor. In long-precision, the halve operation is identical to a multiply operation with one-half as a multiplier. In short-precision, HALVE differs from multiplication with one-half as the multiplier to the extent that halving preserves the contents of the low-order half of the register.

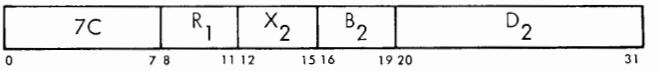
The result of HALVE is replaced by a true zero only when the second-operand fraction is zero, or when exponent underflow occurs with the exponent-underflow mask set to zero. When the fraction of the second operand is zero, except for the low-order bit position, the low-order one is shifted into the guard digit position and participates in the postnormalization.

Multiply

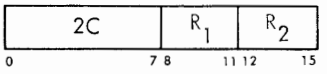
MER R1, R2 **[RR, Short Operands]**



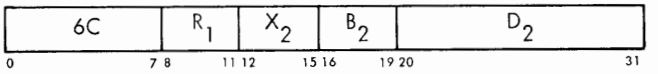
ME R1, D2(X2, B2) **[RX, Short Operands]**



MDR R1, R2 **[RR, Long Operands]**



MD R1, D2(X2, B2) **[RX, Long Operands]**



The normalized product of multiplier (the second operand) and multiplicand (the first operand) replaces the multiplicand.

The multiplication of two floating-point numbers consists of a characteristic addition and a fraction multiplication. The sum of the characteristics less 64 is used as the characteristic of an intermediate product.

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

The product fraction is normalized by prenormalizing the operands and postnormalizing the intermediate product, if necessary. The intermediate product characteristic is reduced by the number of left-shifts. For long operands, the intermediate product fraction is truncated to 15 digits before the left-shifting, if any. For short operands (six-digit fractions), the product fraction has the full 14 digits of the long format, and the two low-order fraction digits are accordingly always zero.

Exponent overflow occurs if the final product characteristic exceeds 127. The operation is completed, and a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is 128 smaller than the correct characteristic. The overflow exception does not occur for an intermediate product characteristic exceeding 127 when the final characteristic is brought within range because of normalization.

Exponent underflow occurs if the final product characteristic is less than zero. If the corresponding mask bit is one, a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is 128 larger than the correct characteristic. If the corresponding mask bit is not one, the result is made a true zero. Underflow is not signaled when an operand's characteristic becomes less than zero during prenormalization, and the correct characteristic and fraction value are used in the multiplication.

When all 15 digits of the intermediate product fraction are zero, the product sign and characteristic are made zero, yielding a true zero result. No interruption for exponent underflow or exponent overflow can occur when the result fraction is zero. The program interruption for lost significance is never taken for multiplication.

Condition Code: The code remains unchanged.

Program Interruptions:

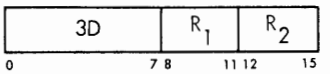
Operation (if floating-point feature is not installed)
Protection (fetch violation by ME and MD only)
Addressing (MD and ME only)
Specification
Exponent overflow
Exponent underflow

Programming Note

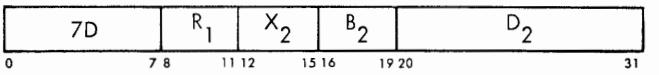
Interchanging the two operands in a floating-point multiplication does not affect the value of the product.

Divide

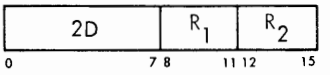
DER ***R_1, R_2*** **[*RR, Short Operands*]**



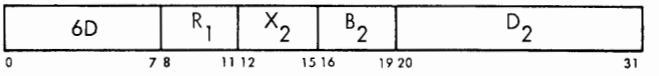
DE ***R_1, $D_2(X_2, B_2)$*** **[*RX, Short Operands*]**



DDR ***R_1, R_2*** **[*RR, Long Operands*]**



DD ***R_1, $D_2(X_2, B_2)$*** **[*RX, Long Operands*]**



The dividend (the first operand) is divided by the divisor (the second operand) and replaced by the quotient. No remainder is preserved.

In short-precision, the low-order halves of the floating-point register are ignored and remain unchanged.

A floating-point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics plus 64 is used as an intermediate quotient characteristic. The sign of the quotient is determined by the rules of algebra.

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right-shift may be called for. The intermediate-quotient characteristic is adjusted for the shifts. All dividend fraction digits participate in forming the quotient, even if the normalized dividend fraction is larger than the normalized divisor fraction. The quotient fraction is truncated to the desired number of digits.

A program interruption for exponent overflow occurs when the final-quotient characteristic exceeds 127. The operation is completed. The fraction is correct and normalized, the sign is correct, and the characteristic is 128 smaller than the correct characteristic.

If the final quotient characteristic is less than zero and the mask bit is one, a program interruption for exponent underflow occurs. The fraction is correct and normalized, the sign is correct, and the characteristic is 128 larger than the correct characteristic. If the corresponding mask bit is not one, the result is made a true zero. Underflow is not signaled for the inter-

mediate quotient or for the operand characteristics during prenormalization.

When division by a divisor with zero fraction is attempted, the operation is suppressed. The dividend remains unchanged, and a program interruption for floating-point divide occurs. When the dividend fraction is zero, the quotient fraction will be zero. The quotient sign and characteristic are made zero, yielding a true zero result without taking the program interruption for exponent underflow and exponent overflow. The program interruption for significance is never taken for division.

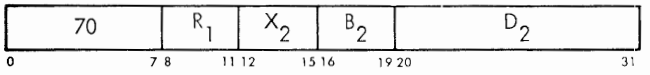
Condition Code: The code remains unchanged.

Program Interruptions:

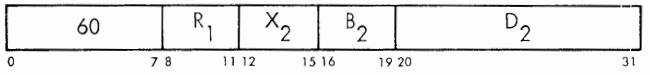
Operation (if floating-point feature is not installed)
Protection (fetch violation by DE and DD only)
Addressing (DD and DE only)
Specification
Exponent overflow
Exponent underflow
Floating-point divide

Store

STE ***R_1, $D_2(X_2, B_2)$*** ***[RX, Short Operands]***



STD ***R_1, $D_2(X_2, B_2)$*** ***[RX, Long Operands]***



The first operand is stored at the second operand location.

In short-precision, the low-order half of the first operand register is ignored. The first operand remains unchanged.

Condition Code: The code remains unchanged.

Program Interruptions:

Operation (if floating-point feature is not installed)
Addressing
Protection (store violation)
Specification

Floating-Point Arithmetic Exceptions

Exceptional operation codes, operand designations, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in floating-point arithmetic.

Operation: The floating-point feature is not installed, and an attempt is made to execute a floating-point instruction. The instruction is suppressed. The condition code and data in registers and storage remain unchanged.

Protection: The key of an operand in storage does not match the protection key in the PSW. The operation is suppressed on a store violation. Therefore, the condition code and data in registers remain unchanged. On a fetch violation, the operation is terminated; result data and the condition code are unpredictable.

Addressing: An address designates an operand location outside the available storage for the installed system. In most cases, the operation is terminated. The result data and the condition code, if affected, are unpredictable and should not be used for further computation. The exception is STORE (STE and STD), which is suppressed.

Specification: A short operand is not located on a 32-bit boundary or a long operand is not located on a 64-bit boundary; or, a floating-point register address other than 0, 2, 4, or 6 is specified. The instruction is suppressed. Therefore, the condition code and data in registers and storage remain unchanged. The address restrictions do not apply to the components from which an address is generated – the content of the D_2 field and the contents of the registers specified by X_2 and B_2.

Exponent Overflow: The result characteristic in addition, subtraction, multiplication, or division exceeds 127, and the result fraction is not zero. The operation is completed, and a program interruption occurs. The fraction is normalized, and the sign and fraction of the result remain correct. The result characteristic is made 128 smaller than the correct characteristic. For addition and subtraction, the condition code is set to 1 when the result is less than zero, and the condition code is set to 2 when the result is greater than zero. For multiplication and division, the condition code remains unchanged.

Exponent Underflow: The result characteristic in addition, subtraction, multiplication, halving, or division is less than zero, and the result fraction is not zero. The operation is completed, and a program interruption occurs if the exponent-underflow mask bit (PSW bit 38) is one.

The setting of the exponent-underflow mask also affects the result of the operation. When the mask bit is zero, the sign, characteristic, and fraction are set to zero, thus making the result a true zero. When the mask bit is one, the fraction is normalized, the char-

acteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct.

For addition and subtraction, the condition code is set to 0 when the exponent-underflow mask bit is zero. With the mask bit one, the condition code for addition and subtraction is set to 1 when the result is less than zero, and the condition code is set to 2 when the result is greater than zero. For multiplication, halving, and division, the condition code is left unchanged.

Significance: The result fraction of an addition or subtraction is zero. A program interruption occurs if the significance mask bit (PSW bit 39) is one. The mask bit affects also the result of the operation. When the significance mask bit is a zero, the operation is completed by replacing the result with a true zero. When the significance mask bit is one, the operation is completed without further change to the characteristic of the result. In either case, the condition code is set to 0.

Floating-Point Divide: Division by a number with zero fraction is attempted. The division is suppressed; therefore, the condition code and data in registers and storage remain unchanged.

Extended Precision and Rounding

Data Format

An extended-precision floating-point number has a 28-digit fraction and consists of two long-precision floating-point numbers in consecutive floating-point registers. Two pairs of adjacent floating-point registers can be used as sources of extended-precision operands or destinations of extended-precision results: registers 0, 2 and registers 4, 6. The designation of any other register pair causes a program interruption for a specification exception.

The two long-precision numbers comprised in an extended-precision number are called the high-order and the low-order parts. The high-order part may be any long-precision floating-point number. If it is normalized, the extended-precision number is considered normalized. The characteristic of the high-order part is called the characteristic of the extended-precision number, and the sign of the high-order part is the sign of the extended-precision number.

The fraction field of the low-order part contains the 14 low-order hexadecimal digits of the 28-digit extended-precision fraction. The sign and characteristic of the low-order part of an extended-precision operand are ignored, the value of the number being assumed such as if the sign of the low-order part were the same as that of the high-order part, and the characteristic of the low-order part were 14 less than that of the high-order part. In extended-precision results, the sign of the low-order part is made the same as that of the high-order part, and the low-order characteristic is made 14 less than the high-order characteristic. When the subtraction of 14 causes the low-order characteristic to become less than zero, it is made 128 larger than its correct value. Exponent-underflow is indicated only when the high-order characteristic underflows.

Programming Note

A long-precision number can be extended to the extended-precision format by appending any long-precision number having a zero fraction, including a true zero. Conversion from extended to long precision can be accomplished by truncation or by means of the long-precision LOAD ROUNDED.

In the absence of an exponent overflow or exponent underflow, the long-precision number constituting the low-order part of an extended-precision result correctly expresses the value of the low-order part of the extended-precision result when the characteristic of the high-order part is 14 or higher. When the high-order characteristic is less than 14, the low-order part, when addressed as a long-precision number, does not have the correct value.

The low-order part of an extended-precision result is not necessarily normalized.

Instructions

Five arithmetic instructions are provided that utilize extended-precision operand or result formats, and two instructions are provided for rounding from extended to long and from long to short formats. A list of these instructions and their mnemonics, formats, and operation codes follows. The table indicates also when the condition code is set and the exceptions in operand designation, data, or results that cause a program interruption.

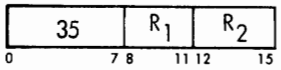
NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Load Rounded (Extended to Long)	LRDR	RR X	S, E	25
Load Rounded (Long to Short)	LRER	RR X	S, E	35
Add Normalized (Extended)	AXR	RR X,C	S,U,E,LS	36
Subtract Normalized (Extended)	SXR	RR X,C	S,U,E,LS	37
Multiply (Extended)	MXR	RR X	S,U,E	26
Multiply (Long/Extended)	MXDR	RR X	S,U,E	27
Multiply (Long/Extended)	MXD	RX X	P,A,S,U,E	67

NOTES

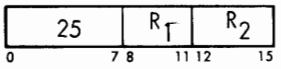
A	Addressing exception
C	Condition code is set
E	Exponent-overflow exception
LS	Significance exception
P	Protection exception
S	Specification exception
U	Exponent-underflow exception
X	Extended-precision feature

Load Rounded

LRER R_1, R_2 **[RR, Long Operand, Short Result]**



LRDR R_1, R_2 **[RR, Extended Operand, Long Result]**



The second operand is rounded to the next smaller format, and the result is placed in the first-operand location.

The second operand remains unchanged unless it appears in the first-operand location. For short results, the low-order half of the result register remains unchanged.

Rounding is predicated on the first fraction bit of the next longer floating-point format that includes the format of the rounded result. When the rounded result is in short precision, rounding consisits of adding one in absolute sense to the contents of bit position 32 of the designated floating-point register and propagating the carry, if any, to the left. For long results, bit 8 of the next-higher addressed register is inspected. If this bit is one, one is added in absolute sense into bit position 63 of the designated register; if this bit is zero, no change is made.

If rounding causes a carry out of the high-order digit position of the fraction, the fraction is shifted right one digit position and the characteristic is increased by one. No normalization takes place.

An exponent-overflow exception is recognized when shifting the fraction right causes the characteristic to exceed 127. The operation is completed by loading a number whose characteristic is 128 smaller than the correct value, and a program interruption for exponent overflow subsequently occurs. The result fraction and sign remain unchanged.

Exponent underflow and significance exceptions cannot occur.

Condition Code: The code remains unchanged.

Program Interruptions:

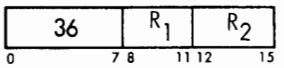
Operation: The instruction is not installed. The operation is suppressed.

Specification: The R_1 field designates a register other than 0, 2, 4, or 6, the R_2 field of LRER designates a register other than 0, 2, 4, or 6, or the R_2 field of LRDR designates a register other than 0 or 4. The operation is suppressed.

Exponent Overflow: The characteristic of the rounded result exceeds 127. The operation is completed.

Add Normalized

AXR R_1, R_2 **[RR, Extended Operands]**



The normalized extended sum of the extended second operand and the extended first operand is placed in the first-operand location.

Addition of two floating-point numbers consists of characteristic comparison and fraction addition. The characteristics of the two operands are compared, and the fraction accompanying the smaller characteristic is shifted right with its characteristic increased by one for each hexadecimal digit of shift until the two characteristics agree.

When an operand is shifted right during alignment, the leftmost hexadecimal digit of the field shifted out is retained as a guard digit, thus forming an intermediate operand consisting of 29 digits. The operand that is not shifted is considered to be extended with a low-order zero. Both operands are considered to be extended with low-order zeros when no alignment shift occurs. The 29-digit fractions are then added algebraically to form an intermediate sum.

The intermediate-sum fraction consists of 29 hexadecimal digits and a possible carry. If a carry is present, the sum is shifted right one digit position and the characteristic is increased by one. If high-order zeros are present, the 29-digit fraction is shifted left to form a normalized number, provided the fraction is not zero. Vacated low-order digit positions are filled with zeros, and the characteristic is reduced by the number of hexadecimal digits of shift. The intermediate-sum fraction is subsequently truncated to 28 hexadecimal digits.

The sign of the sum is derived by the rules of algebra. When all digits of the intermediate-sum fraction are zero, the sign is made plus.

Unless the result is made a true zero, the characteristic, sign, and high-order 14 hexadecimal digits of the normalized and truncated sum fraction, the high-order sum, replace the high-order part of the first operand. The low-order 14 hexadecimal digits of the sum fraction replace the low-order fraction of the first operand. The low-order sign is made equal to the high-order sign. The low-order characteristic is made 14 less than the high-order characteristic unless subtraction of 14 causes it to become less than zero, in which case it is made 128 greater than its correct value.

An exponent-overflow exception is recognized when

a carry from the high-order position of the intermediate-sum fraction causes the characteristic of the normalized sum to exceed 127. The operation is completed by making the high-order characteristic 128 less than the correct value, and a program interruption for exponent overflow occurs. The sum fraction and sign remain unchanged.

An exponent-underflow exception exists when the characteristic of the normalized sum is less than zero and the intermediate-sum fraction, including the guard digit, is not zero. If the exponent-underflow mask bit is one, the operation is completed by making both characteristics 128 greater than their correct values. The sum fraction and sign remain unchanged. A program interruption for exponent underflow then takes place. When exponent underflow occurs and the exponent-underflow mask bit is zero, program interruption does not take place; instead, the operation is completed by making both the high-order and the low-order parts of the sum a true zero. Exponent underflow is not recognized when the low-order characteristic is less than zero, but the high-order characteristic is zero or above.

A significance exception exists when the intermediate-sum fraction, including the guard digit, is zero. If the significance mask bit is one, the intermediate-sum characteristic remains unchanged and becomes the characteristic of the result. No normalization occurs, and a program interruption for significance takes place. If the significance mask bit is zero, the program interruption does not occur; rather, both the high-order and the low-order parts of the sum are made a true zero.

Resulting Condition Code:

0 Sum fractions are zero
1 High-order sum is less than zero
2 High-order sum is greater than zero
3 --

Program Interruptions:

Operation: The instruction is not installed. The operation is suppressed.

Specification: The R_1 or R_2 field designates a register other than 0 or 4. The operation is suppressed.

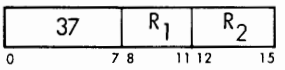
Exponent Overflow: The characteristic of the normalized sum exceeds 127. The operation is completed.

Exponent Underflow: The characteristic of the normalized sum is less than zero, the sum fraction is not zero, and the exponent underflow mask bit is one. The operation is completed.

Significance: The intermediate-sum fraction is zero, and the significance mask bit is one. The operation is completed.

Subtract Normalized

SXR R₁, R₂ **[RR, Extended Operands]**



The extended second operand is subtracted from the extended first operand, and the normalized extended difference is placed in the first-operand location.

The execution of SUBTRACT NORMALIZED is identical to that of ADD NORMALIZED, except that the sign of the second operand is inverted before the addition.

Resulting Condition Code:

0 Difference fractions are zero
1 High-order difference is less than zero
2 High-order difference is greater than zero
3 --

Program Interruptions:

Operation: The instruction is not installed. The operation is suppressed.

Specification: The R_1 or R_2 field designates a register other than 0 or 4. The operation is suppressed.

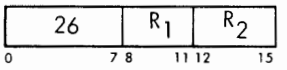
Exponent Overflow: The characteristic of the normalized difference exceeds 127. The operation is completed.

Exponent Underflow: The characteristic of the normalized difference is less than zero, the difference fraction is not zero, and the exponent underflow mask bit is one. The operation is completed.

Significance: The intermediate-sum fraction is zero, and the significance mask bit is one. The operation is completed.

Multiply

MXR R₁, R₂ **[RR, Extended Operands]**



The normalized extended product of the extended second operand (the multiplier) and the extended first operand (the multiplicand) is placed in the first-operand location.

Multiplication of two floating-point numbers consists of exponent addition and fraction multiplication. The operands are normalized, and the sum of the characteristics of the normalized operands, less 64, is used as the characteristic of the intermediate product.

The product of the fractions is developed such that the result has the exact fraction product truncated to

28 hexadecimal digits. When the result is normalized without requiring any postshifting, the intermediate-product fraction is truncated to 28 digits, and the intermediate-product characteristic becomes the final product characteristic. When the intermediate-product fraction has one leading zero digit, it is shifted left one digit position, bringing the contents of the guard digit position into the low-order position of the result fraction, and the intermediate-product characteristic is reduced by one. The intermediate-product fraction is subsequently truncated to 28 digits.

The sign of the product is determined by the rules of algebra. When all digits of the product fraction are zero, the sign is made plus.

Unless the result is made a true zero, the characteristic, sign, and high-order 14 hexadecimal digits of the normalized and truncated product fraction (the high-order product) replace the high-order part of the first operand. The low-order 14 hexadecimal digits of the product fraction replace the low-order fraction of the first operand. The low-order sign is made equal to the high-order sign. The low-order characteristic is made 14 less than the high-order characteristic unless subtraction of 14 causes it to become less than zero, in which case it is made 128 greater than its correct value.

An exponent-overflow exception is recognized when the characteristic of the normalized product exceeds 127 and the fraction of the product is not zero. The operation is completed by making the high-order characteristic 128 less than the correct value. If the low-order characteristic also exceeds 127, it, too, is decreased by 128. The product fraction and sign remain unchanged. A program interruption for exponent overflow then occurs. Exponent overflow is not recognized if the intermediate-product characteristic exceeds 127 but is brought within range by normalization.

An exponent-underflow exception exists when the characteristic of the normalized product is less than zero and the fraction of the product is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristics of both parts 128 greater than their correct values, and a program interruption for exponent underflow occurs. The product fraction and its sign remain unchanged. If the exponent-underflow mask bit is zero, program interruption does not take place; instead the operation is completed by making both the high-order and low-order parts of the product a true zero. Exponent underflow is not recognized when the low-order characteristic is less than zero, but the high-order characteristic is zero or above.

If either or both operand fractions are zero, both parts of the result are made a true zero, and no exceptions are recognized.

Condition Code: The code remains unchanged.

Program Interruptions:

Operation: The instruction is not installed. The operation is suppressed.

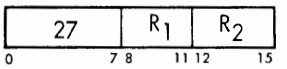
Specification: The R_1 or R_2 field designates a register other than 0 or 4. The operation is suppressed.

Exponent Overflow: The characteristic of the normalized product exceeds 127, and the product fraction is not zero. The operation is completed.

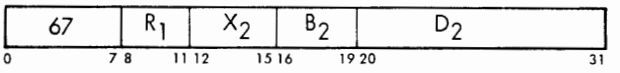
Exponent Underflow: The characteristic of the normalized product is less than zero, the product fraction is not zero, and the exponent underflow mask bit is one. The operation is completed.

Multiply

MXDR R_1, R_2 ***[RR, Long Operands, Extended Result]***



MXD R_1, D_2 (X_2, B_2) ***[RX, Long Operands, Extended Result]***



The normalized extended product of the long second operand (the multiplier) and the long first operand (the multiplicand) is placed in the first-operand location and the next-higher-addressed register.

Multiplication of two floating-point numbers consists of exponent addition and fraction multiplication. The sum of the characteristics, less 64, is used as the characteristic of the intermediate product. The intermediate-product fraction consists of 28 hexadecimal digits and is an exact product of the operand fractions. This fraction is shifted left as necessary to form a normalized result, and the characteristic is reduced by the number of hexadecimal digits of shift. Zeros are provided for the vacated low-order digit positions.

The sign of the product is determined by the rules of algebra. When all digits of the product fraction are zero, the sign is made plus.

Unless the result is made a true zero, the characteristic, sign, and high-order 14 hexadecimal digits of the normalized product fraction (the high-order product) replace the first operand. The low-order 14 hexa-

decimal digits of the product fraction are placed in bit positions 8-63 of the next-higher addressed register. Bit 0 of this register, the low-order sign, is made equal to the high-order sign. The contents of bit positions 1-7 of this register, the low-order characteristic, are made 14 less than the high-order characteristic, unless subtraction of 14 causes it to become less than zero, in which case it is made 128 greater than its correct value. The original contents of the register receiving the low-order product are ignored.

An exponent-overflow exception is recognized when the characteristic of the normalized product exceeds 127 and the fraction of the product is not zero. The operation is completed by making the high-order characteristic 128 less than the correct value. If the low-order characteristic also exceeds 127, it, too, is decreased by 128. The product fraction and sign remain unchanged. A program interruption for exponent overflow then occurs. Exponent overflow is not recognized if the intermediate-product characteristic exceeds 127 but is brought within range by normalization.

An exponent-underflow exception exists when the characteristic of the normalized product is less than zero and the fraction of the product is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristics of both parts 128 greater than their correct values, and a program interruption for exponent underflow occurs. The product fraction and its sign remain unchanged. If the exponent-underflow mask bit is zero, program interruption does not take place; instead the operation is completed by making both the high-order and low-order parts of the product a true zero. Exponent underflow is not recognized when the low-order characteristic is less than zero, but the high-order characteristic is zero or above.

If either or both operand fractions are zero, both parts of the result are made a true zero, and no exceptions are recognized.

Condition Code: The code remains unchanged.

Program Interruptions:

Operation: The instruction is not installed. The operation is suppressed.

Protection: The location of the second operand of MXD is protected for fetching, and the key in storage associated with the operand does not match the protection key in the PSW. The operation is terminated.

Addressing: The location of the second operand of MXD is outside the available main storage of the installation. The operation is terminated.

Specification: The R_1 field designates a register other than 0 or 4, the R_2 field of MXDR designates a register other than 0,2,4, or 6, or, in the MXD format, the second operand is not located on a 64-bit boundary of main storage. The operation is suppressed.

Exponent Overflow: The characteristic of the normalized product exceeds 127, and the product fraction is not zero. The operation is completed.

Exponent Underflow: The characteristic of the normalized product is less than zero, the product fraction is not zero, and the exponent underflow mask bit is one. The operation is completed.

A set of instructions is provided for the logical manipulation of data. Generally, the operands are treated as eight-bit bytes. In a few cases the left or right four bits of a byte are treated separately or operands are shifted a bit at a time. The operands are either in storage or in general registers. Some operands are introduced from the instruction stream.

Processing of data in storage proceeds left to right through fields which may start at any byte position. In the general registers, the processing, as a rule, involves the entire register contents.

Except for the editing instructions, data are not treated as numbers. Editing provides a transformation from packed decimal digits to alphanumeric characters.

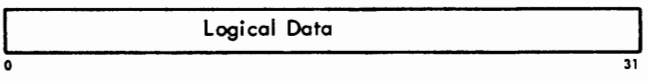
The set of logical operations includes moving, comparing, bit connecting, bit testing, translating, editing, and shift operations. All logical operations other than editing are part of the standard instruction set. Editing instructions are part of the decimal feature.

The condition code is set as a result of all logical comparing, connecting, testing, and editing operations.

Data Format

Data reside in general registers or in storage or are introduced from the instruction stream. The data size may be a single or double word, a single character, or variable in length. When two operands participate they have equal length, except in the editing instructions.

Fixed-Length Logical Information

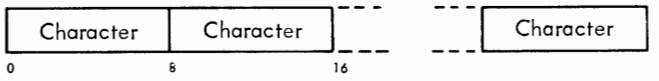


Data in general registers normally occupy all 32 bits. Bits are treated uniformly, and no distinction is made between sign and numeric bits. In a few operations, only the low-order eight bits of a register participate, leaving the remaining 24 bits unchanged. In some shift operations, 64 bits of an even/odd pair of registers participate.

The LOAD ADDRESS introduces a 24-bit address into a general register. The high-order eight bits of the register are made zero.

In storage-to-register operations, the storage data occupy either a word of 32 bits or a byte of eight bits. The word must be located on word boundaries, that is, its address must have the two low-order bits zero.

Variable-Length Logical Information



In storage-to-storage operations, data have a variable field-length format, starting at any byte address and continuing for up to a total of 256 bytes. Processing is left to right.

Operations introducing data from the instruction stream into storage, as immediate data, are restricted to an eight-bit byte. Only one byte is introduced from the instruction stream, and only one byte in storage participates.

Use of general register 1 is implied in TRANSLATE AND TEST and EDIT AND MARK. A 24-bit address may be placed in this register during these operations. The TRANSLATE AND TEST also implies general register 2. The low-order eight bits of register 2 may be replaced by a function byte during a translate-and-test operation.

Editing requires a packed decimal field and generates zoned decimal digits. The digits, signs, and zones are recognized and generated as for decimal arithmetic. Otherwise, no internal data structure is required, and all bit configurations are considered valid.

The translating operations use a list of arbitrary values. A list provides a relation between an argument (the quantity used to reference the list) and the function (the content of the location related to the argument). The purpose of the translation may be to convert data from one code to another code or to perform a control function.

A list is specified by an initial address – the address designating the leftmost byte location of the list. The byte from the operand to be translated is the argument. The actual address used to address the list is obtained by adding the argument to the low-order po-

sitions of the initial address. As a consequence, the list contains 256 eight-bit function bytes. In cases where it is known that not all eight-bit argument values will occur, it may be possible to reduce the size of the list.

In a storage-to-storage operation, the operand fields may be defined in such a way that they overlap. The effect of this overlap depends upon the operation. When the operands remain unchanged, as in COMPARE or TRANSLATE AND TEST, overlapping does not affect the execution of the operation. In the case of MOVE, EDIT, and TRANSLATE, one operand is replaced by new data, and the execution of the operation may be affected by the amount of overlap and the manner in which data are fetched or stored. For purposes of evaluating the effect of overlapped operands, consider that data are handled one eight-bit byte at a time. All overlapping fields are considered valid but, in editing, overlapping fields give unpredictable results.

Condition Code

The results of most logical operations are used to set the condition code in the PSW. The LOAD ADDRESS, INSERT CHARACTERS, STORE CHARACTER, TRANSLATE, and the moving and shift operations leave this code unchanged. The condition code can be used for decision-making by subsequent branch-on-condition instructions.

The condition code can be set to reflect five types of results for logical operations: For COMPARE LOGICAL the states 0, 1, or 2 indicate that the first operand is equal, low, or high.

For the logical-connectives, the states 0 or 1 indicate a zero or nonzero result field.

For TEST UNDER MASK, the states 0, 1, or 3 indicate that the selected bits are all-zero, mixed zero and one, or all-one.

For TRANSLATE AND TEST, the states 0, 1, or 2 indicate an all-zero function byte, a nonzero function byte with the operand incompletely tested, or a last function byte nonzero.

For editing the states 0, 1, or 2 indicate a zero, less than zero, or greater than zero content of the last result field.

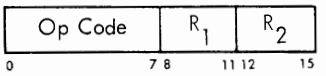
CONDITION CODE SETTING FOR LOGICAL OPERATIONS

	0	1	2	3
And	zero	not zero	--	--
Compare Logical	equal	low	high	--
Edit	zero	< zero	> zero	--
Edit and Mark	zero	< zero	> zero	--
Exclusive Or	zero	not zero	--	--
Or	zero	not zero	--	--
Test Under Mask	zero	mixed	--	one
Translate and Test	zero	incomplete	complete	--

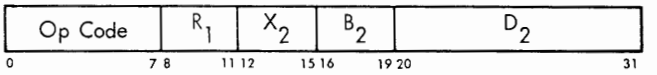
Instruction Format

Logical instructions use the following five formats:

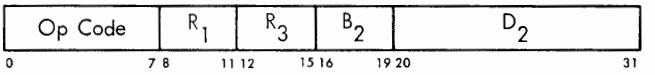
RR Format



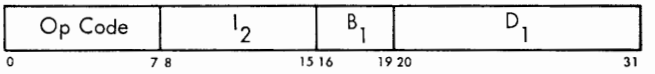
RX Format



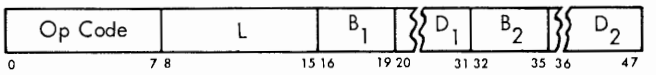
RS Format



SI Format



SS Format



In the RR, RX, and RS formats, the content of the register specified by R_1 is called the first operand.

In the SI and SS formats, the content of the general register specified by B_1 is added to the content of the D_1 field to form an address. This address designates the leftmost byte of the first operand field. The number of bytes to the right of this first byte is specified by the L field in the SS format. In the SI format the operand size is one byte.

In the RR format, the R_2 field specifies the register containing the second operand. The same register may be specified for the first and 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 content of the D_2 field to form the address of the second operand.

In the RS format, used for shift operations, the content of the general register specified by the B_2 field is added to the content of the D_2 field. This sum is not used as an address but specifies the number of bits of the shift. The R_3 field is ignored in the shift operations.

In the SI format, the second operand is the eight-bit immediate data field, I_2, of the instruction.

In the ss format, the content of the general register specified by B_2 is added to the content of the D_2 field to form the address of the second operand. The second operand field has the same length as the first operand field.

A zero in any of the X_2, B_1, or B_2 fields indicates the absence of the corresponding address or shift-amount component. An instruction can specify the same general register both for address modification and for operand location. Address modification is always completed prior to operation execution.

Results replace the first operand, except in STORE CHARACTER, where the result replaces the second operand. A variable-length result is never stored outside the field specified by the address and length.

The contents of all general registers and storage locations participating in the addressing or execution of an operation generally remain unchanged. Exceptions are the result locations, general register 1 in EDIT AND MARK, and general registers 1 and 2 in TRANSLATE AND TEST.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction: For MOVE NUMERICS, for example, MVN is the mnemonic and D_1 (L, B_1), D_2 (B_2) the operand designation.

Instructions

The logical instructions, their mnemonics, formats, and operation codes follow. The table also indicates when the condition code is set and the exceptions in operand designations, data, or results that cause a program interruption.

NAME	MNEMONIC	TYPE		EXCEPTIONS	CODE
Move	MVI	SI		P,A	92
Move	MVC	SS		P,A	D2
Move Numerics	MVN	SS		P,A	D1
Move Zones	MVZ	SS		P,A	D3
Compare Logical	CLR	RR	C		15
Compare Logical	CL	RX	C	P,A,S	55
Compare Logical	CLI	SI	C	P,A	95
Compare Logical	CLC	SS	C	P,A	D5
AND	NR	RR	C		14
AND	N	RX	C	P,A,S	54
AND	NI	SI	C	P,A	94
AND	NC	SS	C	P,A	D4
OR	OR	RR	C		16
OR	O	RX	C	P,A,S	56
OR	OI	SI	C	P,A	96
OR	OC	SS	C	P,A	D6
Exclusive OR	XR	RR	C		17
Exclusive OR	X	RX	C	P,A,S	57
Exclusive OR	XI	SI	C	P,A	97
Exclusive OR	XC	SS	C	P,A	D7
Test Under Mask	TM	SI	C	P,A	91
Insert Character	IC	RX		P,A	43
Store Character	STC	RX		P,A	42
Load Address	LA	RX			41
Translate	TR	SS		P,A	DC
Translate and Test	TRT	SS	C	P,A	DD
Edit	ED	SS	T,C	P,A, D	DE
Edit and Mark	EDMK	SS	T,C	P,A, D	DF
Shift Left Single Logical	SLL	RS			89
Shift Right Single Logical	SRL	RS			88
Shift Left Double Logical	SLDL	RS		S	8D
Shift Right Double Logical	SRDL	RS		S	8C

NOTES

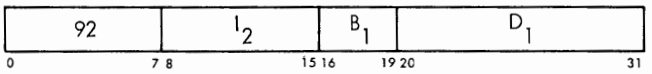
A	Addressing exception
C	Condition code is set
D	Data exception
P	Protection exception
S	Specification exception
T	Decimal feature

Programming Note

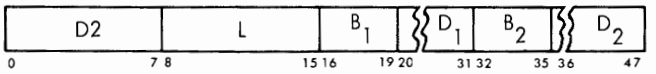
The fixed-point loading and storing instructions also may be used for logical operations.

Move

MVI $D_1(B_1), I_2$ **[*SI*]**



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



The second operand is placed in the first operand location.

The ss format is used for a storage-to-storage move. The SI format introduces one 8-bit byte from the instruction stream.

In storage-to-storage movement the fields may overlap in any desired way. Movement is left to right through each field a byte at a time.

The bytes to be moved are not changed or inspected.

Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store violation for MVI; store or fetch violation for MVC)

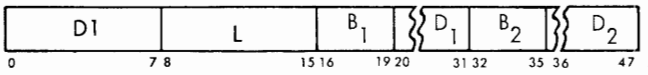
Addressing

Programming Note

It is possible to propagate one character through an entire field by having the first operand field start one character to the right of the second operand field.

Move Numerics

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



The low-order four bits of each byte in the second operand field, the numerics, are placed in the low-order bit positions of the corresponding bytes in the first operand fields.

The instruction is storage to storage. Movement is left to right through each field one byte at a time, and the fields may overlap in any desired way.

The numerics are not changed or checked for validity. The high-order four bits of each byte, the zones, remain unchanged in both operand fields.

Condition Code: The code remains unchanged.

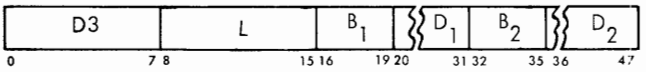
Program Interruptions:

Protection (store or fetch violation)

Addressing

Move Zones

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



The high-order four bits of each byte in the second operand field, the zones, are placed in the high-order four bit positions of the corresponding bytes in the first operand field.

The instruction is storage to storage. Movement is left to right through each field one byte at a time, and the fields may overlap in any desired way.

The zones are not changed or checked for validity. The low-order four bits of each byte, the numerics, remain unchanged in both operand fields.

Condition Code: The code remains unchanged.

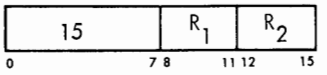
Program Interruptions:

Protection (store or fetch violation)

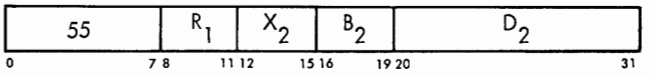
Addressing

Compare Logical

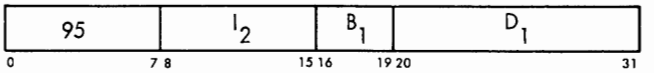
CLR R_1, R_2 [RR]



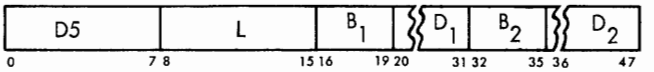
CL $R_1, D_2(X_2, B_2)$ [RX]



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



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



The first operand is compared with the second operand, and the result is indicated in the condition code.

The instructions allow comparisons that are register to register, storage to register, instruction to storage, and storage to storage.

Comparison is binary, and all codes are valid. The operation proceeds left to right and ends as soon as an inequality is found or the end of the fields is reached. However, when part of an operand in CLC is specified in an unavailable location, the operation may be terminated by the addressing exception, even though an inequality could have been found in a comparison of the available operand parts.

Resulting Condition Code:

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

Program Interruptions:

Protection (fetch violation for CL, CLI, and CLC only)

Addressing (CL, CLI, CLC only)

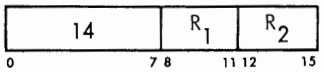
Specification (CL only)

Programming Note

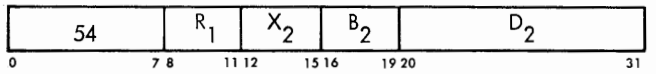
The COMPARE LOGICAL is unique in treating all bits alike as part of an unsigned binary quantity. In variable-length operation, comparison is left to right and may extend to field lengths of 256 bytes. The operation may be used to compare unsigned packed decimal fields or alphameric information in any code that has a collating sequence based on ascending or descending binary values. For example, EBCDIC has a collating sequence based on ascending binary values.

AND

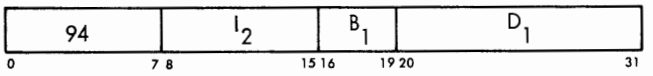
NR R_1, R_2 [RR]



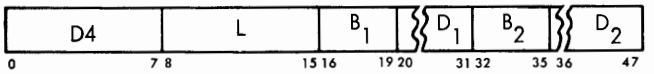
N $R_1, D_2(X_2, B_2)$ [RX]



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



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



The logical product (AND) of the bits of the first and second operand is placed in the first operand location.

Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set to one if the corresponding bit positions in both operands contain a one; otherwise, the result bit is set to zero. All operands and results are valid.

Resulting Condition Code:

0 Result is zero
1 Result not zero
2 --
3 --

Program Interruptions:

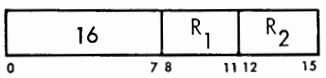
Protection (fetch violation only for N; store violation only for NI; store or fetch violation for NC)
Addressing (N, NI, NC only)
Specification (N only)

Programming Note

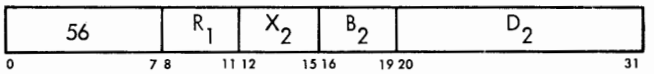
The AND may be used to set a bit to zero.

OR

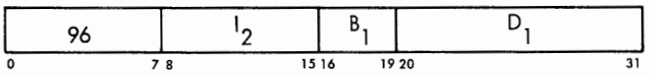
OR R_1, R_2 [RR]



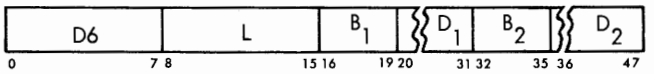
O $R_1, D_2(X_2, B_2)$ [RX]



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



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



The logical sum (OR) of the bits of the first and second operand is placed in the first operand location.

Operands are treated as unstructured logical quantities, and the connective *inclusive* OR is applied bit by bit. A bit position in the result is set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero. All operands and results are valid.

Resulting Condition Code:

0 Result is zero
1 Result not zero
2 --
3 --

Program Interruptions:

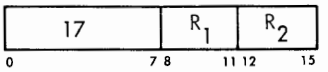
Protection (fetch violation only for O; store violation only for OI; store or fetch violation for OC)
Addressing (O, OI, OC only)
Specification (O only)

Programming Note

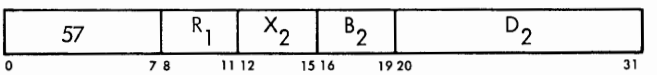
The OR may be used to set a bit to one.

Exclusive OR

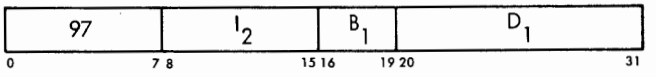
XR R_1, R_2 [RR]



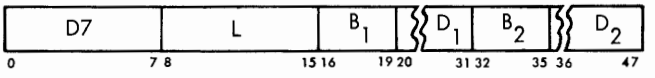
X $R_1, D_2(X_2, B_2)$ [RX]



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



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



The modulo-two sum (*exclusive* OR) of the bits of the first and second operand is placed in the first operand location.

Operands are treated as unstructured logical quantities, and the connective exclusive OR is applied bit by bit. A bit position in the result is set to one if the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

The instruction differs from AND and OR only in the connective applied.

Resulting Condition Code:

0 Result is zero
1 Result not zero
2 --
3 --

Program Interruptions:

Protection (fetch violation only for X; store violation only for XI; store or fetch violation for XC)
Addressing (X, XI, XC only)
Specification (X only)

Programming Notes

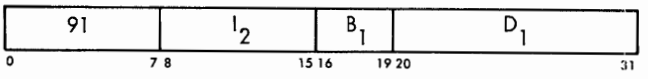
The exclusive OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.

Any field exclusive OR'ed with itself becomes all zeros.

The sequence A exclusive OR'ed B, B exclusive OR'ed A, A exclusive OR'ed B results in the exchange of the contents of A and B without the use of an auxiliary buffer area.

Test Under Mask

TM $D_1(B_1), I_2$ [*SI*]



The state of the first operand bits selected by a mask is used to set the condition code.

The byte of immediate data, I_2, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the character in storage specified by the first operand address.

A mask bit of one indicates that the storage bit is to be tested. When the mask bit is zero, the storage bit is ignored. When all storage bits thus selected are zero, the condition code is made 0. The code is also made 0 when the mask is all-zero. When the selected bits are all-one, the code is made 3; otherwise, the code is made 1. The character in storage is not changed.

Resulting Condition Code:

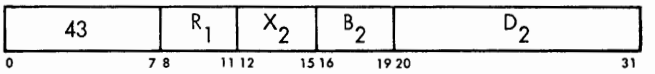
0 Selected bits all-zero; mask is all-zero
1 Selected bits mixed zero and one
2 --
3 Selected bits all-one

Program Interruptions:

Protection (fetch violation)
Addressing

Insert Character

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



The eight-bit character at the second operand address is inserted into bit positions 24-31 of the register specified as the first operand location. The remaining bits of the register remain unchanged.

The instruction is storage to general register. The byte to be inserted is not changed or inspected.

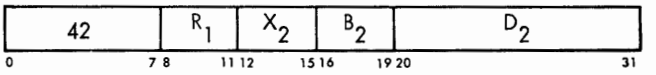
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (fetch violation)
Addressing

Store Character

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



Bit positions 24-31 of the register designated as the first operand are placed at the second operand address.

The instruction is general register to storage. The byte to be stored is not changed or inspected.

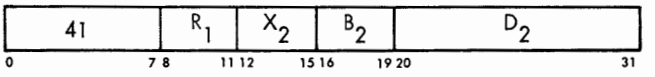
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store violation)
Addressing

Load Address

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



The address of the second operand is inserted in the low-order 24 bits of the general register specified by R_1. The remaining bits of the general register are made zero. No storage references for operands take place.

The address specified by the X_2, B_2, and D_2 fields is inserted in bits 8-31 of the general register specified by R_1. Bits 0-7 are set to zero. The address is not inspected for availability, protection, or resolution.

The address computation follows the rules for address arithmetic. Any carries beyond the 24th bit are ignored.

Condition Code: The code remains unchanged.

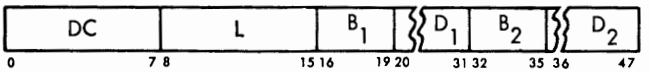
Program Interruptions: None.

Programming Note

The same general register may be specified by the R_1, X_2, and B_2 instruction field, except that general register 0 can be specified only by the R_1 field. In this manner, it is possible to increment the low-order 24 bits of a general register, other than 0, by the contents of the D_2 field of the instruction. The register to be incremented should be specified by R_1 and by either X_2 (with B_2 set to zero) or B_2 (with X_2 set to zero).

Translate

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



The eight-bit bytes of the first operand are used as arguments to reference the list designated by the second operand address. Each eight-bit function byte selected from the list replaces the corresponding argument in the first operand.

The bytes of the first operand are selected one by one for translation, proceeding left to right. Each argument byte is added to the entire initial address, the second operand address, in the low-order bit positions. The sum is used as the address of the function byte, which then replaces the original argument byte.

All data are valid. The operation proceeds until the first operand field is exhausted. The list is not altered unless an overlap occurs.

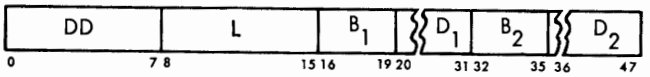
Condition Code: The code remains unchanged.

Program Interruptions:

Protection (store or fetch violation)
Addressing

Translate and Test

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



The eight-bit bytes of the first operand are used as arguments to reference the list designated by the second operand address. Each eight-bit function byte thus selected from the list is used to determine the continuation of the operation. When the function byte is a zero, the operation proceeds by fetching and translating the next argument byte. When the function byte is nonzero, the operation is completed by inserting the related argument address in general register 1, and by inserting the function byte in general register 2.

The bytes of the first operand are selected one by one for translation, proceeding from left to right. The first operand remains unchanged in storage. Fetching of the function byte from the list is performed as in TRANSLATE. The function byte retrieved from the list is inspected for the all-zero combination.

When the function byte is zero, the operation proceeds with the next operand byte. When the first operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting the condition code to 0. The contents of general registers 1 and 2 remain unchanged.

When the function byte is nonzero, the related argument address is inserted in the low-order 24 bits of general register 1. This address points to the argument last translated. The high-order eight bits of register 1 remain unchanged. The function byte is inserted in the low-order eight bits of general register 2. Bits 0-23 of register 2 remain unchanged. The condition code is set to 1 when the one or more argument bytes have not been translated. The condition code is set to 2 if the last function byte is nonzero.

Resulting Condition Code:

0 All function bytes are zero
1 Nonzero function byte before the first operand field is exhausted
2 Last function byte is nonzero
3 --

Program Interruptions:

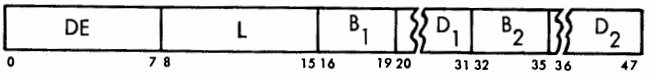
Protection (fetch violation)
Addressing

Programming Note

The TRANSLATE AND TEST is useful for scanning an input stream and locating delimiters. The stream can thus be rapidly broken into statements or data fields for further processing.

Edit

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



The format of the source (the second operand) is changed from packed to zoned and is modified under control of the pattern (the first operand). The edited result replaces the pattern.

Editing includes sign and punctuation control and the suppressing and protecting of leading zeros. It also facilitates programmed blanking for all-zero fields. Several fields may be edited in one operation, and numeric information may be combined with text.

The length field applies to the pattern (the first operand). The pattern has the zoned format and may contain any character. The source (the second operand) has the packed format and must contain valid decimal-digit and sign codes. The leftmost four bits of a source byte must contain only the codes 0000-1001; the codes 1010-1111 are recognized as a data exception and cause a program interruption. The rightmost four bits are recognized as either a sign or a decimal digit.

Both operands are processed left to right one byte at a time. Overlapping pattern and source fields give unpredictable results.

During the editing process, each character of the pattern is affected in one of three ways:

1. It is left unchanged.
2. It is replaced by a source digit expanded to zoned format.
3. It is replaced by the first character in the pattern, called the fill character.

Which of the three actions takes place is determined by one or more of the following: the type of the pattern character, the state of the significance indicator, and whether the source digit examined is zero.

Pattern Characters: There are four types of pattern characters: digit selector, significance starter, field separator, and message character. Their coding is as follows:

NAME	CODE
Digit selector	0010 0000
Significance starter	0010 0001
Field separator	0010 0010
Message character	Any other

The detection of either a digit selector or a significance starter in the pattern causes an examination to be made of the significance indicator and of a source digit. As a result, either the expanded source digit or the fill character, as appropriate, is selected to replace the pattern character. Additionally, encountering a digit selector or a significance starter may cause the significance indicator to be changed.

The field separator identifies individual fields in a multiple-field editing operation. It is always replaced in the result by the fill character, and the significance indicator is always off after the field separator is encountered.

Message characters in the pattern are either replaced by the fill character or remain unchanged in the result, depending on the state of the significance indicator. They may thus be used for padding, punctuation, or text in the significant portion of a field or for the insertion of sign-dependent symbols.

Fill Character: The fill character is obtained from the pattern as part of the editing operation: The first character of the pattern is used as the fill character. The choice of the fill character is not dependent on the code of the first pattern character and on the editing function, if any, initiated upon recognition of the code. If this character is a digit selector or significance starter, the indicated editing action is taken after the code has been assigned to the fill character.

Source Digits: Each time a digit selector or significance starter is encountered in the pattern, a new source digit is examined for placement in the pattern field. The source digit either is zoned and replaces the pattern character or is disregarded. When a sign code is detected in the four high-order bit positions, the operation is terminated.

The source digits are selected one byte at a time, and a source byte is fetched for inspection only once during an editing operation. Each source digit is examined only once for a zero value. The leftmost four bits of each byte are examined first, and the rightmost four bits, when they represent a decimal-digit code, remain available for the next pattern character that calls for a digit examination. At the time the high-order digit of a source byte is examined, the low-order four bits are checked for the existence of a sign code. When a sign code is encountered in the four rightmost bit positions, these bits are not treated as a decimal-digit code, and a new source byte is fetched from storage for the next pattern character that calls for a source-digit examination.

When the source digit is stored in the result, its code is expanded from the packed to the zoned format by attching a zone. When PSW bit 12 is zero, the preferred EBCDIC zone code 1111 is generated. When PSW bit 12 is one, the preferred USASCII-8 zone code 0101 is generated.

Significance Indicator: The significance indicator, by its on or off state, indicates the significance or nonsignificance, respectively, of subsequent source digits or message characters. Significant source digits replace their corresponding digit selectors or significance starters in the result. Significant message characters remain unchanged in the result.

The significance indicator, by its on or off state, indicates also the negative or positive value, respectively, of the source and is used as one factor in the setting of the condition code.

The indicator is set to the off state, if not already so set, at the start of the editing operation, after a field separator is encountered, or after a source byte is examined that has a plus code in the four low-order bit positions. Any of the codes 1010, 1100, 1110, and 1111 is considered a plus code.

The indicator is set to the on state, if not already so set, when a significance starter is encountered whose source digit is a valid decimal digit, or when a digit selector is encountered whose source digit is a nonzero decimal digit, and if in both instances the source byte does not have a plus code in the four low-order bit positions.

In all other situations, the indicator is not changed. A minus sign code has no effect on the significance indicator.

Result Characters: The field resulting from an editing operation replaces and is equal in length to the pattern. It is composed from pattern characters, fill characters, and zoned source digits.

If the pattern character is a message character and the significance indicator is on, the message character

remains unchanged in the result. If the pattern character is a field separator or if the significance indicator is off when a message character is encountered in the pattern, the fill character replaces the pattern character in the result.

If a digit selector or significance starter is encountered in the pattern with the significance indicator off and the source digit zero, the source digit is considered nonsignificant, and the fill character replaces the pattern character. If a digit selector or significance starter is encountered with either the significance indicator on or with a nonzero decimal source digit, the source digit is considered significant, is zoned, and replaces the pattern character in the result.

Result Condition: All digits examined are tested for the code 0000. The sign of the last field edited and whether all source digits in the field contain zeros are recorded in the condition code at the completion of the editing operation.

The condition code is made 0 when all source digits examined in the last field are zeros. When the pattern has no digit selectors or significance starters, the source is not examined, and the condition code is made 0. Similarly, the condition code is made 0 when the last character in the pattern is a field separator or when no digit selector or significance starter is encountered beyond the last field separator.

When the last field edited is nonzero and the significance indicator is on, the condition code is made 1 to indicate a result field less than zero.

When the last field edited is nonzero and the significance indicator is off, the condition code is made 2 to indicate a result field greater than zero.

For multiple-field editing operations the condition code reflects the sign and value only of the field following the last field separator.

Summary: The following table summarizes the functions of the editing operation. The leftmost four columns list all the significant combinations of the four conditions that can be encountered in the execution of an editing operation. The rightmost two columns list the action taken for each case — the type of character placed in the result field and the new setting of the significance indicator. See Appendix A for an instruction-use example of EDIT.

Resulting Condition Code:

0 Source inspected for last field is zero
1 Source inspected for last field is less than zero
2 Source inspected for last field is greater than zero
3 --

Program Interruptions:

Operation (if decimal feature is not installed)
Protection (store or fetch violation)
Addressing
Data

Programming Notes

As a rule the source is shorter than the pattern because for each source digit a zone and numeric are inserted in the result.

The total number of digit selectors and significance starters in the pattern must equal the number of source digits to be edited.

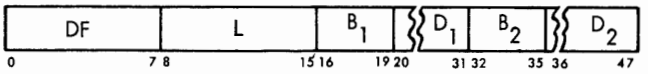
If the fill character is a blank, if no significance starter appears in the pattern, and if the source is all zeros, the editing operation blanks the result field.

CONDITIONS				RESULTS	
PATTERN CHARACTER	PREVIOUS STATE OF SIGNIFICANCE INDICATOR	SOURCE DIGIT	LOW-ORDER SOURCE DIGIT IS A PLUS SIGN	RESULT CHARACTER	STATE OF SIGNIFICANCE INDICATOR AT END OF DIGIT EXAMINATION
Digit selector	off	0	*	fill character	off
		1-9	no	source digit	on
		1-9	yes	source digit	off
	on	0-9	no	source digit	on
		0-9	yes	source digit	off
Significance starter	off	0	no	fill character	on
		0	yes	fill character	off
		1-9	no	source digit	on
		1-9	yes	source digit	off
	on	0-9	no	source digit	on
		0-9	yes	source digit	off
Field separator	*	**	**	fill character	off
Message character	off	**	**	fill character	off
	on	**	**	message character	on

*No effect on result character and new state of significance indicator.
**Not applicable because source digit not examined.

Edit and Mark

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



The format of the source (the second operand) is changed from packed to zoned and is modified under control of the pattern (the first operand). The address of each first significant result character is recorded in general register 1. The edited result replaces the pattern.

The instruction EDIT AND MARK is identical to EDIT except for the additional function of inserting the address of the result character in bit positions 8-31 of general register 1 whenever the result character is a zoned source digit and the significance indicator was off before the examination. The use of general register 1 is implied. The contents of bit positions 0-7 of the register are not changed.

Refer to Appendix A for an instruction-use example.

Resulting Condition Code:

0 Source inspected for last field is zero
1 Source inspected for last field is less than zero
2 Source inspected for last field is greater than zero
3 --

Program Interruptions:

Operation (if decimal feature is not installed)
Protection (store or fetch violation)
Addressing
Data

Programming Notes

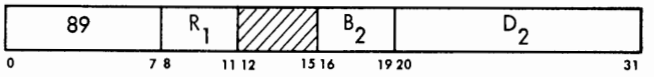
The instruction EDIT AND MARK facilitates the programming of floating currency-symbol insertion. The character address inserted in general register 1 is one more than the address where a floating currency-sign would be inserted. The BRANCH ON COUNT, with zero in the R_2 field, may be used to reduce the inserted address by one.

The character address is not stored when significance is forced. To ensure that general register 1 contains a valid address when significance is forced, it is necessary to place into the register beforehand the address of the pattern character that immediately follows the significance starter.

When a single instruction is used to edit several fields, the address of the first significant result character of each field is inserted into bit positions 8-31 of general register 1. Only the address of the first significant character of the last field is available after the instruction is completed.

Shift Left Single

SLL $R_1, D_2(B_2)$ [RS]



The first operand is shifted left the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

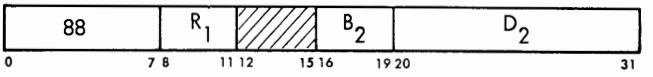
All 32 bits of the general register specified by R_1 participate in the shift. High-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated low-order register positions.

Condition Code: The code remains unchanged.

Program Interruptions: None.

Shift Right Single

SRL $R_1, D_2(B_2)$ [RS]



The first operand is shifted right the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

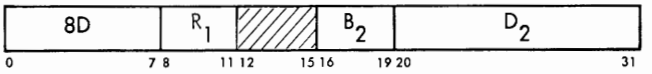
All 32 bits of the general register specified by R_1 participate in the shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated high-order register positions.

Condition Code: The code remains unchanged.

Program Interruptions: None.

Shift Left Double

SLDL $R_1, D_2(B_2)$ [RS]



The double-length first operand is shifted left the number of bits specified by the second operand address.

The R_1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. An odd value for R_1 is a specification exception and causes a program interruption. The second operand address is not used to address data; its low-

order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the even/odd register pair specified by R_1 participate in the shift. High-order bits are shifted out of the even-numbered register without inspection and are lost. Zeros are supplied to the vacated positions of the registers.

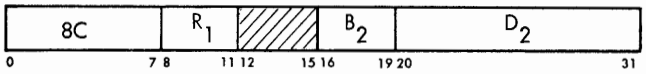
Condition Code: The code remains unchanged.

Program Interruptions:

Specification

Shift Right Double

SRDL ***R_1, $D_2(B_2)$*** ***[RS]***



The double-length first operand is shifted right the number of bits specified by the second operand address.

The R_1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. An odd value for R_1 is a specification exception and causes a program interruption. The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the even/odd register pair specified by R_1 participate in the shift. Low-order bits are shifted out of the odd-numbered register without inspection and are lost. Zeros are supplied to the vacated positions of the registers.

Condition Code: The code remains unchanged.

Program Interruptions:

Specification

Programming Note

The logical shifts differ from the arithmetic shifts in that the high-order bit participates in the shift and is not propagated, the condition code is not changed, and no overflow occurs.

Logical Operation Exceptions

Exceptional operation codes, operand designations, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in logical operations.

Operation: The decimal feature is not installed, and the instruction is EDIT or EDIT AND MARK. The instruction is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Protection: The key of an operand in storage does not match the protection key in the PSW. The operation is suppressed on a store violation. Therefore, the condition code and data in registers and storage remain unchanged. The only exceptions are the variable-length, storage-to-storage operations (those containing a length specification), which are terminated. The operation is terminated on any fetch violation. For terminated operations, the result data and condition code, if affected, are unpredictable and should not be used for further computation.

Addressing: An address designates an operand location outside the available storage for the installation: In most cases, the operation is terminated. The result data and the condition code, if affected, are unpredictable and should not be used for further computation. The exceptions are the immediate operations AND (NI), EXCLUSIVE OR (XI), OR (OI), MOVE (MVI), and STORE CHARACTER, which are suppressed.

Specification: A fullword operand in a storage-to-register operation is not located on a 32-bit boundary or an odd register address is specified for a pair of general registers containing a 64-bit operand. The operation is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Data: A digit code of the second operand in EDIT or EDIT AND MARK is invalid. The operation is terminated. The result data and the condition code are unpredictable and should not be used for further computation.

Operand addresses are tested only when used to address storage. Addresses used as a shift amount are not tested. Similarly, the address generated by the use of LOAD ADDRESS is not tested. The address restrictions do not apply to the components from which an address is generated – the contents of the D_1 and D_2 fields, and the contents of the registers specified by X_2, B_1, and B_2.

Branching

Instructions are performed by the central processing unit primarily in the sequential order of their locations. A departure from this normal sequential operation may occur when branching is performed. The branching instructions provide a means for making a two-way choice, to reference a subroutine, or to repeat a segment of coding, such as a loop.

Branching is performed by introducing a branch address as a new instruction address.

The branch address may be obtained from one of the general registers or it may be the address specified by the instruction. The branch address is independent of the updated instruction address.

The detailed operation of branching is determined by the condition code which is part of the program status word (PSW) or by the results in the general registers which are specified in the loop-closing operations.

During a branching operation, the rightmost half of the PSW, including the updated instruction address, may be stored before the instruction address is replaced by the branch address. The stored information may be used to link the new instruction sequence with the preceding sequence.

The instruction EXECUTE is grouped with the branching instructions. The branch address of EXECUTE designates a single instruction to be inserted in the instruction sequence. The updated instruction address normally is not changed in this operation, and only the instruction located at the branch address is executed.

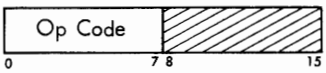
All branching operations are provided in the standard instruction set.

Normal Sequential Operation

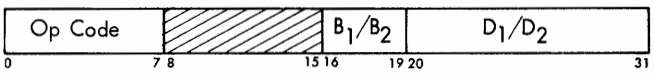
Normally, operation of the CPU is controlled by instructions taken in sequence. An instruction is fetched from a location specified by the instruction-address field of the PSW. The instruction address is increased by the number of bytes of the instruction to address the next instruction in sequence. This new instruction-address value, called the updated instruction address, replaces the previous contents of the instruction-address field in the PSW. The current instruction is executed, and the same steps are repeated, using the updated instruction address to fetch the next instruction.

Instructions occupy a halfword or a multiple thereof. An instruction may have up to three halfwords. The number of halfwords in an instruction is specified by the first two instruction bits. A 00 code indicates a halfword instruction, codes 01 and 10 indicate a two-halfword instruction, and code 11 indicates a three-halfword instruction.

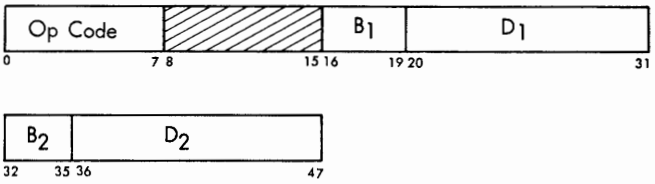
Halfword Format



Two-Halfword Format



Three-Halfword Format



Storage wraps around from the maximum addressable storage location, byte location 16,777,215, to byte location 0. An instruction having its last halfword at the maximum storage location is followed by the instruction at address 0. Also, a multiple-halfword instruction may straddle the upper storage boundary; no special indication is given in these cases.

Conceptually, an instruction is fetched from storage after the preceding operation is completed and before execution of the current operation, even though physical storage width and overlap of instruction execution with storage access may cause actual instruction fetching to be different.

A change in the sequential operation may be caused by branching, status switching, interruption, or manual intervention. Sequential operation is initiated and terminated from the system control panel.

Programming Note

It is possible to modify an instruction in storage by means of the immediately preceding instruction.

Sequential Operation Exceptions

Exceptional instruction addresses or operation codes cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. (In this manual, part of the description of each class of instruc-

tions is a list of the program interruptions that may occur for these instructions.) The new PSW is not checked for exceptions when it becomes current. These checks occur when the next instruction is executed. The following program interruptions may occur in normal instruction sequencing, independently of the instruction performed.

Operation: An operation exception occurs when the CPU attempts to decode an operation code that is not assigned. The operation exception can be accompanied by an addressing or specification exception if the instruction class associated with the undefined operation has uniform requirements for operand designation. An instruction class is a group of instructions whose four leftmost bits are identical.

Protection: A protection exception occurs when an attempt is made to fetch an instruction halfword from a fetch-protected location. This error can occur when normal instruction sequencing goes from an unprotected region into a protected region, or following a branching or load-PSW operation or an interruption.

Addressing: An addressing exception occurs when an instruction halfword is located outside the available storage for the particular installation. This situation can occur when normal instruction sequencing goes from a valid storage region into an unavailable region, or following a branching or load-PSW operation or an interruption. However, when the last locations in available storage contain an instruction that again introduces a valid instruction address (i.e., a branch), no program interruption is caused even though the updated instruction address designates an unavailable location.

Specification: A specification exception occurs when the instruction address in the PSW is odd. This odd-address error can occur only after a branching or load-PSW operation or after an interruption.

A specification exception will occur when the protection key is nonzero and the protection feature is not installed. This error can occur after a PSW is loaded or after an interruption.

In each case, the instruction is suppressed; therefore, the condition code and data in storage and registers remain unchanged. The instruction address stored as part of the old PSW has been updated by the number of halfwords indicated by the instruction length code in the old PSW.

Programming Notes

When a program interruption occurs, the current PSW is stored in the old PSW location. The instruction address stored as part of this old PSW is thus the updated instruction address, having been updated by the number of halfwords indicated in the instruction-length code of the same PSW. The interruption code in this old PSW identifies the cause of the interruption and aids in the programmed interpretation of the old PSW.

If the new PSW for a program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established which may be broken only by an external or I/O interruption. If these interruptions also have an unacceptable new PSW, new supervisor information must be introduced by initial program loading or by manual intervention.

Decision-Making

Branching may be conditional or unconditional. Unconditional branches replace the updated instruction address with the branch address. Conditional branches may use the branch address or may leave the updated instruction address unchanged. When branching takes place, the instruction is called successful; otherwise, it is called unsuccessful.

Whether a conditional branch is successful depends on the result of operations concurrent with the branch or preceding the branch. The former case is represented by BRANCH ON COUNT and the branch-on-index instructions. The latter case is represented by BRANCH ON CONDITION, which inspects the condition code that reflects the result of a previous arithmetic, logical, or I/O operation.

The condition code provides a means for data-dependent decision-making. The code is inspected to qualify the execution of the conditional-branch instructions. The code is set by some operations to reflect the result of the operation, independently of the previous setting of the code. The code remains unchanged for all other operations.

The condition code occupies bit positions 34 and 35 of the PSW. When the PSW is stored during status switching, the condition code is preserved as part of the PSW. Similarly, the condition code is stored as part of the rightmost half of the PSW in a branch-and-link operation. A new condition code is obtained by a LOAD PSW or SET PROGRAM MASK or by the new PSW loaded as a result of an interruption.

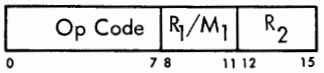
The condition code indicates the outcome of some of the arithmetic, logical, or I/O operations. It is not changed for any branching operation, except for EXECUTE. In the case of EXECUTE, the condition code is set or left unchanged by the subject instruction, as would have been the case had the subject instruction been in the normal instruction stream.

The table at the end of this section lists all instructions capable of altering the condition code and the meaning of the codes for these instructions.

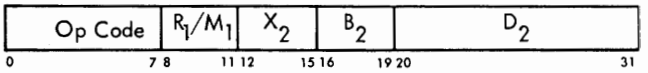
Instruction Formats

Branching instructions use the following three formats:

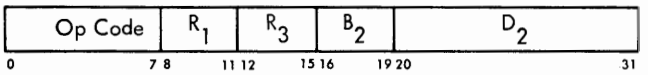
RR Format



RX Format



RS Format



In these formats R_1 specifies the address of a general register. In BRANCH ON CONDITION a mask field (M_1) identifies the bit values of the condition code. The branch address is defined differently for the three formats.

In the RR format, the R_2 field specifies the address of a general register containing the branch address, except when R_2 is zero, which indicates no branching. The same register may be specified by R_1 and R_2.

In the RX format, the contents of the general registers specified by the X_2 and B_2 fields are added to the content of the D_2 field to form the branch address.

In the RS format, the content of the general register specified by the B_2 field is added to the content of the D_2 field to form the branch address. The R_3 field in this format specifies the location of the second operand and implies the location of the third operand. The first operand is specified by the R_1 field. The third operand location is always odd. If the R_3 field specifies an even register, the third operand is obtained from the next higher addressed register. If the R_3 field specifies an odd register, the third operand location coincides with the second operand location.

A zero in a B_2 or X_2 field indicates the absence of the corresponding address component.

An instruction can specify the same general register for both address modification and operand location. The order in which the contents of the general registers are used for the different parts of an operation is:

1. Address computation.
2. Arithmetic or link information storage.
3. Replacement of the instruction address by the branch address obtained under step 1.

Results are placed in the general register specified by R_1. Except for the storing of the final results, the contents of all general registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction. For BRANCH ON INDEX HIGH, for example, BXH is the mnemonic and R_1, R_3, $D_2(B_2)$ the operand designation.

Programming Note

In several instructions the branch address may be specified in two ways: in the RX format, the branch address is the address specified by X_2, B_2, and D_2; in the RR format, the branch address is in the low-order 24 bits of the register specified by R_2. Note that the relation of the two formats in branch-address specification is not the same as in operand-address specification. For operands, the address specified by X_2, B_2, and D_2 is the operand address, but the register specified by R_2 contains the operand itself.

Branching Instructions

The branching instructions and their mnemonics, formats, and operation codes follow. The table also shows the exceptions that cause a program interruption during execution of EXECUTE. The subject instruction of EXECUTE follows its own rules for interruptions. The condition code is never changed for branching instructions.

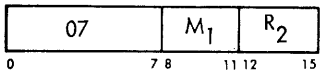
NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE
Branch on Condition	BCR	RR		07
Branch on Condition	BC	RX		47
Branch and Link	BALR	RR		05
Branch and Link	BAL	RX		45
Branch on Count	BCTR	RR		06
Branch on Count	BCT	RX		46
Branch on Index High	BXH	RS		86
Branch on Index Low or Equal	BXLE	RS		87
Execute	EX	RX	P,A,S, EX	44

NOTES

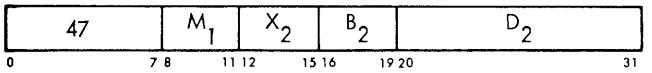
A	Addressing exception
EX	Execute exception
S	Specification exception
P	Protection exception

Branch On Condition

BCR M_1, R_2 [RR]



BC $M_1, D_2(X_2, B_2)$ [RX]



The updated instruction address is replaced by the branch address if the state of the condition code is as specified by M_1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The M_1 field is used as a four-bit mask. The four bits of the mask correspond, left to right, with the four condition codes (0, 1, 2, and 3) as follows:

INSTRUCTION BIT	MASK POSITION VALUE	CONDITION CODE
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the condition code has a corresponding mask bit of one.

Condition Code: The code remains unchanged.

Program Interruptions: None.

Programming Note

When a branch is to be made on more than one condition code, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made on condition codes 0 and 1.

When all four mask bits are ones, that is, the mask position value is 15, the branch is unconditional. When all four mask bits are zero or when the R_2 field in the RR format contains zero, the branch instruction is equivalent to a no-operation.

Condition-Code Settings

	CODE STATE 0	1	2	3
Fixed-Point Arithmetic				
Add H/F	zero	< zero	> zero	overflow
Add Logical	zero, no carry	not zero, no carry	zero, carry	not zero, carry
Compare H/F	equal	low	high	--
Load and Test	zero	< zero	> zero	carry
Load Complement	zero	< zero	> zero	overflow
Load Negative	zero	< zero	--	--
Load Positive	zero	--	> zero	overflow
Shift Left Double	zero	< zero	> zero	overflow
Shift Left Single	zero	< zero	> zero	overflow
Shift Right Double	zero	< zero	> zero	--
Shift Right Single	zero	< zero	> zero	--
Subtract H/F	zero	< zero	> zero	overflow
Subtract Logical	--	not zero, no carry	zero, carry	not zero, carry

	CODE STATE 0	1	2	3
Decimal Arithmetic				
Add Decimal	zero	< zero	> zero	overflow
Compare Decimal	equal	low	high	--
Subtract Decimal	zero	< zero	> zero	overflow
Zero and Add	zero	< zero	> zero	overflow
Floating-Point Arithmetic				
Add Normalized S/L	zero	< zero	> zero	--
Add Unnormalized S/L	zero	< zero	> zero	--
Compare S/L	equal	low	high	--
Load and Test S/L	zero	< zero	> zero	--
Load Complement S/L	zero	< zero	> zero	--
Load Negative S/L	zero	< zero	--	--
Load Positive S/L	zero	--	> zero	--
Subtract Normalized S/L	zero	< zero	> zero	--
Subtract Unnormalized S/L	zero	< zero	> zero	--
Logical Operations				
And	zero	not zero	--	--
Compare Logical	equal	low	high	--
Edit	zero	< zero	> zero	--
Edit and Mark	zero	< zero	> zero	--
Exclusive Or	zero	not zero	--	--
Or	zero	not zero	--	--
Test Under Mask	zero	mixed	--	one
Translate and Test	zero	incomplete	complete	--
Status Switching				
Test and Set	zero	one	--	--
Input/Output Operations				
Halt I/O	interruption pending	CSW stored	burst op stopped	not operational
Start I/O	successful	CSW stored	busy	not operational
Test Channel	available	interruption pending	burst mode	not operational
Test I/O	available	CSW stored	busy	not operational

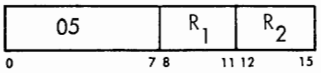
NOTES

available	Unit and channel available
burst op stopped	Burst operation stopped
busy	Unit or channel busy
carry	A carryout of the sign position occurs
complete	Last result byte nonzero
CSW stored	Chanel status word stored
equal	Operands compare equal
F	Fullword
> zero	Result is greater than zero
H	Halfword
halted	Data transmission stopped. Unit in halt-reset mode
high	First operand compares high
incomplete	Nonzero result byte; not last
L	Long precision
< zero	Result is less than zero
low	First operand compares low
mixed	Selected bits are both zero and one
not operational	Unit or channel not operational
not zero	Result is not all zero
one	Selected bits are one
overflow	Result overflows
S	Short precision
zero	Result or selected bits are zero

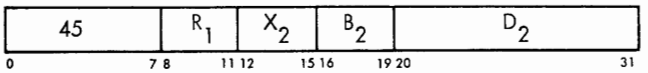
NOTE: The condition code also may be changed by LOAD PSW, SET PROGRAM MASK, and DIAGNOSE and by an interruption.

Branch and Link

BALR R_1, R_2 [*RR*]



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



The rightmost 32 bits of the PSW, including the updated instruction address, are stored as link information in the general register specified by R_1. Subsequently, the instruction address is replaced by the branch address.

The branch address is determined before the link information is stored. The link information contains the instruction length code, the condition code, and the program mask bits, as well as the updated instruction address. The instruction-length code is 1 or 2, depending on the format of the BRANCH AND LINK.

Condition Code: The code remains unchanged.

Program Interruptions: None.

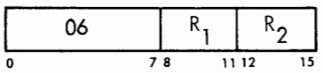
Programming Note

The link information is stored without branching when in the RR format the R_2 field contains zero.

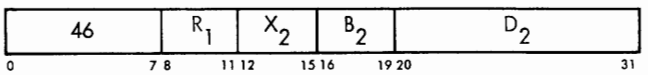
When BRANCH AND LINK is the subject instruction of EXECUTE, the instruction-length code is 2.

Branch On Count

BCTR R_1, R_2 [*RR*]



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



The content of the general register specified by R_1 is algebraically reduced by one. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address is replaced by the branch address.

The branch address is determined prior to the counting operation. Counting does not change the condition code. The overflow occurring on transition from the maximum negative number to the maximum positive number is ignored. Otherwise, the subtraction proceeds as in fixed-point arithmetic, and all 32 bits of the general register participate in the operation.

Condition Code: The code remains unchanged.

Program Interruptions: None.

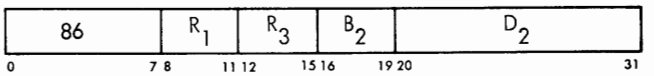
Programming Notes

An initial count of one results in zero, and no branching takes place. An initial count of zero results in minus one and causes branching to be executed.

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

Branch On Index High

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



An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first operand location, regardless of whether the branch is taken. When the sum is high, the instruction address is replaced by the branch address. When the sum is low or equal, instruction sequencing proceeds with the updated instruction address.

The first operand and the increment are in the registers specified by R_1 and R_3. The comparand register address is odd and is either one larger than R_3 or equal to R_3. The branch address is determined prior to the addition and comparison.

Overflow caused by the addition is ignored and does not affect the comparison. Otherwise, the addition and comparison proceed as in fixed-point arithmetic. All 32 bits of the general registers participate in the operations, and negative quantities are expressed in two's-complement notation. When the first operand and comparand locations coincide, the original register contents are used as the comparand.

Condition Code: The code remains unchanged.

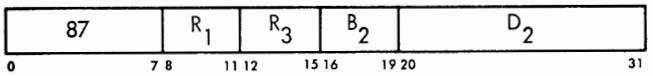
Program Interruptions: None.

Programming Note

The name "branch on index high" indicates that one of the major purposes of this instruction is the incrementing and testing of an index value. The increment may be algebraic and of any magnitude.

Branch On Index Low or Equal

BXLE $R_1, R_3, D_2(B_2)$ [*RS*]



An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first operand

location, regardless of whether the branch is taken. When the sum is low or equal, the instruction address is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address.

The first operand and the increment are in the registers specified by R_1 and R_3. The comparand register address is odd and is either one larger than R_3 or equal to R_3. The branch address is computed prior to the addition and comparison.

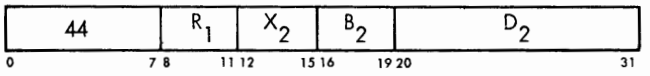
This instruction is similar to BRANCH ON INDEX HIGH, except that the branch is successful when the sum is low or equal compared to the comparand.

Condition Code: The code remains unchanged.

Program Interruptions: None.

Execute

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



The single instruction at the branch address is modified by the content of the general register specified by R_1, and the resulting subject instruction is executed.

Bits 8-15 of the instruction designated by the branch address are OR'ed with bits 24-31 of the register specified by R_1, except when register 0 is specified, which indicates that no modification takes place. The subject instruction may be 16, 32, or 48 bits in length. The OR'ing does not change either the content of the register specified by R_1 or the instruction in storage and is effective only for the interpretation of the instruction to be executed.

The execution and exception handling of the subject instruction are exactly as if the subject instruction were obtained in normal sequential operation, except for instruction address and instruction-length recording.

The instruction address of the PSW is increased by the length of EXECUTE. This updated address and the length code (2) of EXECUTE are stored in the PSW in the event of a branch-and-link subject instruction or in the event of an interruption.

When the subject instruction is a successful branching instruction, the updated instruction address of the PSW is replaced by the branch address of the subject instruction. When the subject instruction in turn is an EXECUTE, an execute exception occurs and results in a program interruption. The effective address of EXECUTE must be even; if not, a specification exception will cause a program interruption.

Condition Code: The code may be set by the subject instruction.

Program Interruptions:

Execute
Protection (fetch violation)
Addressing
Specification

Programming Notes

The OR'ing of eight bits from the general register with the designated instruction permits indirect length, index, mask, immediate data, and arithmetic-register specification.

If the subject instruction is a successful branch, the length code still stands at 2.

An addressing or specification exception may be caused by EXECUTE or by the subject instruction.

Execute Exceptions

Exceptional operand designations and a subject-instruction operation code specifying EXECUTE cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause. Exceptions that cause a program interruption in the use of EXECUTE are:

Execute: An EXECUTE instruction has as its subject instruction another EXECUTE.

Protection: An EXECUTE specifies a subject instruction halfword in a fetch-protected area.

Addressing: The branch address of EXECUTE designates an instruction-halfword location outside the available storage for the particular installation.

Specification: The branch address of EXECUTE is odd.

These four exceptions occur only for EXECUTE. The instruction is suppressed. Therefore, the condition code and data in registers and storage remain unchanged.

Exceptions arising for the subject instruction of EXECUTE are the same as would have arisen had the subject instruction been in the normal instruction stream. However, the instruction address stored in the old PSW is the address of the instruction following EXECUTE. Similarly, the instruction-length code in the old PSW is the instruction-length code (2) of EXECUTE.

The address restrictions do not apply to the components from which an address is generated — the content of the D_1 field and the content of the register specified by B_1.

Programming Note

An unavailable or odd branch address of a successful branch is detected during the execution of the next instruction and not as part of the branch.

Status Switching

A set of operations is provided to switch the status of the CPU, of storage, and of communication between systems.

The over-all CPU status is determined by several program-state alternatives, each of which can be changed independently to its opposite and most of which are indicated by a bit in the program status word (PSW). The CPU status is further defined by the instruction address, the condition code, the instruction-length code, the storage-protection key, and the interruption code. These all occupy fields in the PSW.

Protection of main storage is achieved by matching a key in storage with a protection key in the PSW or in a channel. The protection status of storage may be changed by introducing new storage keys, using SET STORAGE KEY. The storage keys may be inspected by using INSERT STORAGE KEY.

Facilities are provided whereby a system formed by CPU, storage, and I/O can communicate with other systems. The instruction READ DIRECT makes signals available to the CPU; WRITE DIRECT provides signals to other systems.

All status-switching instructions, other than those of the protection feature or direct control feature, are provided in the standard instruction set.

Program States

The four types of program-state alternatives, which determine the over-all CPU status, are named Problem/Supervisor, Wait/Running, Masked/Interruptible, and Stopped/Operating. These states differ in the way they affect the CPU functions and in the way their status is indicated and switched. The masked states have several alternatives; all other states have only one alternative.

All program states are independent of each other in their function, indication, and status switching. Status switching does not affect the contents of the arithmetic registers or the execution of I/O operations but may affect the timer operation.

Problem State

The choice between supervisor and problem state determines whether the full set of instructions is valid. The names of these states reflect their normal use.

In the problem state all I/O, protection, and direct-control instructions are invalid, as well as LOAD PSW, SET SYSTEM MASK, and DIAGNOSE. These are called privileged instructions. A privileged instruction encountered in the problem state constitutes a privileged-operation exception and causes a program interruption. In the supervisor state all instructions are valid.

When bit 15 of the PSW is zero, the CPU is in the supervisor state. When bit 15 is one, the CPU is in the problem state. The supervisor state is not indicated on the operator sections of the system control panel.

The CPU is switched between problem and supervisor state by changing bit 15 of the PSW. This bit can be changed only by introducing a new PSW. Thus status switching may be performed by LOAD PSW, using a new PSW with the desired value for bit 15. Since LOAD PSW is a privileged instruction, the CPU must be in the supervisor state prior to the switch. A new PSW is also introduced when the CPU is interrupted. The SUPERVISOR CALL causes an interruption and thus may change the CPU state. Similarly, initial program loading introduces a new PSW and with it a new CPU state. The new PSW may introduce the problem or supervisor state regardless of the preceding state. No explicit operator control is provided for changing the supervisor state.

Timer updating is not affected by the choice between supervisor and problem state.

Programming Note

To allow return from an interruption-handling routine to a preceding program by a LOAD PSW, the PSW for the interruption routine should specify the supervisor state.

Wait State

In the wait state no instructions are processed, and storage is not addressed repeatedly for this purpose, whereas in the running state, instruction fetching and execution proceed in the normal manner.

When bit 14 of the PSW is one, the CPU is waiting. When bit 14 is zero, the CPU is in the running state. The wait state is indicated on the operator control section of the system control panel by the wait light.

The CPU is switched between wait and running state by changing bit 14 of the PSW. This bit can be changed only by introducing an entire new PSW, as is the case with the problem-state bit. Thus, switching from the running state may be achieved by the privileged instruction LOAD PSW, by an interruption such as for

SUPERVISOR CALL, or by initial program loading. Switching from the wait state may be achieved by an I/O or external interruption or, again, by initial program loading. The new PSW may introduce the wait or running state regardless of the preceding state. No explicit operator control is provided for changing the wait state.

Timer updating is not affected by the choice between running and wait state.

Programming Note

To leave the wait state without manual intervention, the CPU should remain interruptible for some active I/O or external interruption source.

Masked States

The CPU may be masked or interruptible for all I/O, external, and machine-check interruptions and for some program interruptions. When the CPU is interruptible for a class of interruptions, these interruptions are accepted. When the CPU is masked, the system interruptions remain pending, while the program and machine-check interruptions are ignored.

The system mask bits (PSW bits 0-7), the program mask bits (PSW bits 36-39), and the machine-check mask bit (PSW bit 13) indicate as a group the masked state of the CPU. When a mask bit is one, the CPU is interruptible for the corresponding interruptions. When the mask bit is zero, these interruptions are masked off. The system mask bits indicate the masked state of the CPU for multiplexor and selector channels and the external signals. The program mask bits indicate the masked state for four of the 15 types of program exceptions. The machine-check mask bit pertains to all machine checks. Program interruptions not maskable, as well as the supervisor-call interruption, are always taken. The masked states are not indicated on the operator sections of the system control panel.

Most mask bits do not affect the execution of CPU operations. The only exception is the significance mask bit, which determines the manner in which a floating-point operation is completed when a significance exception occurs.

The interruptible state of the CPU is switched by changing the mask bits in the PSW. The program mask may be changed separately by SET PROGRAM MASK, and the system mask may be changed separately by the privileged instruction SET SYSTEM MASK. The machine-check mask bit can be changed only by introducing an entire new PSW, as is the case with the problem-state and wait-state bits. Thus, a change in the entire masked status may be achieved by the privileged instruction LOAD PSW, by an interruption such as for SUPERVISOR CALL, or by initial program loading. The new PSW may introduce a new masked state regardless of the preceding state. No explicit operator control is provided for changing the masked state.

Timer updating is not affected by the choice between masked or interruptible states.

Programming Note

To prevent an interruption-handling routine from being interrupted before necessary housekeeping steps are performed, the new PSW for that interruption should mask the CPU for further interruptions of the kind that caused the interruption.

Stopped State

When the CPU is in the stopped state, instructions and interruptions are not executed. In the operating state, the CPU executes instructions (if not waiting) and interruptions (if not masked off).

The stopped state is indicated on the operator control section of the system control panel by the manual light. The stopped state is not identified by a bit in the PSW.

A change in the stopped or operating state can be effected only by manual intervention or by machine malfunction. No instructions or interruptions can stop or start the CPU. The CPU is commanded to stop when the stop key on the operator intervention section of the system control panel is pressed, when an address comparison indicates equality, and when the rate switch is set to INSTRUCTION STEP. In addition, the CPU is placed in the stopped state after power is turned on or following a system reset, except during initial program loading. The CPU is placed in the operating state when the start key on the operator intervention panel is pressed. The CPU is also placed in the operating state when initial program loading is commenced.

The transition from operating to stopped state occurs at the end of instruction execution and prior to starting the next instruction execution. When the CPU is in the wait state, the transition takes place immediately. All interruptions pending and not masked off are taken while the CPU is still in the operating state. They cause an old PSW to be stored and a new PSW to be fetched before entering the stopped state. When the CPU is in the stopped state, interruptions are not taken and remain pending.

The timer is not updated in the stopped state.

Programming Notes

Except for timing considerations and response to equipment errors, execution of a program is not affected by stopping the CPU.

When because of machine malfunction the CPU is unable to end an instruction, the stop key is not effective, and initial program loading or system reset should be used.

Input/output operations continue to completion while the CPU is in the problem, wait, masked, or stopped state. However, no new I/O operations can be initiated while the CPU is stopped, waiting, or in the problem state. Also, the interruption caused by I/O completion remains pending when masked off or when the CPU is in the stopped state.

Protection

Protection is provided to protect the contents of certain areas of main storage from destruction (or misuse) caused by erroneous storing (or storing and fetching) of information during the execution of a program. Locations may be protected against store violations or against store and fetch violations but never against fetch violations alone. This protection is achieved by identifying blocks of storage with a key and comparing this key with a protection key supplied with the data to be stored. The detection of a mismatch causes the access to be suppressed, and a protection exception is recognized.

Area Identification

For protection purposes, main storage is divided into blocks of 2,048 bytes, each block having an address that is a multiple of 2,048.

Protection Action

A key is associated with each block of storage. The key consists of five bit positions and may be used to establish the right of access. When protection only against destruction is provided, the low-order bit is ignored. When both store and fetch protection is provided, the low-order bit of the five-bit key in storage designates whether the block is protected against fetch-type references. A zero in the low-order bit position indicates that only store-type references are monitored; a one indicates that protection applies to both fetching and storing. The same key setting may be used in many blocks.

When protection applies to a storage reference, the key in storage is compared with the protection key. Access to storage is permitted only when the key in storage matches the protection key in the PSW or in the channel. The keys are said to match when the four high-order bits of the key in storage are equal to the protection key or when the protection key is zero. The protection key of the current PSW is used as the comparand when the operation is specified by an instruction. When the reference is specified by a channel operation, the protection key supplied to the channel by the channel address word is used as the comparand.

The key in storage is not part of addressable storage. The key is changed by SET STORAGE KEY and is inspected by INSERT STORAGE KEY. The protection key of the CPU occupies bits 8-11 of the PSW. The protection key for I/O operations is specified in bit positions 0-3 of the channel address word and is recorded in bits 0-3 of the channel status word stored as a result of the I/O operation.

The protection system is always active. It is independent of the problem, supervisor, or masked state of the CPU and of the type of instruction or I/O command being executed.

When an instruction causes a protection mismatch, the protected main-storage location remains unchanged.

In general, a store violation by the CPU program causes the instruction specifying this location to be suppressed when possible, that is, to be omitted entirely. The operation is terminated only when a protection exception is recognized after execution of the instruction has progressed to the point that suppression is precluded. Fetch violations cause the operation to be terminated.

Protection mismatch due to an I/O operation is indicated in the channel status word stored as a result of the operation.

Protection is optional on some models. The fetch-protection feature requires the presence of the store-protection feature.

Programming Note

When protection is not installed, the protection key in the PSW and the protection key of the channels must be zero; otherwise, a program interruption or program-check I/O termination occurs.

When the fetch-protection feature is not installed, bit 28 of the register specified by the R_1 field of SET STORAGE KEY is ignored, and during execution of INSERT STORAGE KEY, bit 28 of the register is set to zero.

Locations Protected

All main-storage locations where information is stored or fetched in the course of an operation are subject to protection. A location not actually used does not cause protection action.

Locations whose addresses are generated by the CPU for updating or interruption purposes, such as the timer, channel status word, or PSW addresses, are not protected. However, when the program specifies these locations, they are subject to protection.

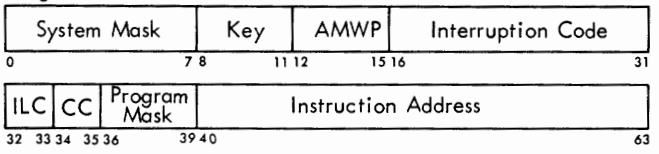
Program Status Word

The PSW contains all information not contained in storage or registers but required for proper program execution. By storing the PSW, the program can preserve the detailed status of the CPU for subsequent inspection. By loading a new PSW or part of a PSW, the state of the CPU may be changed.

In certain circumstances all of the PSW is stored or loaded; in others, only part of it. The entire PSW is stored, and a new PSW is introduced when the CPU is interrupted. The rightmost 32 bits are stored in BRANCH AND LINK. The LOAD PSW introduces a new PSW; SET PROGRAM mask introduces a new condition code and program-mask field in the PSW; SET SYSTEM MASK introduces a new system-mask field.

The PSW has the following format:

Program Status Word



The following is a summary of the purposes of the PSW fields:

System Mask: Bits 0-7 of the PSW are associated with I/O channels and external signals as specified in the following table. When a mask bit is one, the source can interrupt the CPU. When a mask bit is zero, the corresponding source can not interrupt the CPU, and interruptions remain pending.

SYSTEM MASK BIT	INTERRUPTION SOURCE
0	Channel 0
1	Channel 1
2	Channel 2
3	Channel 3
4	Channel 4
5	Channel 5
6	Channel 6
7	Timer
7	Interrupt key
7	External signal

Protection Key: Bits 8-11 of the PSW form the CPU protection key. The key is matched with a storage key whenever a result is stored or whenever information is fetched from a location that is protected against fetching. When protection is not implemented, bits 8-11 must be zero when loaded; otherwise, a specification exception is recognized when an attempt is made to execute the instruction designated by the PSW. The protection key is stored unchanged.

ASCII(A): When bit 12 of the PSW is one, the codes preferred for the USASCII-8 code are generated for decimal results. When PSW bit 12 is zero, the codes preferred for the extended binary-coded-decimal interchange code are generated.

The following instructions cause either the sign or zone code to be generated in accordance with the setting of PSW bit 12:

Add Decimal	(sign code)
Subtract Decimal	(sign code)
Zero and Add	(sign code)
Muliply Decimal	(sign code)
Divide Decimal	(sign code)
Unpack	(zone code only)
Convert to Decimal	(sign code)
Edit	(zone code)
Edit and Mark	(zone code)

Machine-Check Mask (M): When bit 13 of the PSW is one, detection of a machine-check condition causes a machine-check interruption, generation of the machine-check-out signal and logging of diagnostic information. When bit 13 of the PSW is zero, the CPU is disabled for machine-check interruptions, the associated signal and any diagnostic procedures do not take place, and the machine-check condition remains pending.

Wait State (W): When bit 14 of the PSW is one, the CPU is in the wait state. When PSW bit 14 is zero, the CPU is in the running state.

Problem State (P): When bit 15 of the PSW is one, the CPU is in the problem state. When PSW bit 15 is zero, the CPU is in the supervisor state.

Interruption Code: Bits 16-31 of the PSW identify the cause of an interruption. Use of the code for all five interruption types is shown in a table appearing in the "Interruptions" section.

Instruction Length Code (ILC): The code in PSW bits 32 and 33 indicates the length, in halfwords, of the last-interpreted instruction when a program or supervisor-call interruption occurs. The code is unpredictable for I/O, external, or machine-check interruptions. Encoding of these bits is summarized in a table appearing in the "Interruptions" sections.

Condition Code (CC): Bits 34 and 35 of the PSW are the two bits of the condition code. The condition codes for all instructions are summarized in a table appearing in the "Branching" section.

Program Mask: Bits 36-39 of the PSW are the four program mask bits. Each bit is associated with a program exception, as specified in the following table. When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The significance mask bit also determines the manner in which floating-point addition and subtraction are completed.

PROGRAM MASK BIT	PROGRAM EXCEPTION
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow
39	Significance

Instruction Address: Bits 40-63 of the PSW are the instruction address. This address specifies the leftmost eight-bit byte position of the next instruction. Bit 63 must be zero when loaded; otherwise, a specification exception is recognized when an attempt is made to execute the instruction designated by the PSW.

Programming Note

The new PSW is not checked for exceptions when the new PSW becomes current. These checks are made when the next instruction is executed.

Multisystem Operation

Various facilities are provided to permit communication between individual systems. Messages may be transmitted by means of a shared I/O device, a channel-to-channel adapter, or a shared storage unit. Signaling may be accomplished when the direct control feature is installed by WRITE DIRECT and READ DIRECT and by the signal-in lines of the external interruption.

These facilities are augmented by the ability to relocate direct addressed locations, to signal the machine malfunction of one system to another, and to initiate system operation from another system.

Direct Address Relocation

Addresses 0-4095 can be generated without a base address or index. This property is important when the PSW and general register contents must be preserved and restored during program switching. These addresses further include all addresses generated by the CPU for fixed locations, such as old PSW, new PSW, channel address word, channel status word, and timer.

This set of addresses can be relocated by means of a main prefix to permit more than one CPU to use one uniquely addressed storage. Furthermore, an alternate prefix is provided to permit a change in relocation in case storage malfunction occurs or reconfiguration becomes otherwise desirable.

A prefix is used whenever an address has the high-order 12 bits all-zero. The use of the prefix is independent of the manner in which the address is generated and does not apply to the components, such as the base or index registers, from which the address is generated. The use of the prefix applies both to addresses obtained from the program (CPU or I/O) and to fixed addresses generated by the CPU or channel for updating or interruption purposes.

Both the main prefix and alternate prefix occupy 12 bit positions. One or the other replaces the 12 high-order bit positions of the address when these are found to contain zero.

The choice of main or alternate prefix is determined by the prefix trigger. This trigger is set during initial program loading (IPL) and remains unchanged until the next initial program loading occurs. Manual IPL sets the prefix trigger to the state of the prefix-select switch on the operator control section of the system control panel. External start sets the prefix trigger to the state indicated by the signal line used. The state of the prefix is indicated by the alternate-prefix light on the operator intervention section of the system control panel.

The prefixes can be changed by hand within 5 minutes from one prewired encoding to another. The low-order four bits of a prefix always have even parity, and the total number of one-bits in a prefix cannot exceed seven.

Malfunction Indication

A machine check out-signal occurs whenever a machine check is recognized and the machine-check mask bit is one. The signal has 0.5-microsecond to 1.0-microsecond duration and is identical in electronic characteristics to the signals on the signal-out lines of the direct control feature.

The machine check-out signal is given during machine-check handling and has a high probability of being issued in the presence of machine malfunction.

System Initialization

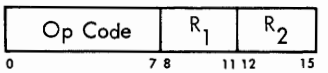
A main external-start line and an alternate external-start line respond to 0.5-microsecond to 1.0-microsecond pulses. Either line, when pulsed, sets the prefix trigger to the state indicated by its name and subsequently starts CPU operation. (Refer to "Initial Program Loading.")

The definition of the signal to which these lines respond is identical in electronic characteristic to the definition for the signal-in lines of the external interruption.

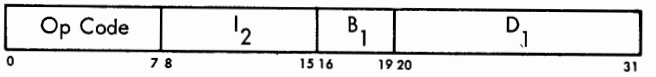
Instruction Format

Status-switching instructions use the following two formats:

RR Format



SI Format



In the RR format, the R_1 field specifies a general register, except for SUPERVISOR CALL. The R_2 field specifies a general register in SET STORAGE KEY and INSERT STORAGE KEY. The R_1 and R_2 fields in SUPERVISOR CALL contain an identification code. In SET PROGRAM MASK the R_2 field is ignored.

In the SI format the eight-bit immediate field (I_2) of the instruction contains an identification code. The I_2 field is ignored in LOAD PSW, SET SYSTEM MASK, and TEST AND SET. The content of the general register specified by B_1 is added to the content of the D_1 field to form an address designating the location of an operand in storage. Only one operand location is required in status-switching operations.

A zero in the B_1 field indicates the absence of the corresponding address component.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction. For LOAD PSW, for example, LPSW is the mnemonic and D_1 (B_1) the operand designation.

Instructions

The status-switching instructions and their mnemonics, formats, and operation codes follow. The table also indicates the feature to which an instruction belongs and the exceptions in instruction and operand designation that cause a program interruption.

NAME	MNEMONIC	TYPE		EXCEPTIONS	CODE
Load PSW	LPSW	SI	L	M,P,A,S	82
Set Program Mask	SPM	RR	L		04
Set System Mask	SSM	SI		M,P,A	80
Supervisor Call	SVC	RR			0A
Set Storage Key	SSK	RR	Z	M, A,S	08
Insert Storage Key	ISK	RR	Z	M, A,S	09
Write Direct	WRD	SI	Y	M,P,A	84
Read Direct	RDD	SI	Y	M,P,A	85
Diagnose	–	SI		M,P,A,S	83
Test and Set	TS	SI	C	P,A	93

NOTES

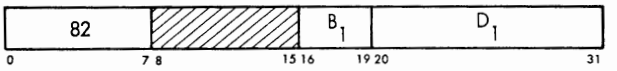
A	Addressing exception
C	Condition code is set
L	New condition code loaded
M	Privileged-operation exception
P	Protection exception
S	Specification exception
Y	Direct control feature
Z	Protection feature

Programming Note

The program status is also switched by interruptions, initial program loading, and manual control.

Load PSW

LPSW $D_1(B_1)$ [*SI*]



The double word at the location designated by the operand address replaces the PSW.

The operand address must have its three low-order bits zero to designate a double word; otherwise, a specification exception results in a program interruption.

The double word which is loaded becomes the PSW for the next sequence of instructions. Bits 8-11 become the new protection key. Bits 40-63 of the double word become the new instruction address. The PSW is not checked for program interruptions during the load-PSW operation. These checks occur as part of the execution of the next instructions.

The interruption code in bit positions 16-31 of the new PSW is not retained as the PSW is loaded. When the PSW is subsequently stored because of an interruption, these bit positions contain a new code. Similarly, bits 32 and 33 of the PSW are not retained upon loading. They will contain the instruction-length code for the last-interpreted instruction when the PSW is stored during a branch-and-link operation or during a program or supervisor-call interruption.

Condition Code: The code is set according to bits 34 and 35 of the new PSW loaded.

Program Interruptions:

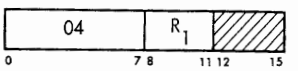
Privileged operation
Protection (fetch violation)
Addressing
Specification

Programming Note

The CPU enters the problem state when LOAD PSW loads a double word with a one in bit position 15 and similarly enters the wait state if bit position 14 is one. The LOAD PSW is the only instruction available for entering the problem state or the wait state.

Set Program Mask

SPM R_1 [*RR*]



Bits 2-7 of the general register specified by the R_1 field replace the condition code and the program mask bits of the current PSW.

Bits 0, 1, and 8-31 of the register specified by the R_1 field are ignored. The contents of the register specified by the R_1 field remain unchanged.

The instruction permits setting of the condition code and the mask bits in either the problem or supervisor state.

Condition Code: The code is set according to bits 2 and 3 of the register specified by R_1.

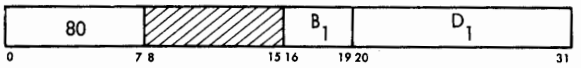
Program Interruptions: None.

Programming Note

Bits 2-7 of the general register may have been loaded from the PSW by BRANCH AND LINK.

Set System Mask

SSM $D_1(B_1)$ [*SI*]



The byte at the location designated by the operand address replaces the system mask bits of the current PSW.

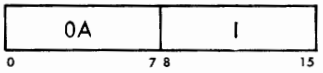
Condition Code: The code remains unchanged.

Program Interruptions:

Privileged operation
Protection (fetch violation)
Addressing

Supervisor Call

SVC *I* [*RR*]



The instruction causes a supervisor-call interruption, with the I field of the instruction providing the interruption code.

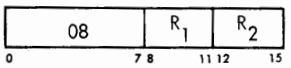
The contents of bit positions 8-15 of the instruction are placed in bit positions 24-31 of the old PSW which is stored in the course of the interruption. Bit positions 16-23 of the old PSW are made zero. The old PSW is stored at location 32, and a new PSW is obtained from location 96. The instruction is valid in both problem and supervisor state.

Condition Code: The code remains unchanged in the old PSW.

Program Interruptions: None.

Set Storage Key

SSK R_1, R_2 [*RR*]



The key of the storage block addressed by the register designated by R_2 is set according to the key in the register designated by R_1.

The storage block of 2,048 bytes, located on a multiple of the block length, is addressed by bits 8-20 of the register designated by the R_2 field. Bits 0-7 and 21-27 of this register are ignored. Bits 28-31 of the register must be zero; otherwise, a specification exception causes a program interruption.

The five-bit key is obtained from bits 24-28 of the register designated by the R_1 field. Bits 0-23 and 29-31 of this register are ignored. When fetch protection is not installed, bit 28 of the register specified by the R_1 field is ignored.

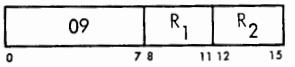
Condition Code: The code remains unchanged.

Program Interruptions:

Operation (if protection feature is not installed)
Privileged operation
Addressing
Specification

Insert Storage Key

ISK R_1, R_2 [*RR*]



The key of the storage block addressed by the register designated by R_2 is inserted in the register designated by R_1.

The storage block of 2,048 bytes, located on a multiple of the block length, is addressed by bits 8-20 of the register designated by the R_2 field. Bits 0-7 and 21-27 of this register are ignored. Bits 28-31 of the register must be zero; otherwise, a specification exception causes a program interruption. The five-bit key is inserted in bits 24-28 of the register specified by the R_1 field. Bits 0-23 of this register remain unchanged, and bits 29-31 are set to zero. When fetch protection is not installed, bit 28 of the register specified by the R_1 field is set to zero.

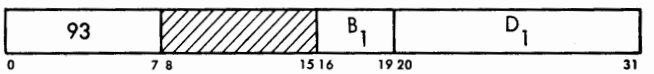
Condition Code: The code remains unchanged.

Program Interruptions:

Operation (if protection feature is not installed)
Privileged operation
Addressing
Specification

Test and Set

TS $D_1(B_1)$ [*SI*]



The leftmost bit (bit position 0) of the byte located at the first operand address is used to set the condition code, and the entire addressed byte is set to all ones.

The byte in storage is set to all ones as it is fetched for the testing of bit position 0. No other access to this location is permitted between the moment of fetching and the moment of storing all ones.

The operation is terminated on any protection violation. The condition-code setting is unpredictable when a protection violation occurs.

Resulting Condition Code:

0 Leftmost bit of byte specified is zero
1 Leftmost bit of byte specified is one
2 ---
3 ---

Program Interruptions:

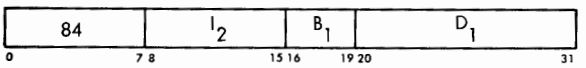
Protection (store or fetch violation)
Addressing

Programming Note

TEST AND SET can be used for controlled sharing of a common storage area by more than one program. To accomplish this, bit position 0 of a byte must be designated as the control bit. The desired interlock can be achieved by establishing a program convention in which a zero in the bit position indicates that the common area is available but a one means that the area is being used. Each using program then must examine this byte by means of TEST AND SET before making access to the common area. If the test sets the condition code to zero, the area is available for use; if it sets the condition code to one, the area cannot be used. Because TEST AND SET permits no access to the test byte between the moment of fetching (for testing) and the moment of storing all ones (setting), the possibility is eliminated of a second program's testing the byte before the first program is able to reset it.

Write Direct

WRD $D_1(B_1), I_2$ [*SI*]



The byte at the location designated by the operand address is made available as a set of direct-out static signals. Eight instruction bits are made available as signal-out timing signals.

The eight data bits of the byte fetched from storage are presented on a set of eight direct-out lines as static signals. These signals remain until the next WRITE DIRECT is executed. No parity is presented with the eight data bits.

Instruction bits 8-15, the I_2 field, are made available simultaneously on a set of eight signal-out lines as 0.5-microsecond to 1.0-microsecond timing signals. On a ninth line (write out) a 0.5-microsecond to 1.0-microsecond timing signal is made available concurrently with these timing signals. The eight signal-out lines are also used in READ DIRECT. No parity is made available with the eight instruction bits.

Condition Code: The code remains unchanged.

Program Interruptions:

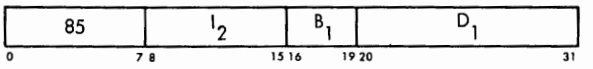
Operation (if direct control feature is not installed)
Privileged operation
Protection (fetch violation)
Addressing

Programming Note

The timing signals and the write-out signal may be used to alert the equipment to which the data are sent. When data are sent to another CPU, the external signal interruption may be used to alert that CPU.

Read Direct

RDD $D_1(B_1), I_2$ [*SI*]



Eight instruction bits are made available as signal-out timing signals. A direct-in data byte is accepted from an external device in the absence of a hold signal and is placed in the location designated by the operand address.

Instruction bits 8-15, the I_2 field, are made available on a set of eight signal-out lines as 0.5-microsecond to 1.0-microsecond timing signals. These signal-out lines are also used in WRITE DIRECT. On a ninth line (Read Out) a 0.5-microsecond to 1.0-microsecond timing signal is made available coincident with these timing signals. The read-out line is distinct from the write-out line in WRITE DIRECT. No parity is made available with the eight instruction bits.

Eight data bits are accepted from a set of eight direct-in lines when the hold signal on the hold-in line is absent. The hold signal is sampled after the read-out signal has been completed and should be absent for at least 0.5-microsecond. No parity is accepted with data signals, but a parity bit is generated as the data are placed in storage. When the hold signal is not removed, the CPU does not complete the instruction. Excessive duration of this instruction may result in incomplete updating of the timer.

Condition Code: The code remains unchanged.

Program Interruptions:

Operation (if direct control feature is not installed)
Privileged operation
Protection (store violation)
Addressing

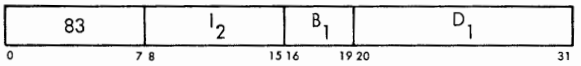
Programming Note

The direct-out lines of one CPU may be connected to the direct-in lines of another CPU, providing CPU-to-CPU static signaling. Further, the write-out signal of the sending CPU may serve as the hold signal for the receiving CPU, temporarily inhibiting a READ DIRECT when the signals are in transition.

Equipment connected to the hold-in line should be so constructed that the hold signal is removed when READ DIRECT is performed. Absence of the hold signal should correspond to absence of current in such a fashion that the CPU can proceed when power is removed from the source of the hold signal.

Diagnose

SI



The CPU performs built-in diagnostic functions.

The purpose of the I_2 field and the operand address may be defined in greater detail for a particular CPU and its appropriate diagnostic procedures. Similarly, the number of low-order address bits which must be zero is further specified for a particular CPU. When the address does not have the required number of low-order zeros, a specification exception causes a program interruption. Whether protection applies to DIAGNOSE depends on the model.

The purpose of the diagnostic functions is verification of proper functioning of the CPU equipment and locating faulty components.

The DIAGNOSE is completed either by taking the next sequential instruction or by obtaining a new PSW from location 112. The diagnostic procedure may affect the problem, supervisor, and interruptible status of the CPU, the condition code, and the contents of storage, registers, and timer, as well as the progress of I/O operations.

Some diagnostic functions turn on the test light on the operator control section of the system control panel.

Since the instruction is not intended for problem-program or supervisor-program use, DIAGNOSE has no mnemonic.

Condition Code: The code is unpredictable.

Program Interruptions:

Privileged operation
Protection (store or fetch violation)
Specification
Addressing

Status-Switching Exceptions

Exceptional instructions, operand designations, or data cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code inserted in the old PSW identifies the cause of the interruption. The following exception conditions cause a program interruption in status-switching operations.

Operation: The direct control feature is not installed, and the instruction is READ DIRECT OR WRITE DIRECT; or, the protection feature is not installed and the instruction is SET STORAGE KEY OR INSERT STORAGE KEY.

Privileged Operation: A LOAD PSW, SET SYSTEM MASK, SET STORAGE KEY, INSERT STORAGE KEY, WRITE DIRECT, READ DIRECT, or DIAGNOSE is encountered while the CPU is in the problem state.

Protection: The key of an operand in storage does not match the protection key in the PSW. The instruction is suppressed on a store violation, except for READ DIRECT and TEST AND SET, which are terminated. The operation is terminated on a fetch violation.

Addressing: An address designates a location outside the available storage for the installation. The operation is terminated, except for DIAGNOSE, which is suppressed.

Specification: The operand address of a LOAD PSW does not have all three low-order bits zero; the operand address of DIAGNOSE does not have as many low-order zero bits as required for the particular CPU; the block address specified by SET STORAGE KEY or INSERT STORAGE KEY does not have the four low-order bits all-zero; or the protection feature is not installed and a PSW with a nonzero protection key is introduced.

When an instruction is suppressed, storage and external signals remain unchanged, and the PSW is not changed by information from storage. Although storage remains unchanged, READ DIRECT may have made a timing signal available.

When an interruption is taken, the instruction address stored as part of the old PSW has been updated by the number of halfwords indicated by the instruction-length code in the old PSW.

Operand addresses are tested only when used to address storage. The address restrictions do not apply to the components from which an address is generated: the content of the D_1 field and the content of the register specified by B_1.

Programming Notes

When a program interruption occurs, the current PSW is stored in the old PSW location. The instruction address stored as part of this old PSW is thus the updated instruction address, having been updated by the number of halfwords indicated in the instruction-length code of the same PSW. The interruption code in this old PSW identifies the cause of the interruption and aids in the programmed interpretation of the old PSW.

If the new PSW for a program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established which may be broken only by an external or I/O interruption. If these interruptions also have an unacceptable new PSW, new supervisor information must be introduced by initial program loading or by manual intervention.

The interruption system permits the CPU to change its state as a result of conditions external to the system, in I/O units, or in the CPU itself. The five classes of these conditions are input/output, program, supervisor-call, external, and machine-check interruptions.

Interruption Action

An interruption consists of storing the current PSW as an old PSW and fetching a new PSW.

Processing resumes in the state indicated by the new PSW. The new PSW is not checked for programming errors when it becomes the current PSW. These checks are made when the next instruction is executed. The old PSW contains the address of the instruction that would have been executed next if an interruption had not occurred and the instruction-length code of the last-interpreted instruction.

Interruptions are taken only when the CPU is interruptible for the interruption source. Input/output and external interruptions may be masked by the system mask, four of the 15 program interruptions may be masked by the program mask, and the machine-check interruptions may be masked by the machine-check mask.

An interruption always takes place after one instruction interpretation is finished and before a new instruction interpretation is started. However, the occurrence of an interruption may affect the execution of the current instruction. To permit proper programmed action following an interruption, the cause of the interruption is identified and provision is made to locate the last-interpreted instruction.

When the CPU is commanded to stop, the current instruction is finished, and all interruptions that are pending or become pending before the end of the instruction, and which are not masked, are taken.

The details of instruction execution, source identification, and location determination are explained in later sections and are summarized in the following table.

Programming Note

A pending interruption will be taken even if the CPU becomes interruptible during only one instruction.

SOURCE IDENTIFICATION	INTERRUPTION CODE PSW BITS 16-31	MASK BITS	ILC SET	OPERATION EXECUTION
Input/Output (old PSW 56, new PSW 120, priority 4)				
Channel 0	00000000 aaaaaaaa	0	x	completed
Channel 1	00000001 aaaaaaaa	1	x	completed
Channel 2	00000010 aaaaaaaa	2	x	completed
Channel 3	00000011 aaaaaaaa	3	x	completed
Channel 4	00000100 aaaaaaaa	4	x	completed
Channel 5	00000101 aaaaaaaa	5	x	completed
Channel 6	00000110 aaaaaaaa	6	x	completed
Program (old PSW 40, new PSW 104, priority 2)				
Operation	00000000 00000001		1,2,3	suppressed
Privileged operation	00000000 00000010		1,2	suppressed
Execute	00000000 00000011		2	suppressed
Protection	00000000 00000100		0,2,3	suppressed or terminated
Addressing	00000000 00000101		0,1,2,3	suppressed or terminated
Specification	00000000 00000110		1,2,3	suppressed
Data	00000000 00000111		2,3	terminated
Fixed-point overflow	00000000 00001000	36	1,2	completed
Fixed-point divide	00000000 00001001		1,2	suppressed or completed
Decimal overflow	00000000 00001010	37	3	completed
Decimal divide	00000000 00001011		3	suppressed
Exponent overflow	00000000 00001100		1,2	completed
Exponent underflow	00000000 00001101	38	1,2	completed
Significance	00000000 00001110	39	1,2	completed
Floating-point divide	00000000 00001111		1,2	suppressed
Supervisor Call (old PSW 32, new PSW 96, priority 2)				
Instruction bits	00000000 rrrrrrrr		1	completed
External (old PSW 24, new PSW 88, priority 3)				
Timer	00000000 1nnnnnnn	7	x	completed
Interrupt key	00000000 n1nnnnnn	7	x	completed
External signal 2	00000000 nn1nnnnn	7	x	completed
External signal 3	00000000 nnn1nnnn	7	x	completed
External signal 4	00000000 nnnn1nnn	7	x	completed
External signal 5	00000000 nnnnn1nn	7	x	completed
External signal 6	00000000 nnnnnn1n	7	x	completed
External signal 7	00000000 nnnnnnn1	7	x	completed
Machine Check (old PSW 48, new PSW 112, priority 1)				
Machine malfunction	cccccccc cccccccc	13	x	terminated

NOTES

- a Device address bits
- c Bits of model-dependent code
- n Other external-interruption conditions
- r Bits of R_1 and R_2 field of SUPERVISOR CALL
- x Unpredictable

Instruction Execution

An interruption occurs when the preceding instruction is finished and the next instruction is not yet started. The manner in which the preceding instruction is finished may be influenced by the cause of the interruption. The instruction is said to have been completed, terminated, or suppressed.

In the case of instruction completion, results are stored and the condition code is set as for normal instruction operation, although the result may be influenced by the exception which has occurred.

In the case of instruction termination, all, part, or none of the result may be stored. Therefore, the result data are unpredictable. The setting of the condition code, if called for, may also be unpredictable. In general, the results should not be used for further computation.

In the case of instruction suppression, the execution proceeds as if no operation were specified. Results are not stored, and the condition code is not changed.

Source Identification

The five classes of interruptions are distinguished by the storage locations in which the old PSW is stored and from which the new PSW is fetched. The detailed causes are further distinguished by the interruption code of the old PSW, except for the machine-check interruption. The bits of the interruption code are numbered 16-31, according to their position in the PSW.

For I/O interruptions, additional information is provided by the contents of the channel status word stored as part of the I/O interruption.

For machine-check interruptions, additional information is provided by the diagnostic procedure, which is part of the interruption.

The following table lists the permanently allocated main-storage locations.

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 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 on the particular model and I/O channels.

Location Determination

For some interruptions, it is desirable to locate the instruction being interpreted when the interruption occurred. Since the instruction address in the old PSW designates the instruction to be executed next, it is necessary to know the length of the preceding instruction. This length is recorded in bit positions 32 and 33 of the PSW as the instruction-length code.

The instruction-length code is predictable only for program and supervisor-call interruptions. For I/O and external interruptions, the interruption is not caused by the last-interpreted instruction, and the code is not predictable for these instructions. For machine-check interruptions, the setting of the code may be affected by the malfunction and, therefore, is unpredictable.

For the supervisor-call interruption, the instruction-length code is 1, indicating the halfword length of SUPERVISOR CALL. For program interruptions, the codes 1, 2, and 3 indicate the instruction length in halfwords. The code 0 is reserved for program interruptions where the length of the instruction is not available because of certain overlapping conditions in instruction fetching. In code-0 cases, the instruction address in the old PSW does not represent the next instruction address. Instruction-length code 0 can occur for a program interruption only when the interruption is caused by a protected or an unavailable data address. The following table shows the states of the instruction-length code.

ILC	PSW BITS 32-33	INSTRUCTION BITS 0-1	INSTRUCTION LENGTH	FORMAT
0	00		Not available	
1	01	00	One halfword	RR
2	10	01	Two halfwords	RX
2	10	10	Two halfwords	RS or SI
3	11	11	Three halfwords	SS

Programming Notes

When a program interruption is due to an incorrect branch address, the location determined from the instruction address and instruction-length code is the branch address and not the location of the branch instruction.

When an interruption occurs while the CPU is in the wait state, the instruction-length code is always unpredictable.

The instruction EXECUTE represents upon interruption an instruction-length code which does not reflect the length of the instruction executed, but is 2, the length of EXECUTE.

Input/Output Interruption

The I/O interruption provides a means by which the CPU responds to signals from I/O devices.

A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests are preserved in the I/O section until accepted by the CPU. Priority is established among requests so that only one interruption request is processed at a time.

An I/O interruption can occur only after execution of the current instruction is completed and while the CPU is interruptible for the channel presenting the request. Channels are masked by system mask bits 0-6. Interruptions masked off remain pending.

The I/O interruption causes the old PSW to be stored at location 56 and causes the channel status word associated with the interruption to be stored at location 64. Subsequently, a new PSW is loaded from location 120.

The interruption code in the old PSW identifies the channel and device causing the interruption. The instruction-length code is unpredictable.

Program Interruption

Exceptions resulting from improper specification or use of instructions and data cause a program interruption.

The current instruction is completed, terminated, or suppressed. Only one program interruption occurs for a given instruction and is identified in the old PSW. The occurrence of a program interruption does not preclude the simultaneous occurrence of other program-interruption causes. Which of several causes is identified may vary from one occasion to the next and from one model to another.

A program interruption can occur only when the corresponding mask bit, if any, is one. When the mask bit is zero, the interruption is ignored. Program interruptions do not remain pending. Program mask bits 36-39 permit masking of four of the 15 interruption causes.

The program interruption causes the old PSW to be stored at location 40 and a new PSW to be fetched from location 104.

The cause of the interruption is identified by the four low-order bit positions in the interruption code, PSW bits 28-31. The remainder of the interruption code, bits 16-27 of the PSW, are made zero. The instruction-length code indicates the length of the preceding instruction in halfwords. For a few cases, the instruction length is not available. These cases are indicated by code 0.

If the new PSW for a program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established which may be broken only by an external or I/O interruption. If these interruptions also have an unacceptable new PSW, new supervisor information must be introduced by initial program loading or by manual intervention.

A description of the individual program exceptions follows. The application of these rules to each class of instructions is further described in the applicable sections. Some of the exceptions listed may also occur in operations executed by I/O channels. In that event, the exception is indicated in the channel status word stored with the I/O interruption (as explained under "Input/Output Operations").

Operation Exception

When an operation code is not assigned or the assigned operation is not available on the particular model, an operation exception is recognized. The operation is suppressed.

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

Privileged-Operation Exception

When a privileged instruction is encountered in the problem state, a privileged-operation exception is recognized. The operation is suppressed.

The instruction-length code is 1 or 2.

Execute Exception

When the subject instruction of EXECUTE is another EXECUTE, an execute exception is recognized. The operation is suppressed.

The instruction-length code is 2.

Protection Exception

When the key of an instruction halfword or an operand in storage does not match the protection key in the PSW, a protection exception is recognized.

The operation is suppressed on a store violation, except in the case of STORE MULTIPLE, READ DIRECT, TEST AND SET, and variable-length operations, which are terminated.

Except for EXECUTE, which is suppressed, the operation is terminated on a fetch violation.

The instruction-length code is 0, 2, or 3.

Addressing Exception

When an address specifies any part of data, an instruction, or a control word outside the available storage for the particular installation, an addressing exception is recognized.

In most cases, the operation is terminated for an invalid data address. Data in storage remain unchanged, except when designated by valid addresses. In a few cases, an invalid data address causes the instruction to be suppressed — AND (NI), EXCLUSIVE OR (XI), OR

(OI), MOVE (MVI), CONVERT TO DECIMAL, DIAGNOSE, EXECUTE, and certain store operations (ST, STC, STH, STD, and STE). The operation is suppressed for an invalid instruction address.

The instruction-length code normally is 1, 2 or 3; but may be 0 in the case of a data address.

Specification Exception

A specification exception is recognized when:

1. A data, instruction, or control-word address does not specify an integral boundary for the unit of information.
2. The R_1 field of an instruction specifies an odd register address for a pair of general registers that contains a 64-bit operand.
3. A floating-point register address other than 0, 2, 4, or 6 is specified.
4. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.
5. The first operand field is shorter than or equal to the second operand field in decimal multiplication or division.
6. The block address specified in SET STORAGE KEY or INSERT STORAGE KEY has the four low-order bits not all zero.
7. A PSW with a nonzero protection key is encountered when protection is not installed.

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

Data Exception

A data exception is recognized when:

1. The sign or digit codes of operands in decimal arithmetic or editing operations or in CONVERT TO BINARY are incorrect.
2. Fields in decimal arithmetic overlap incorrectly.
3. The decimal multiplicand has too many high-order significant digits.

The operation is terminated. The instruction-length code is 2 or 3.

Fixed-Point-Overflow Exception

When a high-order carry occurs or high-order significant bits are lost in fixed-point add, subtract, shift, or sign-control operations, a fixed-point-overflow exception is recognized.

The operation is completed by ignoring the information placed outside the register. The interruption may be masked by PSW bit 36.

The instruction-length code is 1 or 2.

Fixed-Point-Divide Exception

A fixed-point-divide exception is recognized when a quotient exceeds the register size in fixed-point division, including division by zero, or the result of CONVERT TO BINARY exceeds 31 bits.

Division is suppressed. Conversion is completed by ignoring the information placed outside the register.

The instruction-length code is 1 or 2.

Decimal-Overflow Exception

When the destination field is too small to contain the result field in a decimal operation, a decimal-overflow exception is recognized.

The operation is completed by ignoring the overflow information. The interruption may be masked by PSW bit 37.

The instruction-length code is 3.

Decimal-Divide Exception

When a quotient exceeds the specified data field size, a decimal-divide exception is recognized. The operation is suppressed.

The instruction-length code is 3.

Exponent-Overflow Exception

When the result characteristic in floating-point addition, subtraction, multiplication, or division exceeds 127 and the result fraction is not zero, an exponent-overflow exception is recognized. The operation is completed. The fraction is normalized, and the sign and fraction of the result remain correct. The result characteristic is made 128 smaller than the correct characteristic.

The instruction-length code is 1 or 2.

Exponent-Underflow Exception

When the result characteristic in floating-point addition, subtraction, multiplication, halving, or division is less than zero and the result fraction is not zero, an exponent-underflow exception is recognized. The operation is completed.

The setting of the exponent-underflow mask (PSW bit 38) affects the results of the operation. When the mask bit is zero, the sign, characteristic, and fraction are set to zero, making the result a true zero. When the mask bit is one, the fraction is normalized, the characteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct.

The instruction-length code is 1 or 2.

Significance Exception

When the result of a floating-point addition or subtraction has an all-zero fraction, a significance exception is recognized.

The operation is completed. The interruption may be masked by PSW bit 39. The manner in which the operation is completed is determined by the mask bit.

The instruction-length code is 1 or 2.

Floating-Point-Divide Exception

When division by a floating-point number with zero fraction is attempted, a floating-point divide exception is recognized. The operation is suppressed.

The instruction-length code is 1 or 2.

Supervisor-Call Interruption

The supervisor-call interruption occurs as a result of the execution of SUPERVISOR CALL.

The supervisor-call interruption causes the old PSW

to be stored at location 32 and a new PSW to be fetched from location 96.

The contents of bit positions 8-15 of the SUPERVISOR CALL become bits 24-31 in the interruption code of the old PSW. PSW bit positions 16-23 in the old PSW are made zero. The instruction-length code is 1, indicating the halfword length of SUPERVISOR CALL.

Programming Notes

The name "supervisor call" indicates that one of the major purposes of the interruption is the switching from problem to supervisor state. This major purpose does not preclude the use of this interruption for other types of status switching.

The interruption code may be used to convey a message from the calling program to the supervisor.

When SUPERVISOR CALL is performed as the subject instruction of EXECUTE, the instruction-length code is 2.

External Interruption

The external interruption provides a means by which the CPU responds to signals from the timer, from the interrupt key, and from external units.

A request for an external interruption may occur at any time, and requests from different sources may occur at the same time. Requests are preserved until honored by the CPU. All pending requests are presented simultaneously when an external interruption occurs. Each request is presented only once. When several requests from one source are made before the interruption is taken, only one interruption occurs.

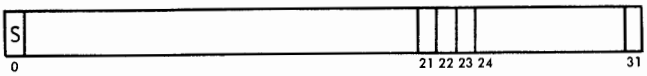
An external interruption can occur only when system mask bit 7 is one and after execution of the current instruction is completed. The interruption causes the old PSW to be stored at location 24 and a new PSW to be fetched from location 88.

The source of the interruption is identified in bit positions 24-31 of the old PSW. The remainder of the interruption code, PSW bits 16-23, is made zero. The instruction-length code is unpredictable for external interruptions.

Timer

A timer value changing from positive to negative causes an external interruption with PSW bit 24 set to one.

The timer occupies a 32-bit word at main-storage location 80 with a format as shown in the following illustration. The count is treated as a signed integer by following the rules for fixed-point arithmetic. The negative overflow, occurring as the timer value changes from a large negative number to a large positive number, is ignored. The interruption is initiated as the count proceeds from a positive number, including zero, to a negative number. The full cycle of the timer is 15.5 hours.



In the basic form, the contents of the timer are reduced by one in bit position 21 and in bit position 23 every 1/60 of a second, or the timer contents are reduced by one in bit position 21 and in bit position 22 every 1/50 of a second. The choice is determined by the available line frequency. In either case, the timer operates as if bit position 23 were being decremented by one every 1/300 of a second.

The line-frequency timer causes the word at storage location 80 to be updated whenever access to storage permits. The updating occurs only between the execution of instructions. An updated timer value is normally available at the end of each instruction execution, and, if no other activity in the system interferes with timer updating, any reference to the word at location 80 yields a timer value that, within the resolution of the timer, is not off by more than the execution time of the instruction. When the execution of an instruction or other activity in the system causes storage updating to be delayed by more than one period, the content of the word at location 80 subsequently may be reduced by more than one unit in a single updating cycle, depending on the length of the delay and the extent of timer backup storage. Timer updating may be omitted when I/O data transmission approaches the limit of storage capability, when a channel sharing CPU equipment and operating in burst mode causes CPU activity to be locked out, or when the instruction time for READ DIRECT is excessive. The program is not alerted when omission of updating causes the real-time count to be lost.

The value of the line-frequency timer is accessible to any reference to word location 80, provided the location is not protected for the type of reference. The 32-bit timer value may be changed at any time by storing a new value in location 80. Fetching of location 80 provides the current contents of bit positions 0-31. Bit positions 24-31 of the line-frequency timer do not participate in the updating, and their contents are not changed by the timer.

Higher resolution of timing may be obtained in some models by decrementing the content of bit position 31 approximately every 13 microseconds. The exact frequency of decrementing bit position 31 is such as to count at 300 cycles per second in bit position 23.

To avoid excessive interference in main storage when a timer with higher resolution is provided, all or part of the eight-bit high-resolution portion of the timer may be kept in internal timer backup storage.

Location 80 may not be updated for each time increment, but, storage accesses permitting, updating occurs at least as frequently as with the line-frequency timer. As in the case of the line-frequency timer, when activity in the system has caused updating of location 80 to be delayed, two or more contiguous updating cycles may occur, or, upon excessive activity, updating may be omitted, thus causing the real-time count to be lost.

When a CPU equipped with a high-resolution timer addresses the timer as a source of an operand, the instruction refers to both the main storage and backup storage portions of the timer, provided the location is not protected for the type of reference. All 32 bits of the timer may be changed by storing a new value at location 80, while on fetching an operand, location 80 yields a timer value that, within the resolution of the timer, is not off by more than the execution time of the instruction. When the content of the timer is fetched by the channel or is used as a source of an instruction, the content of the low-order byte is unpredictable. Similarly, when the channel stores data at location 83, the effect of the operation on the eight low-order bits of the timer value is unpredictable.

When location 80 is protected, any attempt to change the value of the timer causes a program interruption with protection exception. When protection exception is indicated, the timer value in main storage, as well as in the CPU internal backup storage, remains unchanged.

In a system having shared main storage, a CPU has access only to the main-storage part of another CPU's timer, provided the location is not protected for the type of reference. On fetching of the timer of another CPU, the content of the low-order byte is unpredictable when a high-resolution timer has been installed, whereas storing in the low-order byte of the timer of another CPU may have an unpredictable effect on the eight low-order bits of the timer value.

The timer value remains unchanged during initial program loading when the CPU is in the stopped state, or when the rate switch on the operator intervention panel is set to INSTRUCTION STEP.

The timer is an optional feature on some models.

Programming Notes

The timer in association with a program can serve both as a real-time clock and as an interval timer.

The value of the timer may be changed without losing the real-time count by loading the new value in byte locations 84-87 and then shifting bytes 80-87 into byte locations 76-83 by means of the instruction MOVE, thus placing in a single operation the new timer value into word location 80 and making the old value available at location 76. If two instructions are used to interrogate and then replace the value of the timer, the program may lose a time increment if an updating cycle occurs between the execution of the two instructions.

When the value of the timer is to be recorded on an I/O device, the program should first store the timer value in a temporary storage location to which the I/O operation should subsequently refer. When the channel fetches the timer value directly from location 80, the value as recorded on the I/O device is unpredictable. The channel may fetch the information from storage a byte or a halfword at a time, and the CPU may update the contents of location 80 between channel references.

Interrupt Key

Pressing the interrupt key on the operator control section of the system control panel causes an external interruption with PSW bit 25 set to one.

The key is active while power is on.

External Signal

An external signal causes an external interruption, with the corresponding bit in the interruption code turned on.

A total of six signal-in lines may be connected to the CPU for receiving external signals. The pattern presented in PSW bit positions 26-31 depends upon the pattern received before the interruption is taken. Because of circuit skew, one or more of several simultaneously generated external signals may arrive at the CPU too late to be included in the external interruption resulting from the earliest signal. These late signals may cause another interruption to be taken. An interruption will clear all signals concurrently if the CPU was disabled for external interruptions at the time the signals arrived.

The facility to accept external signals is part of the direct control feature. It may also be available as a separate feature.

Programming Note

The signal-in lines of one CPU may be connected to the signal-out timing lines of the direct control feature of another CPU, or the signal-in lines may be connected to the machine check-out lines of other CPU's. An interconnection of this kind allows one CPU to interrupt another. Also, the direct-out lines of one CPU may be connected to the direct-in lines of the other, and vice versa.

Machine-Check Interruption

The machine-check interruption provides a means for recovery from and fault location of machine malfunction.

When the machine-check mask bit is one, occurrence of a machine check terminates the current instruction, initiates an internal diagnostic procedure, issues a signal on the machine check-out line, and subsequently causes the machine-check interruption.

The old PSW is stored at location 48 and, depending on the model, the interruption code may identify the type of malfunction. The state of the CPU is scanned out into the storage area, starting with location 128 and extending through as many words as the given CPU requires. The new PSW is fetched from location 112. Proper execution of these steps depends on the nature of the machine check.

The machine check-out signal is provided as part of the multisystem feature. The signal is a 0.5-microsecond to 1.0-microsecond timing signal that follows the I/O interface line-driving and terminating specifications. The signal is designed so that it has a high probability of being issued in the presence of machine malfunction.

When the machine-check mask bit is zero, an attempt is made to complete the current instruction upon the occurrence of a machine check and to proceed with the next sequential instruction. No diagnostic procedure, signal, or interruption occurs.

A change in the machine-check mask bit due to the loading of a new PSW results in a change in the treatment of machine checks. Depending on the nature of a machine check, the earlier treatment may still be in force for several cycles.

Following emergency power turn-off and turn-on or system reset, incorrect parity may exist in storage or registers. Unless new information is loaded, a machine check may occur erroneously. Once storage and registers are cleared, a machine check can be caused only by machine malfunction and never by data or instructions.

Machine checks occurring in operations executed by I/O channels either cause a machine-check interruption or are recorded in the channel status word for that operation.

Priority of Interruptions

During execution of an instruction, several interruption-causing events may occur simultaneously. The instruction may give rise to a program interruption, an external interruption may occur, a machine check may occur, and an I/O interruption request may be made. Instead of the program interruption, a supervisor-call interruption might occur; however, both cannot occur since these two interruptions are mutually exclusive. Simultaneous interruption requests are honored in a predetermined order.

The machine-check interruption has highest priority. When it occurs, the current operation is terminated. Program and supervisor-call interruptions that would have occurred as a result of the current instruction are eliminated. Every reasonable attempt is made to limit the side-effects of a machine check. Normally, I/O and external interruptions, as well as the progress of the I/O data transfer and the updating of the timer, remain unaffected.

When no machine check occurs, the program interruption or supervisor-call interruption is taken first, the external interruption is taken next, and the I/O interruption is taken last. The action consists of storing the old PSW and fetching the new PSW belonging to the interruption first taken. This new PSW is subsequently stored without any instruction execution, and the next interruption PSW is fetched. This storing and fetching continues until no more interruptions are to be serviced. The external and I/O interruptions are taken only if the immediately preceding PSW indicates that the CPU is interruptible for these causes.

Instruction execution is resumed using the last-fetched PSW. The order of executing interruption subroutines is therefore the reverse of the order in which the PSW's are fetched.

The interruption code of a new PSW is not loaded since a new interruption code is always stored. The instruction-length code in a new PSW is similarly ignored since it is unpredictable for all interruptions other than program or supervisor call. The protection key of a new PSW is stored unchanged. When the feature is not installed, the protection key must be zero when loaded.

If the new PSW for the program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established. This string may be broken by an external or I/O interruption. If these interruptions also have an unacceptable new PSW, new supervisor information must be introduced by initial program loading or by manual intervention.

Programming Note

When interruption sources are not masked off, the order of priority in handling the interruption subroutines is machine check, I/O, external, and program or supervisor call. This order can be changed to some extent by masking. The priority rule applies to interruption requests made simultaneously. An interruption request made after some interruptions have already been taken is honored according to the priority prevailing at the moment of request.

Input/Output Operations

Transfer of information to and from main storage, other than to or from the central processing unit or via the direct control path, is referred to as input and output operation. An input/output (I/O) operation involves the use of an input/output device. Input/output devices perform I/O operations under control of control units, which are attached to the central processing unit (CPU) by means of channels.

This portion of the manual describes the programmed control of I/O devices by the channels and by the CPU. Formats are defined for the various types of I/O control information. The formats apply to all I/O operations, and are independent of the type of I/O device, its speed, or its mode of operation.

The formats described include provision for functions unique to an I/O device, such as erase gap on a magnetic tape unit. The way in which a device makes use of the format is defined in the SRL publication for the particular device.

NOTE: This part of the manual provides detailed information about the I/O system. A more general description appears in the Input and Output section of the System Structure part of this manual.

Attachment of Input/Output Devices

Input/Output Devices

Input/output devices provide external storage and a means of communication between data processing systems or between a system and its environment. Input/output devices include such equipment as card read punches, magnetic tape units, direct-access-storage devices (disks and drums), typewriter-keyboard devices, printers, teleprocessing devices, and process control equipment.

Most types of I/O devices, such as printers, card equipment, or tape devices, deal directly with external documents, and these devices are physically distinguishable and identifiable. Other types consist only of electronic equipment and do not directly handle physical recording media. The channel-to-channel adapter, for example, provides a channel-to-channel data transfer path, and the data never reach a physical recording medium outside main storage. Similarly, the IBM 2702 Transmission Control handles transmission of information between the data processing system and a remote station, and its input and output are signals on a transmission line. Furthermore, in this latter case, the 2702 may be time-shared for concurrent operation of a number of remote stations, and the 2702 is distinguished as a particular I/O device only during the time period associated with the operation on the corresponding remote station.

Input/output devices ordinarily are attached to one control unit and are accessible from one channel. Switching equipment is available to make some devices accessible to two or more channels by switching them between two or more control units. The time required for switching occurs during device selection time and may be ignored.

Control Units

The control unit provides the logical capabilities necessary to operate and control an I/O device, and adapts the characteristics of each device to the standard form of control provided by the channel.

All communication between the control unit and the channel takes place over the I/O interface. The control unit accepts control signals from the channel, controls the timing of data transfer over the I/O interface, and provides indications concerning the status of the device.

The I/O interface provides an information format and a signal sequence common to all I/O devices. The interface consists of a set of lines that can connect a number of control units to the channel. Except for the signal used to establish priority among control units, all communications to and from the channel occur over a common bus, and any signal provided by the channel is available to all control units. At any one instant, however, only one control unit is logically connected to the channel.

The selection of a control unit for communication with the channel is controlled by a signal from the channel that passes serially through all control units and permits, sequentially, each control unit to respond to additional signals provided by the channel. A control unit remains logically connected on the interface until it has transferred the information it needs or has, or until the channel signals it to disconnect, whichever occurs earlier.

The I/O device attached to the control unit may be designed to perform only certain limited operations, or it may perform many different operations. A typical operation is moving the recording medium and recording data. To accomplish these functions, the device needs detailed signal sequences peculiar to the type

of device. The control unit decodes the commands received from the channel, interprets them for the particular type of device, and provides the signal sequence required for execution of the operation.

A control unit may be housed separately or it may be physically and logically integral with the I/O device. In the case of most electromechanical devices, a well-defined interface exists between the device and the control unit because of the difference in the type of equipment the control unit and the device contain. These electromechanical devices often are of a type where only one device of a group attached to a control unit is required to operate at a time (magnetic tape units or disk access mechanisms, for example), and the control unit is shared among a number of I/O devices. On the other hand, in electronic I/O devices such as the channel-to-channel adapter, the control unit does not have an identity of its own.

From the user's point of view, most functions performed by the control unit can be merged with those performed by the I/O device. Therefore, this manual normally does not make specific mention of the control unit function; the execution of I/O operations is described as if the I/O devices communicated directly with the channel. Reference is made to the control unit only when emphasizing a function performed by it or when sharing of the control unit among a number of devices affects the execution of I/O operations.

Channels

The channel directs the flow of information between I/O devices and main storage. It relieves the CPU of the task of communicating directly with the devices and permits data processing to proceed concurrently with I/O operations.

The channel provides a standard interface for connecting different types of I/O devices to the CPU and to main storage. It accepts control information from the CPU in the format supplied by the program and changes it into a sequence of signals acceptable to a control unit. After the operation with the device has been initiated, the CPU is released for other work and the channel assembles or disassembles data and synchronizes the transfer of data bytes over the interface with main-storage cycles. To accomplish this, the channel maintains and updates an address and a count that describe the destination or source of data in main storage. Similarly, when an I/O device provides signals that should be brought to the attention of the program, the channel converts the signals to a format compatible to that used in the CPU.

The channel contains all the common facilities for the control of I/O operations. When these facilities are provided in the form of separate autonomous equipment designed specifically to control I/O devices, I/O operations are completely overlapped with the activity in the CPU. The only main-storage cycles required during I/O operations in such channels are those needed to transfer data and control information to or from the final locations in main storage. These cycles do not interfere with the CPU program, except when both the CPU and the channel concurrently attempt to refer to the same main storage.

Alternatively, the system may use to a greater or lesser extent the facilities of the CPU for controlling I/O devices. When the CPU and the channel share common facilities, channel operations cause interference to the CPU, varying in intensity from occasional delay of a CPU cycle through a complete lockout of CPU activity. The intensity depends on the extent of sharing and on the I/O data rate. The sharing of the facilities, however, is accomplished automatically, and the program is not affected by CPU delays, except for an increase in execution time.

Modes of Operation

Data transfer between main storage and an I/O device occurs in one of two modes: burst or multiplex.

In burst mode, the I/O device monopolizes the I/O interface and stays logically connected to the channel for the transfer of a burst of information. No other device can communicate over the interface during the time a burst is transferred. The burst can consist of a few bytes, a whole block of data, or a sequence of blocks with associated control and status information.

Some channels can tolerate an absence of data transfer during a burst mode operation, such as occurs when reading a long gap on tape, for not more than approximately one-half minute. Equipment malfunction may be indicated when an absence of data transfer exceeds this time.

In multiplex mode, the facilities in the channel may be shared by a number of concurrently operating I/O devices. The multiplex mode causes all I/O operations to be split into short intervals of time during which only a segment of information is transferred over the interface. During an interval, only one device is logically connected to the channel. The intervals associated with the concurrent operation of multiple I/O devices are sequenced in response to demands from the devices. The channel controls are occupied with any one operation only for the time required to transfer a segment of information. The segment can consist of a single byte of data, a few bytes of data, a status report from the device, or a control sequence used for initiation of a new operation.

A short burst of data can be handled in either multiplex or burst mode. The distinction between a short burst occurring in the multiplex mode and an operation in the burst mode is in the length of the bursts. A channel that can operate in either mode determines its mode of operation by "time-out." Whenever the burst causes the device to be connected to the channel for more than approximately 100 microseconds, the channel is considered to be operating in the burst mode.

Operation in burst and multiplex modes is differentiated because of the way the channels respond to I/O instructions. A channel operating a device in the burst mode appears busy to new I/O instructions, whereas a channel operating one or more devices in the multiplex mode is available for initiating operation of another device. If a channel that can operate in either mode happens to be communicating with an I/O device at the instant a new I/O instruction is issued, action on the instruction is delayed by the channel until the current mode of operation is established by time-out. A new I/O operation is initiated only after the channel has serviced all outstanding requests for data transfer from devices previously placed in operation.

Types of Channels

A system can be equipped with two types of channels: selector and multiplexor. Channels are classified according to the modes of operation they can sustain. A multiplexor channel can operate either in multiplex mode or in burst mode, depending upon the device. A selector channel operates only in burst mode.

The channel facilities required for sustaining a single I/O operation are termed a *subchannel.* The subchannel consists of the channel storage used for recording the addresses, count, and any status and control information associated with the I/O operation. The mode in which a channel can operate depends upon whether it has one or more subchannels.

The selector channel has one subchannel and always forces the I/O device to transfer data in the burst mode. The burst extends over the whole block of data, or, when command chaining is specified, over the whole sequence of blocks. The selector channel cannot perform any multiplexing and therefore can be involved in only one data transfer operation at a time. In the meantime, other I/O devices attached to the channel can be executing previously initiated operations that do not involve communication with the channel, such as rewinding tape to load point. When the selector channel is not executing an operation or a chain of operations and is not processing an interruption, it monitors the attached devices for status information.

The multiplexor channel contains multiple subchannels and can operate in either multiplex or burst mode. In multiplex mode, one or more devices may operate concurrently, each on a separate subchannel. In burst mode, only one device on the channel may be transferring data. The mode of operation is determined by the I/O device, and the mode can change at any time. The data transfer associated with an operation can occur partially in the multiplex mode and partially in the burst mode. Ordinarily, devices with a high rate of data transfer operate with the channel in burst mode, and slower devices run in multiplex mode. Some control units have a manual switch for setting the mode of operation.

Multiplexor channels vary in the number of subchannels they contain. When the multiplexor channel operates in multiplex mode, it can sustain concurrently one I/O operation per subchannel, provided that the total load on the channel does not exceed its capacity. Each subchannel appears to the program as an independent selector channel, except for those aspects of communication that pertain to the physical channel (e.g., individual subchannels on a multiplexor channel are not distinguished as such by the TEST CHANNEL instruction or by the mask controlling I/O interruptions from the channel). When the multiplexor channel is not servicing an I/O device, it monitors its devices for data and for interruption conditions.

When the multiplexor channel is transferring data in burst mode, the subchannel associated with the burst operation monopolizes the data transfer facilities of the channel. Other subchannels on the multiplexor channel cannot respond to requests from devices until the burst is completed.

Subchannels on the multiplexor channel may be either nonshared or shared.

A subchannel is referred to as nonshared if it is associated and can be used only with a single I/O device. A nonshared subchannel is used with devices that do not have any restrictions on the concurrency of data transfer, such as the IBM 1442 Card Read Punch Model N1 or the IBM 2250 Display Unit Model 1.

A subchannel is referred to as shared if data transfer to or from a set of devices implies the use of the same subchannel. Only one device associated with a shared subchannel may be involved in data transmission at a time. Shared subchannels are used with devices, such as magnetic tape units or disk access mechanisms, that share a control unit. For such devices, the sharing of the subchannel does not restrict the concurrency of I/O operations since the control unit permits only one device to be involved in a data-transfer operation at a

time. A subchannel is not necessarily shared, however, even when the I/O devices share a control unit. For example, each transmission line attached to the IBM 2702 Transmission Control is assigned a nonshared subchannel, although all of the transmission lines share the common control unit.

System Operation

Input/output operations are initiated and controlled by information with three types of formats: instructions, commands, and orders. Instructions are decoded by the CPU and are part of the CPU program. Commands are decoded and executed by the channels and I/O devices, and initiate I/O operations, such as reading and writing. One or more commands arranged for sequential execution form a channel program. Both instructions and commands are fetched from main storage and are common to all types of I/O devices, although the modifier bits in the command code may specify device-dependent conditions for the execution of a data-transfer operation at the device.

Functions peculiar to a device, such as rewinding tape or positioning the access mechanism on a disk drive, are specified by orders. Orders are decoded and executed by I/O devices. The control information specifying an order may appear in the modifier bits of a control command code, may be transferred to the device as data during a control or write operation, or may be made available to the device by other means.

The CPU program initiates I/O operations with the instruction START I/O. This instruction identifies the channel and device and causes the channel to fetch the channel address word (CAW) from a fixed location in main storage. The CAW contains the protection key and designates the location in main storage from which the channel subsequently fetches the first channel command word (CCW). The CCW specifies the command to be executed and the storage area, if any, to be used.

If the channel is not operating in burst mode and if the subchannel associated with the addressed I/O device is not busy, the channel attempts to select the device by sending the address of the device to all control units attached to the channel. A control unit that recognizes the address connects itself logically to the channel and responds to its selection by returning its address. The channel subsequently sends the command code part of the CCW over the interface, and the device responds with a status byte indicating whether it can execute the command.

At this time, the execution of START I/O is terminated. The results of the attempt to initiate the execution of the command are indicated by setting the condition code in the program status word (PSW), and, under certain conditions, by storing pertinent information in the channel status word (CSW).

If the operation is initiated at the device and its execution involves transfer of data, the subchannel is set up to respond to service requests from the device and assumes further control of the operation. In the case of operations that do not require any data to be transferred to or from the device, the device may signal the end of the operation immediately on receipt of the command code.

An I/O operation may involve transfer of data to one storage area, designated by a single CCW, or to a number of noncontiguous storage areas. In the latter case, a list of CCW's is used for execution of the I/O operation, each CCW designating a contiguous storage area, and the CCW's are said to be coupled by data chaining. Data chaining is specified by a flag in the CCW and causes the channel to fetch another CCW upon the exhaustion or filling of the storage area designated by the current CCW. The storage area designated by a CCW fetched on data chaining pertains to the I/O operation already in progress at the I/O device, and the I/O device is not notified when a new CCW is fetched. Provision is made in the CCW format for the programmer to specify that, when the CCW is decoded, the channel request an I/O interruption as soon as possible, thereby notifying the CPU program that chaining has progressed to a particular CCW in the channel program.

Termination of the I/O operation normally is indicated by two conditions: channel end and device end. The channel-end condition indicates that the I/O device has received or provided all information associated with the operation and no longer needs channel facilities. The device-end signal indicates that the I/O device has terminated execution of the operation. The device-end condition can occur concurrently with the channel-end condition or later.

Operations that keep the control unit busy after releasing channel facilities may, under certain conditions, cause a third type of signal. This signal, called control unit end, may occur only after channel end and indicates that the control unit has become available for initiation of another operation.

The conditions signaling the termination of an I/O operation can be brought to the attention of the program by I/O interruptions or, when the channel is masked, by programmed interrogation of the I/O device. In either case, these conditions cause storing the CSW, which contains additional information concerning the execution of the operation. At the time the channel-end condition is generated, the channel identifies to the program the last CCW used and provides its residual byte count, thus indicating the extent of main storage used. Both the channel and the device can

provide indications of unusual conditions with channel end. The control-unit-end and device-end conditions can be accompanied by error indications from the device.

Facilities are provided for the program to initiate execution of a chain of I/O operations with a single START I/O. When the chaining flags in the current CCW specify command chaining and no unusual conditions have been detected in the operation, the receipt of the device-end signal causes the channel to fetch a new CCW and to initiate a new command at the device. A chained command is initiated by means of the same sequence of signals over the I/O interface as the first command specified by START I/O. The ending signals occurring at the termination of an operation caused by a CCW specifying command chaining are not made available to the program when another operation is initiated by the command chaining; the channel continues execution of the channel program. If, however, an unusual condition has been detected, the ending signals cause suppression of command chaining and termination of the channel program.

Conditions that initiate I/O interruptions are asynchronous to activity in the CPU, and more than one condition can occur at the same time. The channel and the CPU establish priority among the conditions so that only one interruption request is processed at a time. The conditions are preserved in the I/O devices and subchannels until accepted by the CPU.

Execution of an I/O operation or chain of operations thus involves up to four levels of participation:

1. Except for the effects caused by the integration of CPU and channel equipment, the CPU is busy for the duration of execution of START I/O, which lasts at most until the addressed I/O device responds to the first command.

2. The subchannel is busy with the execution from the initiation of the operation at the I/O device until the channel-end condition for the last operation of the command chain is accepted by the CPU.

3. The control unit may remain busy after the subchannel has been released and may generate the control-unit-end condition when it becomes free.

4. The I/O device is busy from the initiation of the first command until the device-end condition associated with the last operation is accepted or cleared by the CPU.

A pending device-end condition causes the associated device to appear busy, but does not affect the state of any other part of the system. A pending control unit end blocks communications through the control unit to any device attached to it, and a pending channel end normally blocks all communications through the subchannel.

Compatibility of Operation

The organization of the I/O system provides for a uniform method of controlling I/O operations. The capability of a channel, however, depends on its use and on the model to which it belongs. Channels are provided with different data-transfer capabilities, and an I/O device designed to transfer data only at a specific rate (a magnetic tape unit or a disk storage, for example) can operate only on a channel that can accommodate at least this data rate.

The data rate a channel can accommodate depends also on the way the I/O operation is programmed. The channel can sustain its highest data rate when no data chaining is specified. Data chaining reduces the maximum allowable rate, and the extent of the reduction depends on the frequency at which new CCW's are fetched and on the address resolution of the first byte in the new main-storage area. Furthermore, since in most instances the channel may share main storage with the CPU and other channels, activity in the rest of the system affects the accessibility of main storage and, hence, the instantaneous load the channel can sustain.

In view of the dependence of channel capacity on programming and on activity in the rest of the system, an evaluation of the ability of a specific I/O configuration to function concurrently must be based on a consideration of both the data rate and the way the I/O operations are programmed. Two systems employing identical complements of I/O devices may be able to execute certain programs in common, but it is possible that other programs requiring, for example, data chaining, may not run on one of the systems because of the increased load caused by the data chaining.

Control of Input/Output Devices

The CPU controls I/O operations by means of four I/O instructions: START I/O, TEST I/O, HALT I/O, and TEST CHANNEL.

The instruction TEST CHANNEL addresses a channel; it does not address an I/O device. The other three I/O instructions address a channel and a device on that channel.

Input/Output Device Addressing

An I/O device and the associated access path are designated by an I/O address. The I/O address is a 16-bit binary number and consists of two parts: a channel address in the eight high-order bit positions and a device address in the eight low-order bit positions.

The channel-address field provides for identifying up to 256 channels, out of which only channels 0-6 may be installed; channel-addresses 7 and up are considered invalid. Channel 0 is a multiplexor channel;

channels numbered 1-6 may be either multiplexor or selector channels, as shown below. The number and type of channels available, as well as their address assignment, depend on the system model and the particular installation.

ADDRESS		
Channel	Device	ASSIGNMENT
0000 0000	XXXX XXXX	Devices on channel 0
0000 0001	XXXX XXXX	Devices on channel 1
0000 0010	XXXX XXXX	Devices on channel 2
0000 0011	XXXX XXXX	Devices on channel 3
0000 0100	XXXX XXXX	Devices on channel 4
0000 0101	XXXX XXXX	Devices on channel 5
0000 0110	XXXX XXXX	Devices on channel 6
0000 0111	XXXX XXXX	
TO		INVALID
1111 1111	XXXX XXXX	

The device address identifies the particular I/O device and control unit on the designated channel. The address identifies, for example, a particular magnetic tape drive, disk access mechanism, or transmission line. Any number in the range 0-255 can be used as a device address, providing facilities for addressing up to 256 devices per channel.

On the multiplexor channel, the device address identifies the subchannel as well as the I/O device and control unit. Addresses with a "zero" in the high-order bit position of the device-address field pertain to subchannels that are not shared. Each nonshared subchannel is identified by a unique device address. Addresses with a "one" in the high-order bit position may designate either a shared or a nonshared subchannel. Shared subchannels are assigned sets of contiguous addresses, the number of addresses within the set being a power of two, and the first address within the set being a multiple of the set size. When a multiplexor channel has both shared and nonshared subchannels in the address range 128-255 (high-order bit is one), the nonshared subchannels are assigned lower addresses than the shared subchannels. The number and the type of subchannels available on a particular multiplexor channel and their address assignment depend on the system model and the installation.

Devices that do not share a control unit with other devices may be assigned any device address in the range 0-255, provided the address is not recognized by any other control unit. Logically, such devices are not distinguishable from their control unit, and both are identified by the same address.

Devices sharing a control unit (i.e., magnetic tape drives or disk access mechanisms) are assigned addresses within sets of contiguous numbers. The size of such a set is equal to the maximum number of devices that can share the control unit, or 16, whichever is smaller. Furthermore, such a set starts with an address in which the number of low-order zeros is at least equal to the number of bit positions required for specifying the set size. The high-order bit positions of an address within such a set identify the control unit, and the low-order bit positions designate the device on the control unit.

Control units designed to accommodate more than 16 devices (i.e., IBM 2702 Transmission Control) may be assigned nonsequential sets of addresses, each set consisting of 16, or the number required to bring the total number of assigned addresses equal to the maximum number of devices attachable to the control unit, whichever is smaller. The addressing facilities are added in increments of a set so that the number of device addresses assigned to a control unit does not exceed the number of devices attached by more than 15.

The control unit does not respond to any address outside its assigned set or sets. For example, if a control unit is designed to control devices having only bits 0000-1001 in the low-order positions of the device address, it does not recognize addresses containing 1010-1111 in these bit positions. On the other hand, a control unit responds to all addresses in the assigned set, regardless of whether the device associated with the address is installed. For example, the IBM 2803 Tape Control with four tape units installed, and not equipped with the 16-drive addressing feature responds to all of the eight addresses within the set assigned to it. If no control unit responds to an address, the I/O device appears not operational. If a control unit responds to an address for which no device is installed, the absent device appears in the not-ready state.

Input/output devices accessible through more than one channel have a distinct address for each path of communications. This address identifies the channel, the subchannel, and the control unit. For sets of devices connected to two or more control units, the portion of the address identifying the device on the control unit is fixed, and does not depend on the path of communications.

Except for the rules described, the assignment of channel and device addresses is arbitrary. The assignment is made at the time of installation, and the addresses normally remain fixed thereafter.

States of the Input/Output System

The state of the I/O system identified by an I/O address depends on the collective state of the channel, subchannel, and I/O device. Each of these components of the I/O system can have up to four states, as far as the response to an I/O instruction is concerned. These states are listed in the following table. The name of the state is followed by its abbreviation and a brief definition.

CHANNEL	ABBREV	DEFINITION
Available	A	None of the following states
Interruption pending	I	Interruption immediately available from channel
Working	W	Channel operating in burst mode
Not operational	N	Channel not operational

SUBCHANNEL	ABBREV	DEFINITION
Available	A	None of the following states
Interruption pending	I	Information for CSW available in subchannel
Working	W	Subchannel executing an operation
Not operational	N	Subchannel not operational

I/O DEVICE	ABBREV	DEFINITION
Available	A	None of the following states
Interruption pending	I	Interruption condition pending in device
Working	W	Device executing an operation
Not operational	N	Device not operational

A channel, subchannel, or I/O device that is available, that contains a pending interruption condition, or that is working, is said to be operational. The states of containing an interruption condition, working, or being not operational are collectively referred to as "not available."

In the case of the multiplexor channel, the channel and subchannel are easily distinguishable and, if the channel is operational, any combination of channel and subchannel states are possible. Since the selector channel can have only one subchannel, the channel and subchannel are functionally coupled, and certain states of the channel are related to those of the subchannel. In particular, the working state can occur only concurrently in both the channel and subchannel and, whenever an interruption condition is pending in the subchannel, the channel also is in the same state. The channel and subchannel, however, are not synonymous, and an interruption condition not associated with data transfer, such as attention, does not affect the state of the subchannel. Thus, the subchannel may be available when the channel has an interruption condition pending. Consistent distinction between the subchannel and channel permits the two types of channels, selector and multiplexor, to be covered uniformly by a single description.

The device referred to in the preceding table includes both the device proper and its control unit. For some types of devices, such as magnetic tape units, the working and the interruption-pending states can be caused by activity in the addressed device or control unit. A shared control unit imposes its state on all devices attached to the control unit. The states of the devices are not related to those of the channel and subchannel.

When the response to an I/O instruction is determined on the basis of the states of the channel and subchannel, the components further removed are not interrogated. Thus, ten composite states are identified as conditions for the execution of the I/O instruction. Each composite state is identified in the following discussion by three alphabetic characters; the first character position identifies the state of the channel, the second identifies the state of the subchannel, and the third refers to the state of the device. Each character position can contain A, I ,W, or N, denoting the state of the component. The symbol x in place of a letter indicates that the state of the corresponding component is not significant for the execution of the instruction.

Available (AAA): The addressed channel, subchannel, control unit, and I/O device are operational, are not engaged in the execution of any previously initiated operations, and do not contain any pending interruption conditions.

Interruption Pending in Device (AAI) or Device Working (AAW): The addressed channel and subchannel are available. The addressed control unit or I/O device is executing a previously initiated operation or contains a pending interruption conditon. These situations are possible:

1. The device is executing an operation after signaling the channel-end condition, such as rewinding tape or seeking on a disk file.
2. The control unit associated with the device is executing an operation after signaling the channel-end condition, such as backspacing file on a magnetic tape unit.
3. The device or control unit is executing an operation on another subchannel or channel.
4. The device or control unit contains the device-end, control-unit-end, or attention condition or, on the selector channel, the channel-end condition associated with an operation terminated by HALT I/O.

Device Not Operational (AAN): The addressed channel and subchannel are available. The addressed I/O device is not operational. A device appears not operational when no control unit recognizes the address. This occurs when the control unit is not provided in the system, when power is off in the control unit, or when the control unit has been logically switched off the I/O interface. The not-operational state is indicated also when the control unit is provided and is designed to attach the device, but the device has not been installed and the address has not been assigned to the control unit (for example, the second set of lines on the IBM 2702 Transmission Control). See also "Input/Output Device Addressing."

If the addressed device is not installed or has been logically removed from the control unit, but the associated control unit is operational and the address has been assigned to the control unit (for example, access

mechanism 7 on the IBM 2841 Storage Control that has only access mechanism 0-3 installed) the device is said to be not-ready. When an instruction is addressed to a device in the not-ready state, the control unit responds to the selection and indicates unit check whenever the not-ready state precludes a successful execution of the operation. See "Unit Check."

Interruption Pending In Subchannel (AIX): The addressed channel is available. An interruption condition is pending in the addressed subchannel because of the termination of the portion of the operation involving the use of channel facilities. The subchannel is in a position to provide information for a complete CSW. The interruption condition can indicate termination of an operation at the addressed I/O device or at another device on the subchannel. The state of the addressed device is not significant, except when TEST I/O is addressed to the device associated with the terminated operation, in which case the CSW contains status information provided by the device.

The state AIX does not occur on the selector channel. On the selector channel, the existence of an interruption condition in the subchannel immediately causes the channel to assign to this condition the highest priority for I/O interruptions and, hence, leads to the state IIX.

Subchannel Working (AWX): The addressed channel is available. The addressed subchannel is executing a previously initiated operation or chain of operations in the multiplex mode and has not yet reached the channel end for the last operation. The state of the addressed device is not significant, except when HALT I/O is issued, in which case the CSW contains status provided by the device.

The subchannel-working state does not occur on the selector channel since all operations on the selector channel are executed in the burst mode and cause the channel to be in the working state(WWX).

Subchannel Not Operational (ANX): The addressed channel is available. The addressed subchannel on the multiplexor channel is not operational. A subchannel is not operational when it is not provided in the system. This state cannot occur on the selector channel.

Interruption Pending in Channel (IXX): The addressed channel is not working and has established which device will cause the next I/O interruption from this channel. The state where the channel contains a pending interruption condition is distinguished only by the instruction TEST CHANNEL. This instruction does not cause the subchannel and I/O device to be interrogated. The other I/O instructions consider the channel available when it contains a pending interruption condition. When the channel assigns priority for interruption among devices, the interruption condition is preserved in the I/O device or subchannel. (See "Interruption Conditions.")

Channel Working (WXX): The addressed channel is operating in the burst mode. In the case of the multiplexor channel, a burst of bytes is currently being handled. In the case of the selector channel, an operation or a chain of operations is currently being executed, and the channel end for the last operation has not yet been reached. The states of the addressed device and, in the case of the multiplexor channel, of the subchannel are not significant.

Channel Not Operational (NXX): The addressed channel is not operational, or the channel address in the instruction is invalid. A channel is not operational when it is not provided in the system, when power is off in the channel, or when it has been switched to the test mode. The states of the addressed I/O device and subchannel are not significant.

Resetting of the Input/Output System

Two types of resetting can occur in the I/O system. The reset states overlap the hierarchy of states distinguished for the purpose of responding to the CPU during the execution of I/O instructions. Resetting terminates the current operation, disconnects the device from the channel, and may place the device in certain modes of operation. The meaning of the two reset states for each type of I/O device is specified in the Systems Reference Library (SRL) publication for the device.

System Reset

The system-reset function is performed when the system-reset or load key is pushed, or when a system power-on sequence is completed.

System reset causes the channel to terminate operations on all subchannels. Status information and interruption conditions in the subchannels are reset, and all operational subchannels are placed in the available state. The channel sends the system-reset signal to all I/O devices attached to it.

If the device is currently communicating over the I/O interface, the device immediately disconnects from the channel. Data transfer and any operation using the facilities of the control unit are immediately terminated, and the I/O device is not necessarily positioned at the beginning of a block. Mechanical motion not involving the use of the control unit, such as rewinding magnetic tape or positioning a disk access mechanism, proceeds to the normal stopping point, if possible. The device appears in the working state until the termination of mechanical motion or the inherent cycle of operation, if any, whereupon it becomes

available. Status information in the device and control unit is reset, and no interruption condition is generated upon completing the operation.

A control unit accessible by more than one channel is reset if it is currently associated with a channel on the CPU generating the reset.

Malfunction Reset

The malfunction-reset function is performed when the channel detects equipment malfunctioning.

Execution of malfunction reset in the channel depends on the type of malfunction and the model. It may cause all operations in the channel to be terminated and all operational subchannels to be reset to the available state. The channel may send either the malfunction-reset signal to the device connected to the channel at the time the malfunctioning is detected, or channels sharing common equipment with the CPU may send the system-reset signal to all devices attached to the channel.

When the channel signals malfunction reset over the interface, the device immediately disconnects from the channel. Data transfer and any operation using the facilities of the control unit are immediately terminated, and the I/O device is not necessarily positioned at the beginning of a block. Mechanical motion not involving the control unit, such as rewinding magnetic tape or positioning a disk access mechanism, proceeds to the normal stopping point, if possible. The device appears in the working state until the termination of mechanical motion or the inherent cycle of operation, if any. Status information associated with the addressed device is reset, but an interruption condition may be generated upon completing any mechanical operation.

When a malfunction reset occurs, the program is alerted by an I/O interruption or, when the malfunction is detected during the execution of an I/O instruction, by the setting of the condition code. In either case the CSW identifies the condition. The device addressed by the I/O instruction or the device identified by the I/O interruption, however, is not necessarily the one placed in the malfunction-reset state. In channels sharing common equipment with the CPU, malfunctioning detected by the channel may be indicated by a machine-check interruption, in which case a CSW is not stored and a device is not identified. The method of identifying malfunctioning depends upon the model.

Condition Code

The results of certain tests by the channel and device, and the original state of the addressed part of the I/O system are used during the execution of an I/O instruction to set one of four condition codes in bit positions 34 and 35 of the PSW. The condition code is set at the time the execution of the instruction is completed, that is, the time the CPU is released to proceed with the next instruction. The condition code indicates whether or not the channel has performed the function specified by the instruction and, if not, the reason for the rejection. Immediately subsequent branch-on-condition operations can use the code for decision-making.

The following table lists the conditions that are identified and the corresponding condition codes for each instruction. The states of the system and their abbreviations were previously defined in "States of the Input/Output System." The digits in the table represent the numeric value of the code. The instruction START I/O can set code 0 or 1 for the AAA state, depending on the type of operation that is initiated. Equipment malfunctions and programming errors generally cause condition code 1 to be set and the CSW to be stored.

CONDITIONS		CONDITION CODE FOR START I/O	TEST I/O	HALT I/O	HALT TEST CHAN
Available	A A A	0,1°	0	1°	0
Interruption pend. in device	A A I	1°	1°	1°	0
Device working	A A W	1°	1°	1°	0
Device not operational	A A N	3	3	3	0
Interruption pend. in subchannel	A I X †				
For the addressed device		2	1°	0	0
For another device		2	2	0	0
Subchannel working	A W X †	2	2	1°#	0
Subchannel not operational	A N X †	3	3	3	0
Interruption pend. in channel	I X X †	see note below			1
Channel working	W X X †	2	2	2	2
Channel not operational	N X X †	3	3	3	3

°The CSW or its status portion is stored at location 64 during execution of the instruction.

†The symbol X stands for A, I, W, and N, and indicates that the state of the corresponding component is not significant. As an example, AIX denotes the states AIA, AII, AIW, and AIN, while IXX represents a total of 16 states, some of which do not occur.

#When a device-not-operational response is received in selecting the addressed device, a condition code of 3 is set.

—The condition cannot be identified during execution of the instruction.

NOTE: For the purpose of executing START I/O, TEST I/O, and HALT I/O, a channel containing a pending interruption condition appears the same as an available channel, and the condition-code setting depends upon the states of the subchannel and device. The condition codes for the IXX states are the same as for the AXX states, where the X's represent the states of the subchannel and the device. As an example, the condition-code for the IAA state is the same as for the AAA state, and the condition code for the IAW state is the same as for the AAW state.

The available condition is indicated only when no errors are detected during the execution of the I/O instruction. When a programming error occurs in the information placed in the CAW or CCW and the addressed channel or subchannel is working, either con-

dition code 1 or 2 may be set, depending upon the model. Similarly, either code 1 or 3 may be set when a programming error occurs and a part of the addressed I/O system is not operational.

When a subchannel on the multiplexor channel contains a pending interruption condition (state AIX), the I/O device associated with the terminated operation normally is in the interruption-pending state. When the channel detects during execution of TEST I/O that the device is not operational, condition code 3 is set. Similarly, condition code 3 is set when HALT I/O is addressed to a subchannel in the working state and operating in the multiplex mode (state AWX), but the device turns out to be not operational. The not-operational state in both situations can be caused by operator intervention or by machine malfunctioning.

The error conditions listed in the preceding table include all equipment or programming errors detected by the channel or the I/O device during execution of the I/O instruction. Except for channel equipment errors, in which case, depending on the model, machine check may be indicated and no CSW may be stored, the status portion of the CSW identifies the error. Three types of errors can occur:

Channel Equipment Error: The channel can detect the following equipment errors during execution of START I/O, TEST I/O, and HALT I/O:

1. The device address that the channel received on the interface during initial selection either has a parity error or is not the same as the one the channel sent out. Some device other than the one addressed may be malfunctioning.
2. The unit-status byte that the channel received on the interface during initial selection has a parity error.
3. A signal from the I/O device occurred during initial selection at an invalid time or had invalid duration.
4. The channel detected an error in its control equipment.

The channel may perform the malfunction-reset function, depending on the type of error and the model. If a CSW is stored, channel control check or interface control check is indicated, depending on the type of error.

Channel Programming Error: The channel can detect the following programming errors during execution of START I/O:

1. Invalid CCW address in CAW
2. Invalid CCW address specification in CAW
3. Invalid storage protection key in CAW
4. Invalid CAW format
5. Location of first CCW protected for fetching
6. First CCW specifies transfer in channel
7. Invalid command code in first CCW
8. Initial data address exceeds addressing capacity of model (see "Definition of Storage Area")
9. Invalid count in first CCW
10. Invalid format of first CCW

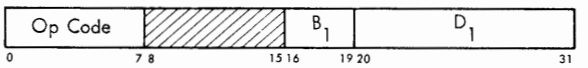
The CSW indicates program check, except for condition 5, in which case protection check is indicated.

Device Error: Programming or equipment errors detected by the device during the execution of START I/O are indicated by unit check or unit exception in the CSW.

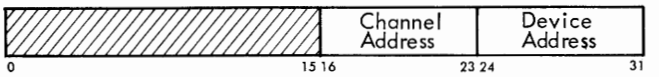
The conditions responsible for unit check and unit exception for each type of I/O device are detailed in the SRL publication for the device.

Instruction Format

All I/O instructions use the following SI format:



Bit positions 8-15 of the instruction are ignored. The content of the B_1 field designates a register. The sum obtained by the addition of the content of register B_1 and content of the D_1 field identifies the channel and the I/O device. This sum has the format:



Bit positions 0-7 are not part of the address. Bit positions 8-15, which constitute the high-order portion of the address, are ignored. Bit positions 16-23 of the sum contain the channel address, while bit positions 24-31 identify the device on the channel and, additionally in the case of the multiplexor channel, the subchannel.

NOTE: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/360 assembly language are shown with each instruction. In the case of START I/O, for example, SIO is the mnemonic and D_1 (B_1) the operand designation.

Instructions

The mnemonics, format, and operation codes of the I/O instructions follow. The table also indicates that all I/O instructions cause program interruption when they are encountered in the problem state, and that all I/O instructions set the condition code.

NAME	MNEMONIC	TYPE	EXCEPTION	CODE
Start I/O	SIO	SI, C	M	9C
Test I/O	TIO	SI, C	M	9D
Halt I/O	HIO	SI, C	M	9E
Test Channel	TCH	SI, C	M	9F

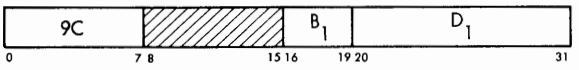
NOTES
C Condition code is set
M Privileged-operation exception

Programming Note

The instructions START I/O, TEST I/O, and HALT I/O may cause a CSW to be stored. To prevent the contents of the CSW stored by the instruction from being destroyed by an immediately following I/O interruption, all channels must be masked before issuing START I/O, TEST I/O, or HALT I/O and must remain masked until the information in the CSW provided by the instruction has been acted upon or stored elsewhere for later use.

Start I/O

SIO $D_1(B_1)$ [*S*]



A write, read, read backward, control or sense operation is initiated at the addressed I/O device and subchannel. The instruction START I/O is executed only when the CPU is in the supervisor state.

Bit positions 16-31 of the sum formed by the addition of the content of register B_1 and the content of the D_1 field identify the channel, subchannel, and I/O device to which the instruction applies. The CAW at location 72 contains the protection key for the subchannel and the address of the first CCW. The CCW so designated specifies the operation to be performed, the main-storage area to be used, and the action to be taken when the operation is completed.

The I/O operation specified by START I/O is initiated if the addressed I/O device and subchannel are available, the channel is available or is in the interruption-pending state, and errors or exceptional conditions have not been detected. The I/O operation is not initiated when the addressed part of the I/O system is in any other state or when the channel or device detects any error or exceptional condition during execution of the instruction.

When any of the following conditions occurs, with the channel either available or in the interruption-pending state and with the subchannel available before the execution of the instruction, START I/O causes the status portion, bit positions 32-47, of the CSW at location 64 to be replaced by a new set of status bits. The status bits pertain to the device addressed by the instruction. The contents of the other fields of the CSW are not changed.

1. An immediate operation was executed, and either no command chaining is specified, or chaining is suppressed because of unusual conditions detected during the operation. An operation is called *immediate* when the I/O device signals the channel-end condition immediately on receipt of the command code. The CSW contains the channel-end bit and any other indications provided by the channel or the device. The busy bit is off. The I/O operation has been initiated, but no information has been transferred to or from the storage area designated by the CCW. No interruption conditions are generated at the device or subchannel, and the subchannel is available for a new I/O operation.

2. The I/O device contains a pending interruption condition due to device end or attention, the control unit contains a pending control unit end for the addressed device, or, on the selector channel, the control unit contains for the addressed device a pending channel end following the execution of HALT I/O. The CSW unit-status field contains the busy bit, identifies the interruption condition, and may contain other bits provided by the device or control unit. The interruption condition is cleared. The channel-status field indicates any error conditions detected by the channel and contains the PCI bit if specified in the first CCW.

3. The I/O device or the control unit is executing a previously initiated operation, or the control unit has pending an interruption condition associated with a device other than the one addressed. The CSW unit-status field contains the busy bit or, if the control unit is busy, the busy and status-modifier bits. The channel-status field indicates any error conditions detected by the channel and contains the PCI bit if specified in the first CCW.

4. The I/O device or channel detected an equipment or programming error during execution of the instruction. The CSW identifies the error condition. The channel-end and busy bits are off, unless the error was detected after the device was selected, and the device was found to be busy, in which case the busy bit, as well as any bits indicating pending interruption conditions, are on. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation has not been initiated. No interruption conditions are generated at the I/O device or subchannel.

On the multiplexor channel, START I/O causes the addressed device to be selected and the operation to be initiated only after the channel has serviced all outstanding requests for data transfer for previously initiated operations.

Resulting Condition Code:

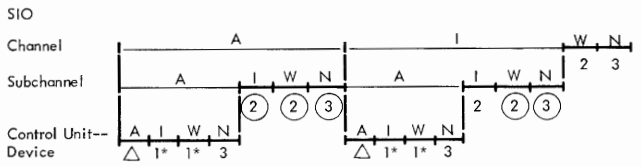
0 I/O operation initiated and channel proceeding with its execution
1 CSW stored
2 Channel or subchannel busy
3 Not operational

Program Interruptions:

Privileged operation.

The condition code set by START I/O for all possible states of the I/O system is shown graphically in

Figure 23. See the "States of the Input/Output System" section of this manual for thorough definition of the A, I, W, and N states.



A = Available

I = Interruption Pending

W = Working

N = Not Operational

* = CSW Stored

When △ = 0, a regular I/O operation has been initiated and channel is proceeding with its execution.

When △ = 1*, an immediate operation has been initiated and no command chaining is taking place; or the device is not ready; or an error condition has been detected.

Note: Encircled condition codes pertain to conditions that can occur only on the multiplexor channel.

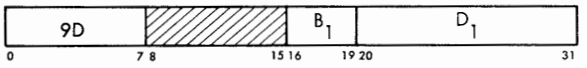
Figure 23. Condition Code set by START I/O

Programming Note

When the channel detects a programming error during execution of START I/O and the addressed device contains an interruption condition, with the channel and subchannel in the available state, START I/O may or may not clear the interruption condition, depending on the type of error and the model. If the instruction has caused the device to be interrogated, as indicated by the presence of the busy bit in the CSW, the interruption condition has been cleared, and the CSW contains program or protection check, as well as the status from the device.

Test I/O

TIO $D_1(B_1)$ [SI]



The state of the addressed channel, subchannel, and device is indicated by setting the condition code in the PSW and, under certain conditions, by storing the CSW. Pending interruption conditions may be cleared. The instruction TEST I/O is executed only when the CPU is in the supervisor state.

Bit positions 16-31 of the sum formed by the addition of the content of register B_1 and the content of the D_1 field identify the channel, subchannel, and I/O device to which the instruction applies.

When any of the following conditions occurs with the channel either available or in the interruption-pending state, TEST I/O causes the CSW at location 64 to be stored. The content of the entire CSW pertains to the I/O device addressed by the instruction.

1. The subchannel contains a pending interruption condition due to a terminated operation at the addressed device. The CSW identifies the interruption condition, and the interruption condition is cleared. The protection key, command address, and count fields contain the final values for the I/O operation, and the status may include other bits provided by the channel and the device. The interruption condition in the subchannel is not cleared, and the CSW is not stored if the interruption condition is associated with an operation on a device other than the one addressed.

2. The subchannel is available and the I/O device contains a pending interruption condition due to device end or attention, the control unit contains a pending control unit end for the addressed device, or, on the selector channel, the control unit contains for the addressed device a pending channel end following the execution of HALT I/O. The CSW unit-status field identifies the interruption condition and may contain other bits provided by the device or control unit. The interruption condition is cleared. The busy bit in the CSW is off. The other fields of the CSW contain zeros unless an equipment error is detected.

3. The subchannel is available and the I/O device or the control unit is executing a previously initiated operation or the control unit has a pending interruption condition associated with a device other than the one addressed. The CSW unit-status field contains the busy bit or, if the control unit is busy, the busy and status-modifier bits. Other fields of the CSW contain zeros unless an equipment error is detected.

4. The subchannel is available and the I/O device or channel detected an equipment error during execution of the instruction or the addressed device is in the not-ready state and does not have any pending interruption condition. The CSW identifies the error conditions. If the device is not ready, unit check is indicated. No interruption conditions are generated at the I/O device or the subchannel.

When TEST I/O is used to clear an interruption condition from the subchannel and the channel has not yet accepted the condition from the device, the instruction causes the device to be selected and the interruption condition in the device to be cleared. During certain I/O operations, some types of devices cannot provide their current status in response to TEST I/O. The tape control unit, for example, is in such a state when it has provided the channel-end condition

and is executing the backspace-file operation. When TEST I/O is issued to a control unit in such a state, the unit-status field of the CSW contains the busy and status-modifier bits, with zeros in the other CSW fields. The interruption condition in the device and in the subchannel is not cleared.

On some types of devices, such as the 2702 Transmission Control, the device never provides its current status in response to TEST I/O, and an interruption condition can be cleared only by permitting an I/O interruption. When TEST I/O is issued to such a device, the unit-status field contains the status-modifier bit, with zeros in the other CSW fields. The interruption condition in the device and in the subchannel, if any, is not cleared.

However, at the time the channel assigns the highest priority for interruptions to a condition associated with an operation at the subchannel, the channel accepts the status from the device and clears the corresponding condition at the device. When TEST I/O is addressed to a device for which the channel has already accepted the interruption condition, the device is not selected, and the condition in the subchannel is cleared regardless of the type of device and its present state. The CSW contains unit status and other information associated with the interruption condition.

On the multiplexor channel, TEST I/O causes the addressed device to be selected only after the channel has serviced all outstanding requests for data transfer for previously initiated operations.

Resulting Condition Code:

0 Available
1 CSW stored
2 Channel or subchannel busy
3 Not operational

Program Interruptions:

Privileged operation

The condition code set by TEST I/O for all possible states of the I/O system is shown graphically in Figure 24. See the "States of the Input/Output System" section of this manual for thorough definition of the A, I, W, and N states.

Programming Notes

Masking of channels provides the program a means of controlling the priority of I/O interruptions selectively by channels. The priority of devices attached on a channel is fixed and cannot be controlled by the program. The instruction TEST I/O permits the program to clear interruption conditions selectively by I/O device.

When a CSW is stored by TEST I/O, the interface-control-check and channel-control-check indications may be due to a condition already existing in the channel or due to a condition created by TEST I/O. Similarly, presence of the unit-check bit in the absence of channel-end, control-unit-end or device-end bits may be due to a condition created by the preceding operation, the not-ready state, or an equipment error detected during the execution of TEST I/O. The TEST I/O cannot be used to clear a pending interruption condition due to the PCI flag while the subchannel is in the working state.

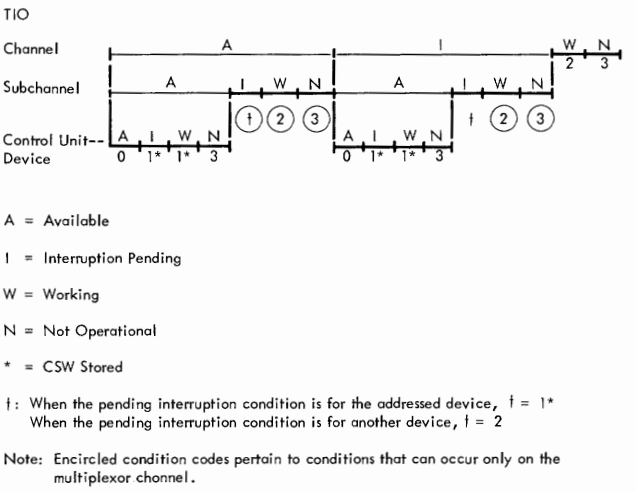
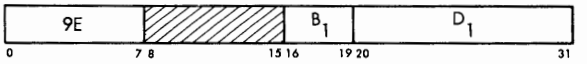


Figure 24. Condition Code set by TEST I/O

Halt I/O

HIO ***$D_1(B_1)$*** ***[S]***



Execution of the current I/O operation at the addressed I/O device, subchannel, or channel is terminated. The subsequent state of the subchannel depends on the type of channel. The instruction HALT I/O is executed only when the CPU is in the supervisor state.

Bit positions 16-31 of the sum formed by the addition of the contents of register B_1 and the contents of the D_1 field identify the channel, and, when the channel is not working, identify the subchannel and the I/O device to which the instruction applies.

When the channel is either available or in the interruption-pending state, with the subchannel either available or working, HALT I/O causes the addressed device to be selected and to be signaled to terminate the current operation, if any. If the subchannel is available, its state is not affected. If, on the multiplexor channel, the subchannel is working, data transfer is

immediately terminated, but the subchannel remains in the working state until the device provides the next status byte, whereupon the subchannel is placed in the interruption-pending state.

When HALT I/O is issued to a channel operating in the burst mode, data transfer for the burst operation is terminated, and the device performing the burst operation is immediately disconnected from the channel. The subchannel and I/O device address in the instruction, in this case, is ignored.

The termination of a burst operation by HALT I/O on the selector channel causes the channel and subchannel to be placed in the interruption-pending state. Generation of the interruption condition is not contingent on the receipt of a status byte from the device. When HALT I/O causes a burst operation on the multiplexor channel to be terminated, the subchannel associated with the burst operation remains in the working state until the device provides channel end, whereupon the subchannel enters the interruption-pending state.

On the multiplexor channel operating in the multiplex mode, the device is selected and the instruction is executed only after the channel has serviced all outstanding requests for data transfer for previously initiated operations, including the operation to be halted. If the control unit does not accept the HALT-I/O signal because it is in the not-operational or control-unit-busy state, the subchannel, if working, is set up to signal termination of device operation the next time the device requests or offers a byte of data. If command chaining is indicated in the subchannel and the device presents status next, chaining is suppressed.

When the addressed subchannel has a pending interruption condition, with the channel in the available or interruption-pending state, HALT I/O does not cause any action.

When any of the following conditions occur, HALT I/O causes the status portion, bit positions 32-47, of the CSW at location 64 to be replaced by a new set of status bits. The contents of the other fields of the CSW are not changed. The CSW stored by HALT I/O pertains only to the execution of HALT I/O and does not describe under what conditions the I/O operation at the addressed subchannel is terminated. The extent of data transfer, and the conditions of termination of the operation at the subchannel, are provided in the CSW associated with the interruption condition due to the termination.

1. The addressed device has been selected and signaled to terminate the current operation. The CSW contains zeros in the status field unless an equipment error is detected.

2. The channel attempted to select the addressed device, but the control unit could not accept the HALT-I/O signal because it is executing a previously initiated operation or has an interruption condition associated with a device other than the one addressed. The signal to terminate the operation has not been transmitted to the device, and the subchannel, if in the working state, has been set up to signal termination the next time the device identifies itself. The CSW unit-status field contains the busy and status-modifier bits. The channel-status field contains zeros unless an equipment error is detected.

3. The channel detected an equipment malfunction during the execution of HALT I/O. The status bits in the CSW identify the error condition. The state of the channel and the progress of the I/O operation are unpredictable.

When HALT I/O causes data transfer to be terminated, the control unit associated with the operation remains unavailable until the data-handling portion of the operation in the control unit is terminated. Termination of data-transfer portion of the operation is signaled by generation of channel end, which may occur at the normal time for the operation, earlier, or later, depending on the operation and type of device. If the control unit is shared, all devices attached to the control unit appear in the working state until the channel-end condition is accepted by the CPU. The I/O device executing the terminated operation remains in the working state until termination of the inherent cycle of the operation, at which time device end is generated. If blocks of data at the device are defined, such as reading on magnetic tape, the recording medium is advanced to the beginning of the next block.

When HALT I/O is issued at a time when the subchannel is available and no burst operation is in progress, the effect of the HALT-I/O signal depends on the type of device and its state and is specified in the SRL publication for the device. The HALT-I/O signal has no effect on devices that are not in the working state or are executing an operation of a fixed duration, such as rewinding tape or positioning a disk access mechanism. If the device is executing a type of operation that is variable in duration, the device interprets the signal as one to terminate the operation. Pending attention or device-end conditions at the device are not reset.

Resulting Condition Code:

0 Interruption pending in subchannel
1 CSW stored
2 Burst operation terminated
3 Not operational

Program Interruptions:

Privileged operation

The condition code set by HALT I/O for all possible states of the I/O system is shown graphically in Fig-

ure 25. See the "States of the Input/Output System" section of this manual for thorough definition of the A, I, W, and N states.

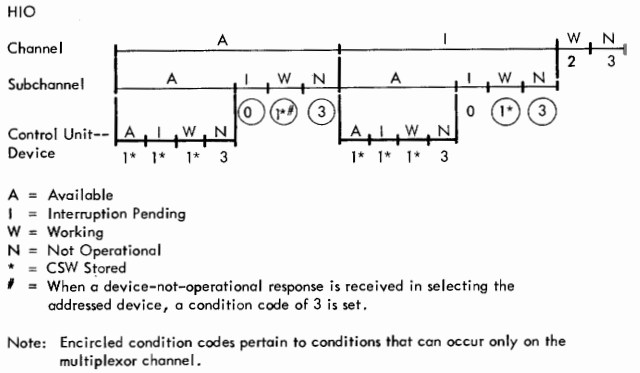


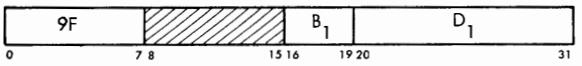
Figure 25. Condition Code set by HALT I/O

Programming Note

The instruction HALT I/O provides the program a means of terminating an I/O operation before all data specified in the operation have been transferred or before the operation at the device has reached its normal ending point. It permits the program to immediately free the selector channel for an operation of higher priority. On the multiplexor channel, HALT I/O provides a means of controlling real-time operations and permits the program to terminate data transmission on a communication line.

Test Channel

TCH $D_1(B_1)$ [SI]



The condition code in the PSW is set to indicate the state of the addressed channel. The state of the channel is not affected, and no action is caused. The instruction TEST CHANNEL is executed only when the CPU is in the supervisor state.

Bit positions 16-23 of the sum formed by the addition of the content of register B_1 and the content of the D_1 field identify the channel to which the instruction applies. Bit positions 24-31 of the address are ignored.

The instruction TEST CHANNEL inspects only the state of the addressed channel. It tests whether the channel is operating in the burst mode, is aware of any outstanding interruption conditions from its devices, or is not operational. When the channel is operating in the burst mode and contains a pending interruption condition, the condition code is set as for operation in the burst mode. When none of these conditions exists, the available state is indicated. No device is selected, and, on the multiplexor channel, the subchannels are not interrogated.

Resulting Condition Code:

0 Channel available
1 Interruption pending in channel
2 Channel operating in burst mode
3 Channel not operational

Program Interruptions:

Privileged operation

The condition code set by TEST CHANNEL for all possible states of the addressed channel is shown graphically in Figure 26. See the "States of the Input/Output System" section of this manual for thorough definition of the A, I, W, and N states.

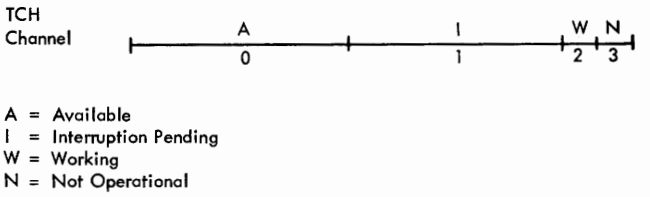


Figure 26. Condition Code set by TEST CHANNEL

Input/Output Instruction Exception Handling

Before the channel is signaled to execute an I/O instruction, the instruction is tested for validity by the CPU. Exceptional conditions detected at this time cause a program interruption. When the interruption occurs, the current PSW is stored as the program old PSW and is replaced by the program new PSW. The interruption code in the old PSW identifies the cause of the interruption.

The following exception may cause a program interruption:

Privileged Operation: An I/O instruction is encountered when the CPU is in the problem state. The instruction is suppressed before the channel has been signaled to execute it. The CSW, the condition code in the PSW, and the state of the addressed subchannel and I/O device are not affected by the attempt to execute an I/O instruction while in the problem state.

Execution of Input/Output Operations

The channel can execute six commands: write, read, read backward, control, sense, and transfer in channel. Each command except transfer in channel initiates a corresponding I/O operation. The term "I/O operation" refers to the activity initiated by a command in the I/O device and associated subchannel. The sub-

channel is involved with the execution of the operation from the initiation of the command until the channel-end signal is received or, in the case of command chaining, until the device-end signal is received. The operation in the device lasts until device end occurs.

Blocking of Data

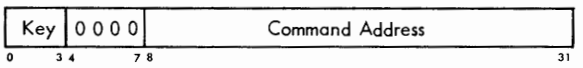
Data recorded by an I/O device may be divided into blocks. A block of data is defined for each type of I/O device as the amount of information recorded in the interval between adjacent starting and stopping points of the device. The length of a block depends on the document; for example, a block can be a card, a line of printing, or the information recorded between two consecutive gaps on magnetic tape.

The maximum amount of information that can be transferred in one I/O operation is one block. An I/O operation is terminated when the associated storage area is exhausted or the end of the block is reached, whichever occurs first. For some operations, such as writing on a magnetic tape unit or on an inquiry station, blocks are not defined, and the amount of information transferred is controlled only by the program.

Channel Address Word

The channel address word (CAW) specifies the storage protection key and the address of the first CCW associated with START I/O. It is assigned location 72. The channel refers to the CAW only during the execution of START I/O. The pertinent information thereafter is stored in the subchannel, and the program is free to change the content of the CAW. Fetching of the CAW by the channel does not affect the contents of location 72.

The CAW has the following format:



The fields in the CAW are allocated for the following purposes:

Protection Key: Bits 0-3 form the protection key for all commands associated with START I/O. This key is matched with a key in storage whenever reference is made to main storage.

Command Address: Bits 8-31 designate the location of the first CCW in main storage.

Bit positions 4-7 of the CAW must contain zeros. When the protection feature is not implemented, the protection key must be zero. The three low-order bits of the command address must be zero to specify the CCW on integral boundaries for double words. If any of these restrictions is violated or if the command address specifies a location protected for fetching or outside the main storage of the particular installation, START I/O causes the status portion of the CSW to be stored with the protection-check or program-check bit on. In this event, the I/O operation is not initiated.

Programming Note

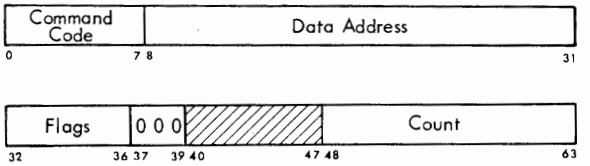
Bit positions 4-7 of the CAW, which presently must contain zeros, may in the future be assigned for the control of new functions. It is therefore recommended that these bit positions not be set to one for the purpose of obtaining an intentional program-check indication.

Channel Command Word

The channel command word (CCW) specifies the command to be executed and, for commands initiating 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, and more than one can be associated with a START I/O. 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 sequence is obtained when the operation has progressed to the point where the additional CCW is needed. Fetching of the CCW's by the channel does not affect the contents of the location in main storage.

The CCW has the following format:



The fields in the CCW are allocated for the following purposes:

Command Code: Bits 0-7 specify the operation to be performed.

Data Address: Bits 8-31 specify the location of an eight-bit byte in main storage. It is the first location referred to in the area designated by the CCW.

Chain-Data (CD) Flag: Bit 32, when one, specifies chaining of data. It causes the storage area designated by the next CCW to be used with the current operation.

Chain-Command (CC) Flag: Bit 33, when one, and when the CD flag is zero, specifies chaining of commands. It causes the operation specified by the command code in the next CCW to be initiated on normal completion of the current operation.

Suppress-Length-Indication (SLI) Flag: Bit 34 controls whether an incorrect length condition is to be

indicated to the program. When this bit is one and the CD flag is zero in the last CCW used, the incorrect length indication is suppressed. When both the CC and SLI flags are one, command chaining takes place regardless of the presence of an incorrect length condition.

Skip (SKIP) Flag: Bit 35, when one, specifies suppression of transfer of information to storage during a read, read backward, or sense operation..

Program-Controlled-Interruption (PCI) Flag: Bit 36, when one, causes the channel to generate an interruption condition when the CCW takes control of the channel. When bit 36 is zero, normal operation takes place.

Count: Bits 48-63 specify the number of eight-bit byte locations in the storage area designated by the CCW.

Bit positions 37-39 of every CCW other than one specifying transfer in channel must contain zeros. Violation of this restriction generates the program-check condition. When the first CCW designated by the CAW does not contain the required zeros, the I/O operation is not initiated, and the status portion of the CSW with the program-check indication is stored during execution of START I/O. Detection of this condition during data chaining causes the I/O device to be signaled to terminate the operation. When the absence of these zeros is detected during command chaining, the new operation is not initiated, and an interruption condition is generated.

The content of bit positions 40-47 of the CCW is ignored.

Programming Note

Bit positions 37-39, of the CCW, which presently must contain zeros, may in the future be assigned for the control of new functions. It is therefore recommended that these bit positions not be set to one for the purpose of obtaining a program-check indication.

Command Code

The command code, bit positions 0-7 of the CCW, specifies to the channel and the I/O device the operation to be performed.

The two low-order bits or, when these bits are 00, the four low-order bits of the command code identify the operation to the channel. The channel distinguishes among the following four operations:

Output forward (write, control)
Input forward (read, sense)
Input backward (read backward)
Branching (transfer in channel)

The channel ignores the high-order bits of the command code.

Commands that initiate I/O operations (write, read, read backward, control, and sense) cause all eight bits of the command code to be transferred to the I/O device. In these command codes, the high-order bit positions contain modifier bits. The modifier bits specify to the device how the command is to be executed. They may cause, for example, the device to compare data received during a write operation with data previously recorded, and they may specify such conditions as recording density and parity. For the control command, the modifier bits may contain the order code specifying the control function to be performed. The meaning of the modifier bits depends on the type of I/O device and is specified in the SRL publication for the device.

The command code assignment is listed in the following table. The symbol x indicates that the bit position is ignored; M identifies a modifier bit.

CODE	COMMAND
x x x x 0 0 00	Invalid
MMMM 0 1 00	Sense
x x x x 1 0 00	Transfer in channel
MMMM 1 1 00	Read backward
MMMM MM01	Write
MMMM MM10	Read
MMMM MM11	Control

Whenever the channel detects an invalid command code during the initiation of a command, the program-check condition is generated. When the first CCW designated by the CAW contains an invalid command code, the status portion of the CSW with the program-check indication is stored during execution of START I/O. When the invalid code is detected during command chaining, the new operation is not initiated, and an interruption condition is generated. The command code is ignored during data chaining, unless it specifies transfer in channel.

Definition of Storage Area

The main-storage area associated with an I/O operation is defined by CCW's. A CCW defines an area by specifying the address of the first eight-bit byte to be transferred and the number of consecutive eight-bit bytes contained in the area. The address of the first byte appears in the data-address field of the CCW. The number of bytes contained in the storage area is specified in the count field.

In write, read, control, and sense operations storage locations are used in ascending order of addresses. As information is transferred to or from main storage, the content of the address field is incremented, and the content of the count field is decremented. The read-backward operation causes data to be placed in storage in a descending order of addresses, and both the count and the address are decremented. When the count in any operation reaches zero, the storage area defined by the CCW is exhausted.

Any main-storage location provided in the system can be used to transfer data to or from an I/O device, provided that the location is not protected for the type of reference. Similarly, the CCW's can be specified in any part of available main storage, provided the location is not protected for a fetch-type reference. When the channel attempts to refer to a protected location, the protection-check condition is generated, and the device is signaled to terminate the operation.

In the event the channel refers to a location not provided in the system, the program-check condition is generated. The method of indicating the error condition and terminating the I/O operation upon detection of an invalid address depends on whether or not the address exceeds the addressing capacity of the model. The term "addressing capacity" refers to the model's facilities for addressing main storage. Most models have facilities for addressing up to 16,777,216 bytes regardless of the storage provided in the particular installation. In some models, however, the addressing facilities in the channel are restricted to main storage of less than 16,777,216 bytes. When the first CCW designated by the CAW is at a nonexistent location or the first CCW contains a data address exceeding the addressing capacity of the model, the I/O operation is not initiated and the status portion of the CSW with the program-check indication is stored during the execution of START I/O. Invalid data addresses within the addressing capacity of the model, as well as any invalid CCW addresses detected on chaining, are indicated to the program with the interruption conditions at the termination of the operation or chain of operations.

During an output operation, the channel may fetch data from the main storage prior to the time the I/O device requests the data. As many as 16 bytes may be prefetched and buffered. Similarly, on data chaining during an output operation, the channel may fetch the new CCW when as many as 16 bytes remain to be transferred under the control of the current CCW. When the I/O operation uses data and CCW's from locations near the end of the available storage, such prefetching may cause the channel to refer to locations that do not exist. Invalid addresses detected during prefetching of data or CCW's do not affect the execution of the operation and do not cause error indications until the I/O operation actually attempts to use the information. If the operation is terminated by the I/O device or by HALT I/O before the invalid information is needed, the condition is not brought to the attention of the program.

Storage addresses do not wrap around to location 0 unless the system has the maximum addressable storage (16,777,216 bytes). When the maximum addressable storage is provided, location 0 follows location 16,777,215 and, on reading backward, location 16,777,215 follows location 0.

The count field in the CCW can specify any number of bytes up to 65,535. Except for a CCW specifying transfer in channel, the count field may not contain the value zero. Whenever the count field in the CCW initially contains a zero, the program-check condition is generated. When this occurs in the first CCW designated by the CAW, the operation is not initiated, and the status portion of the CSW with the program-check indication is stored during execution of START I/O. When a count of zero is detected during data chaining, the I/O device is signaled to terminate the operation. Detection of a count of zero during command chaining suppresses initiation of the new operation and generates an interruption condition.

Chaining

When the channel has performed the transfer of information specified by a CCW, it can continue the activity initiated by START I/O by fetching a new CCW. Such fetching of a new CCW is called chaining, and the CCW's belonging to such a sequence are said to be chained.

Chaining takes place only between CCW's located in successive double-word locations in storage. It proceeds in an ascending order of addresses; that is, the address of the new CCW is obtained by adding eight to the address of the current CCW. Two chains of CCW's located in noncontiguous storage areas can be coupled for chaining purposes by a transfer in channel command. All CCW's in a chain apply to the I/O device specified in the original START I/O.

Two types of chaining are provided: chaining of data and chaining of commands. Chaining is controlled by the chain-data (CD) and chain-command (CC) flags in conjunction with the suppress-length-indication (SLI) flag in the CCW. These flags specify the action to be taken by the channel upon the exhaustion of the current CCW and upon receipt of ending status from the device, as shown in Figure 27.

The specification of chaining is effectively propagated through a transfer in channel command. When in the process of chaining a transfer-in-channel command is fetched, the CCW designated by the transfer in channel is used for the type of chaining specified in the CCW preceding the transfer in channel.

The CD and CC flags are ignored in the transfer-in-channel command.

Data Chaining

During data chaining, the new CCW fetched by the channel defines a new storage area for the original I/O

operation. Execution of the operation at the I/O device is not affected. Data chaining occurs only when all data designated by the current CCW have been transferred to or from the device and causes the operation to continue, using the storage area designated by the new CCW. The content of the command-code field of the new CCW is ignored, unless it specifies transfer in channel.

Data chaining is considered to occur immediately after the last byte of data designated by the current CCW has been transferred to or from the device. When the last byte has been placed in main storage or accepted by the device, the new CCW takes over the control of the operation and replaces the pertinent information in the subchannel. If the device sends channel end after exhausting the count of the current CCW but before transferring any data to or from the storage area designated by the new CCW, the CSW associated with the termination pertains to the new CCW.

If programming errors are detected in the new CCW or during its fetching, an error indication is generated, and the device is signaled to terminate the operation when it attempts to transfer data designated by the new CCW. If the device signals the channel-end condition before transferring any data designated by the new CCW, program check or protection check is indicated in the CSW associated with the termination. Unless the address of the new CCW is invalid, the location is protected for fetching, or programming errors are detected in an intervening transfer-in-channel command, the content of the CSW pertains to the new CCW. A data address referring to a nonexistent or protected area causes an error indication only after the I/O device has attempted to transfer data to or from the invalid location, but an address exceeding the addressing capacity of the model is detected immediately upon fetching the CCW.

Data chaining during an input operation causes the new CCW to be fetched when all data designated by the current CCW have been placed in main storage. On an output operation, the channel may fetch the new CCW from main storage ahead of the time data chaining occurs. The earliest such prefetching may occur is when 16 bytes still remain to be transferred under the control of the current CCW. Any programming errors in the prefetched CCW, however, do not affect the execution of the operation until all data designated by the current CCW have been transferred to the I/O device. If the device terminates the operation before all data designated by the current CCW have been transferred, the conditions associated with the prefetched CCW are not indicated to the program.

Only one CCW describing a data area may be prefetched and buffered in the channel. If the prefetched CCW specifies transfer in channel, only one more CCW is fetched before the exhaustion of the current CCW.

Flags in Current CCW			Action in Channel upon Exhaustion of Count or Receipt of Channel End			
			IMMEDIATE OPERATION	REGULAR OPERATION		
CD	CC	SLI		Count Exhausted, End of Block at Device not Reached	Count Exhausted and Channel End from Device	Count not Exhausted and Channel End from Device
0	0	0	End, -	Stop, IL	End, -	End, IL
0	0	1	End, -	Stop, -	End, -	End, -
0	1	0	Chain Command	Stop, IL	Chain Command	End, IL
0	1	1	Chain Command	Chain Command	Chain Command	Chain Command
1	0	0	End, -	Chain Data	*	End, IL
1	0	1	End, -	Chain Data	*	End, IL
1	1	0	End, -	Chain Data	*	End, IL
1	1	1	End, -	Chain Data	*	End, IL

End	The operation is terminated. If the operation is immediate and has been specified by the first CCW associated with a START I/O, a condition code of 1 is set, and the status portion of the CSW is stored as part of the execution of the START I/O. In all other cases an interruption condition is generated in the subchannel.
Stop	The device is signaled to terminate data transfer, but the subchannel remains in the working state until channel end is received; at this time an interruption condition is generated in the subchannel.
IL	Incorrect length is indicated with the interruption condition.
-	Incorrect length is not indicated.
Chain Command	The channel performs command chaining upon receipt of device end.
Chain Data	The channel immediately fetches a new CCW for the same operation.
*	The situation where the count is zero but data chaining is indicated at the time the device provides channel end cannot validly occur. When data chaining is indicated, the channel fetches the new CCW after transferring the last byte of data designated by the current CCW but before the device provides the next request for data or status transfer. As a result, the channel recognizes the channel end from the device only after it has fetched the new CCW, which cannot contain a count of zero unless a programming error has been made.

Figure 27. Effect of CD, CC, and SLI Flags on an I/O Operation

Programming Note

Data chaining may be used to rearrange information as it is transferred between main storage and an I/O device. Data chaining permits blocks of information to be transferred to or from noncontiguous areas of storage, and, when used in conjunction with the skipping function, data chaining enables the program to place in main storage selected portions of a block of data.

When, during an input operation, the program specifies data chaining to a location into which data have been placed under the control of the current CCW, the channel fetches the new content of the location, even if the location contains the last byte transferred under the control of the current CCW. When a channel program data-chains to a CCW placed in storage by the CCW specifying data chaining, the input block is said to be self-describing. A self-describing block contains one or more CCW's that specify storage locations and counts for subsequent data in the same input block.

When data chaining is used during a read-backward operation, the channel places data in storage in a descending sequence, but fetches CCW's in an ascending sequence. Therefore, if a magnetic tape block is to be written so that it can be read in either the forward or backward direction as a self-describing block, the CCW must be written at both the beginning and the end of the block. If more than one CCW is to be used, the order of the CCW's must be reversed at the end of the block because the storage areas associated with the CCW's are used in reverse sequence.

Use of self-describing blocks, however, is equivalent to use of unchecked data. An I/O data transfer malfunction that affects validity of a block of information is signaled only at the completion of data transfer. The error condition normally does not prematurely terminate or otherwise affect the execution of the operation. Thus, there is no assurance that a CCW read as data is valid until the operation is completed. If the CCW thus read is in error, use of the CCW in the current operation may cause subsequent data to be placed in wrong locations in main storage with resultant destruction of its contents, subject to the control of the protection system.

Command Chaining

During command chaining, the new CCW fetched by the channel specifies a new I/O operation. The channel fetches the new CCW and initiates the new operation upon the receipt of the device-end signal for the current operation. When command chaining takes place, the completion of the current operation does not cause an I/O interruption, and the count indicating the amount of data transferred during the current operation is not made available to the program. For operations involving data transfer, the new command always applies to the next block of data at the device.

Command chaining takes place and the new operation is initiated only if no unusual conditions have been detected in the current operation. In particular, the channel initiates a new I/O operation by command chaining upon receipt of a status byte containing only the following bit combinations: device end, device end and status modifier, device end and channel end, device end and channel end and status modifier. In the former two cases a channel end must have been signaled before device end, with all other status bits off. If a condition such as attention, unit check, unit exception, incorrect length, program check, or protection check has occurred, the sequence of operations is terminated, and the status associated with the current operation causes an interruption condition to be generated. The new CCW in this case is not fetched. The incorrect-length condition does not suppress command chaining if the current CCW has the SLI flag on.

An exception to sequential chaining of CCW's occurs when the I/O device presents the status-modifier condition with the device-end signal. When command chaining is specified and no unusual conditions have been detected, the combination of status-modifier and device-end bits causes the channel to fetch and chain to the CCW whose main-storage address is 16 higher than that of the current CCW.

When both command and data chaining are used, the first CCW associated with the operation specifies the operation to be executed, and the last CCW indicates whether another operation follows.

Programming Note

Command chaining makes it possible for the program to initiate transfer of multiple blocks of data by means of a single START I/O. It also permits a subchannel to be set up for execution of auxiliary functions, such as positioning the disk access mechanism, and for data transfer operations without interference by the program at the end of each operation. Command chaining, in conjunction with the status-modifier condition, permits the channel to modify the normal sequence of operations in response to signals provided by the I/O device.

Skipping

Skipping is the suppression of main-storage references during an I/O operation. It is defined only for read, read backward, and sense operations and is controlled by the skip flag, which can be specified individually for each CCW. When the skip flag is one, skipping occurs; when zero, normal operation takes place. The

setting of the skip flag is ignored in all other operations.

Skipping affects only the handling of information by the channel. The operation at the I/O device proceeds normally, and information is transferred to the channel. The channel keeps updating the count but does not place the information in main storage. If the chain-command or chain-data flag is one, a new CCW is obtained when the count reaches zero. In the case of data chaining, normal operation is resumed if the skip flag in the new CCW is zero.

No checking for invalid or protected data addresses takes place during skipping, except that the initial data address in the CCW cannot exceed the addressing capacity of the model.

Programming Note

Skipping, when combined with data chaining, permits the program to place in main storage selected portions of a block of information from an I/O device.

Program-Controlled Interruption

The program-controlled interruption (PCI) function permits the program to cause an I/O interruption during execution of an I/O operation. The function is controlled by the PCI flag in the CCW. The flag can be on either in the first CCW specified by START I/O or in a CCW fetched during chaining. Neither the PCI flag nor the associated interruption affects the execution of the current operation.

Whenever the PCI flag in the CCW is on, the channel attempts to interrupt the program. When the first CCW associated with an operation contains the PCI flag, either initially or upon command chaining, the interruption may occur as early as immediately upon the initiation of the operation. The PCI flag in a CCW fetched on data chaining causes the interruption to occur after all data designated by the preceding CCW have been transferred. The time of the interruption, however, depends on the model and the current activity in the system and may be delayed even if the channel is not masked. No predictable relation exists between the time the interruption due to the PCI flag occurs and the progress of data transfer to or from the area designated by the CCW, but the fields within the CSW pertain to the same instant of time.

If chaining occurs before the interruption due to the PCI flag has taken place, the PCI condition is carried over to the new CCW. This carryover occurs both on data and command chaining and, in either case, the condition is propagated through the transfer-in-channel command. The PCI conditions are not stacked; that is, if another CCW is fetched with a PCI flag before the interruption due to the PCI flag of the previous CCW has occurred, only one interruption takes place.

A CSW containing the PCI bit may be stored by an interruption while the operation is still proceeding or by an interruption or TEST I/O upon the termination of the operation. It cannot be stored by TEST I/O while the subchannel is in the working state.

When the CSW is stored by an interruption before the operation or chain of operations has been terminated, the command address is eight higher than the address of the current CCW, and the count is unpredictable. All unit-status bits in the CSW are zero. If the channel has detected any unusual conditions, such as channel data check, program check, or protection check by the time the interruption occurs, the corresponding channel-status bit is on, although the condition in the subchannel is not reset and is indicated again upon the termination of the operation.

Presence of any unit-status bit in the CSW indicates that the operation or chain of operations has been terminated. The CSW in this case has its regular format with the PCI bit added.

However, when the interruption condition due to the PCI flag has been delayed until the operation at the subchannel has been terminated, two interruptions from the subchannel may still take place, with the first interruption indicating and clearing the PCI condition alone, and the second providing the CSW associated with the ending status. Whether one or two interruptions occur depends on the model, and on whether the PCI condition has been assigned the highest priority for interruption at the time of termination. The TEST I/O addressed to the device associated with an interruption condition in the subchannel clears the PCI condition as well as the one associated with the termination.

The setting of the PCI flag is inspected in every CCW except those specifying transfer in channel. In a CCW specifying transfer in channel, the setting of the flag is ignored. The PCI flag is ignored also during initial program loading.

Programming Note

Since no unit-status bits are placed in the CSW associated with the termination of an operation on the selector channel by HALT I/O, the presence of a unit-status bit with the PCI bit is not a necessary condition for the operation to be terminated. When the selector channel contains the PCI bit at the time the operation is terminated by HALT I/O, the CSW associated with the termination is indistinguishable from the CSW provided by an interruption during execution of the operation.

Program-controlled interruption provides a means of alerting the program of the progress of chaining

during an I/O operation. It permits programmed dynamic main-storage allocation.

Commands

The following table lists the command codes for the six commands and indicates which flags are defined for each command. The flags are ignored for all commands for which they are not defined.

NAME	CODE	FLAG
Write	MMMM MM01	CD CC SLI PCI
Read	MMMM MM10	CD CC SLI SKIP PCI
Read backward	MMMM 1 1 00	CD CC SLI SKIP PCI
Control	MMMM MM11	CD CC SLI PCI
Sense	MMMM 0 1 00	CD CC SLI SKIP PCI
Transfer in channel	X X X X 1 0 00	

NOTES

CD	Chain data
CC	Chain command
SLI	Suppress length indication
SKIP	Skip
PCI	Program-controlled interruption
M	Modifier bit
X	Ignored

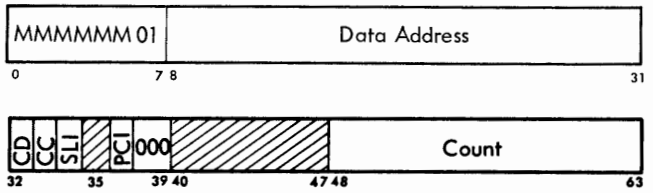
All flags have individual significance, except that the CC and SLI flags are ignored when the CD flag is on. The SLI flag is ignored on immediate operations, in which case the incorrect-length indication is suppressed regardless of the setting of the flag. The PCI flag is ignored during initial program loading.

Each command is described below with an illustration of its CCW format.

Programming Note

A malfunction that affects the validity of data transferred in an I/O operation is signaled at the end of the operation by means of unit check or channel data check, depending on whether the device (control unit) or the channel detected the error. In order to make use of the checking facilities provided in the system, data read in an input operation should not be used until the end of the operation has been reached and the validity of the data has been checked. Similarly, on writing, the copy of data in main storage should not be destroyed until the program has verified that no malfunction affecting the transfer and recording of data was detected.

Write



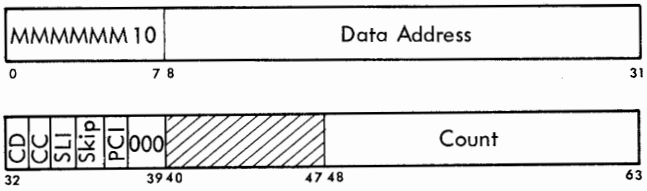
A write operation is initiated at the I/O device, and the subchannel is set up to transfer data from main storage to the I/O device. Data in storage are fetched in an ascending order of addresses, starting with the address specified in the CCW.

A CCW used in a write operation is inspected for the CD, CC, SLI, and the PCI flags. The setting of the SKIP flag is ignored. Bit positions 0-5 of the CCW contain modifier bits.

Programming Note

When writing on devices for which block length is not defined, such as a magnetic tape unit or an inquiry station, the amount of data written is controlled only by the count in the CCW. Every operation terminated under count control causes the incorrect-length indication, unless the indication is suppressed by the SLI flag.

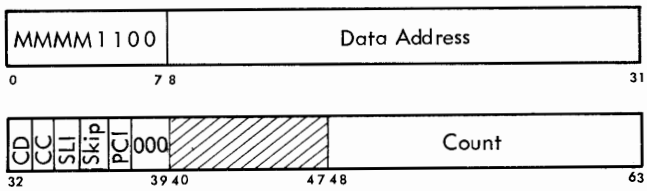
Read



A read operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to main storage. For devices such as magnetic tape units, disk storage, and card equipment, the bytes of data within a block are provided in the same sequence as written by means of a write command. Data in storage are placed in an ascending order of addresses, starting with the address specified in the CCW.

A CCW used in a read operation is inspected for every one of the five flags — CD, CC, SLI, SKIP, and PCI. Bit positions 0-5 of the CCW contain modifier bits.

Read Backward

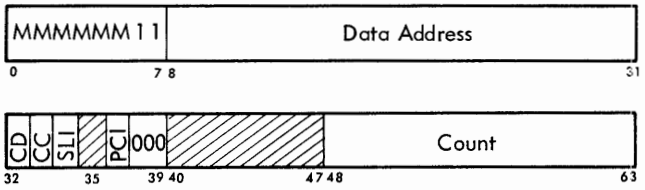


A read-backward operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to main storage. On magnetic tape units, read backward causes reading to be performed with the tape moving backwards. The bytes of data within a block are sent to the channel in a sequence opposite to that on writing. The channel places the bytes in storage in a descending order of address, start-

ing with the address specified in the CCW. The bits within an eight-bit byte are in the same order as sent to the device on writing.

A CCW used in a read-backward operation is inspected for every one of the five flags — CD, CC, SLI, SKIP, and PCI. Bit positions 0-3 of the CCW contain modifier bits.

Control



A control operation is initiated at the I/O device, and the subchannel is set up to transfer data from main storage to the device. The device interprets the data as control information. The control information, if any, is fetched from storage in an ascending order of addresses, starting with the address specified in the CCW. A control command is used to initiate at the I/O device an operation not involving transfer of data — such as backspacing or rewinding magnetic tape or positioning a disk access mechanism.

For most control functions, the entire operation is specified by the modifier bits in the command code, and the function is performed over the I/O interface as an immediate operation (see "Immediate Operations"). If the command code does not specify the entire control function, the data-address field of the CCW designates the location containing the required additional information. This control information may include an order code further specifying the operation to be performed or an address, such as the disk address for the seek function, and is transferred in response to requests by the device.

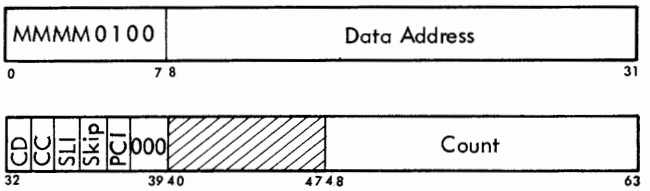
A control command code containing zeros for the six modifier bits is defined as no operation. The no-operation order causes the addressed device to respond with channel end and device end without causing any action at the device. The order can be executed as an immediate operation, or the device can delay the status until after the initiation sequence is completed. Other operations that can be initiated by means of the control command depend on the type of I/O device. These operations and their codes are specified in the SRL publication for the device.

A CCW used in a control operation is inspected for the CD, CC, SLI, and the PCI flags. The setting of the skip flag is ignored. Bit positions 0-5 of the CCW contain modifier bits.

Programming Note

Since a CCW with a count of zero is invalid, the program cannot use the CCW count field to specify that no data be transferred to the I/O device. Any operation terminated before data have been transferred causes the incorrect-length indication, provided the operation is not immediate and has not been rejected during the initiation sequence. The incorrect-length indication is suppressed when the SLI flag is on.

Sense



A sense operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to main storage. The data are placed in storage in an ascending order of addresses, starting with the address specified in the CCW.

Data transferred during a sense operation provide information concerning both unusual conditions detected in the last operation and the status of the device. The status information provided by the sense command is more detailed than that supplied by the unit-status byte and may describe reasons for the unit-check indication. It may also indicate, for example, if the device is in the not-ready state, if the tape unit is in the file-protected state, or if magnetic tape is positioned beyond the end-of-tape mark.

For most devices, the first six bits of the sense data describe conditions detected during the last operation. These bits are common to all devices having this type of information and are designated as follows:

BIT	DESIGNATION
0	Command reject
1	Intervention required
2	Bus-out check
3	Equipment check
4	Data check
5	Overrun

The following is the meaning of the first six bits:

Command Reject: The device has detected a programming error. A command has been received which the device is not designed to execute, such as read backward issued to a direct-access storage device, or which the device cannot execute because of its present state, such as write issued to a file-protected tape unit. Command reject is also indicated when the program issues an invalid sequence of commands, such as write to a direct-access storage device without previously designating the data block.

Intervention Required: The last operation could not be executed because of a condition requiring some type of intervention at the device. This bit indicates conditions such as an empty hopper in a card punch or the printer being out of paper. It is also turned on when the addressed device is in the not-ready state, is in test mode, or is not provided on the control unit.

Bus-Out Check: The device or the control unit has received a data byte or a command code with an invalid parity over the I/O interface. During writing, bus-out check indicates that incorrect data have been recorded at the device, but the condition does not cause the operation to be terminated prematurely. Parity errors on command codes and control information cause the operation to be immediately terminated and suppresses checking for command-reject and intervention-required conditions.

Equipment Check: During the last operation, the device or the control unit has detected equipment malfunctioning, such as an invalid card hole count or printer buffer parity error.

Data Check: The device or the control unit has detected a data error other than those included in bus-out check. Data check identifies errors associated with the recording medium and includes conditions such as reading an invalid card code or detecting invalid parity on data recorded on magnetic tape.

On an input operation, data check indicates that incorrect data may have been placed in main storage. The control unit forces correct parity on data sent to the channel. On writing, this condition indicates that incorrect data may have been recorded at the device. Unless the operation is of a type where the error precludes meaningful continuation, data errors on reading and writing do not cause the operation to be terminated prematurely.

Overrun: The channel has failed to respond on time to a request for service from the device. Overrun can occur when data are transferred to or from a non-buffered control unit operating with a synchronous medium, and the total activity initiated by the program exceeds the capability of the channel. When the channel fails to accept a byte on an input operation, the following data in main storage are shifted to fill the gap. On an output operation, overrun indicates that data recorded at the device may be invalid. The overrun bit is also turned on when the device receives the new command too late during command chaining.

All information significant to the use of the device normally is provided in the first two bytes. Any bit positions following those used for programming information contain diagnostic information, which may extend to as many bytes as needed. The amount and the meaning of the status information are peculiar to the type of I/O device and are specified in the SRL publication for the device.

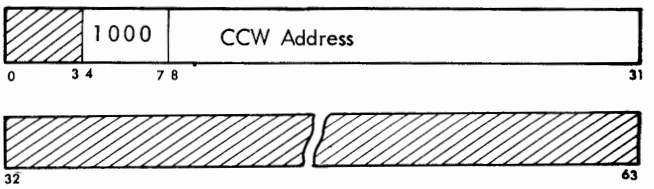
The basic sense command has zero modifier bits. This command initiates a sense operation on all devices and cannot cause the command-reject, intervention-required, data-check, or overrun bits to be turned on. If the control unit detects an equipment malfunction, or invalid parity of the sense command code, the equipment-check or bus-out-check bits are turned on, and unit check is sent with channel end.

Devices that can provide special diagnostic sense information or can be instructed to perform other special functions by use of the sense command, may define modifier bits for the control of these functions. The special sense operations may be initiated by a unique combination of modifier bits, or a group of codes may specify the same function. Any remaining sense command codes may be considered invalid, thus causing the unit-check indication, or may cause the same action as the basic sense command, depending upon the type of device.

The sense information pertaining to the last I/O operation is reset the next time the program causes the associated control unit to be selected, unless the selection is due to the execution of TEST I/O, or HALT I/O, or unless the basic sense operation, or a no-operation order is initiated at the control unit.

A CCW used in a sense operation is inspected for every one of the five flags — CD, CC, SLI, SKIP, and PCI. Bit positions 0-3 of the CCW contain modifier bits.

Transfer In Channel



The next CCW is fetched from the location designated by the data-address field of the CCW specifying transfer in channel. The transfer-in-channel command does not initiate any I/O operation at the channel, and the I/O device is not signaled of the execution of the command. The purpose of the transfer in channel command is to provide chaining between CCW's not located in adjacent double-word locations in an ascending order of addresses. The command can occur in both data and command chaining.

The first CCW designated by the CAW may not specify transfer in channel. When this restriction is violated, no I/O operation is initiated, and the program-check

condition is generated. The error causes the status portion of the CSW with the program-check indication to be stored during the execution of START I/O.

To address a CCW on integral boundaries for double words, a CCW specifying transfer in channel must contain zeros in bit positions 29-31. Furthermore, a CCW specifying a transfer in channel may not be fetched from a location designated by an immediately preceding transfer in channel. When either of these errors is detected or when an invalid address is specified in transfer in channel, the program-check condition is generated. When the transfer-in-channel command designates a CCW in a location protected for fetching, the protection-check condition is generated. Detection of these errors during data chaining causes the operation at the I/O device to be terminated, whereas during command chaining they cause an interruption condition to be generated.

The contents of the second half of the CCW, bit positions 32-63, are ignored. Similarly, the contents of bit positions 0-3 of the CCW are ignored.

Termination of Input/Output Operations

When the operation or sequence of operations initiated by START I/O is terminated, the channel and the device generate status conditions. These conditions can be brought to the attention of the program by means of an I/O interruption, by TEST I/O, or, in certain cases, by START I/O. The status conditions, as well as an address and a count indicating the extent of the operation sequence, are presented to the program in the form of a channel status word (CSW).

Types of Termination

Normally an I/O operation at the subchannel lasts until the device signals channel end. The channel-end condition can be signaled during the sequence initiating the operation, or later. When the channel detects equipment malfunctioning or a system reset is performed, the channel disconnects the device without receiving channel end. The program can force a device on the selector channel to be disconnected prematurely by issuing HALT I/O.

Termination at Operation Initiation

After the addressed channel and subchannel have been verified to be in a state where START I/O can be executed, certain tests are performed on the validity of the information specified by the program and on the availability of the addressed control unit and I/O device. This testing occurs both during the execution of START I/O and during command chaining.

A data-transfer operation is initiated at the subchannel and device only when no programming or equipment errors are detected by the channel and when the device responds with zero status during the initiation sequence. When the channel detects or the device signals any unusual condition during the initiation of an operation, but channel end is off, the command is said to be rejected.

Rejection of the command during the execution of START I/O is indicated by the setting of the condition code in the PSW. Unless the device is not operational, the conditions that precluded the initiation are detailed by the portion of the CSW stored by START I/O. The device is not started, no interruption conditions are generated, and the subchannel is not tied up beyond the initiation sequence. The device is immediately available for the initiation of another operation, provided the command was not rejected because of the busy or non-operational condition.

When an unusual condition causes a command to be rejected during initiation of an I/O operation by command chaining, an interruption condition is generated, and the subchannel is not available until the condition is cleared. The conditions are indicated to the program by means of the corresponding status bits in the CSW. The not-operational condition, which during the execution of START I/O causes condition code 3 to be set, is indicated by means of the interface-control-check bit. The new operation at the I/O devices is not started.

Immediate Operations

Instead of accepting or rejecting a command, the I/O device can signal the channel-end condition immediately upon receipt of the command code. An I/O operation causing the channel-end condition to be signaled during the initiation sequence is called an "immediate operation."

When the first CCW designated by the CAW initiates an immediate operation, no interruption condition is generated. If no command chaining occurs, the channel-end condition is brought to the attention of the program by causing START I/O to store the CSW status portion, and the subchannel is immediately made available to the program. The I/O operation, however, is initiated, and, if channel-end is not accompanied by device end, the device remains busy. Device end, when subsequently provided by the device, causes an interruption condition to be generated.

When command chaining is specified after an immediate operation and no unusual conditions have been detected during the execution, START I/O does not cause storing of CSW status. The subsequent commands in the chain are handled normally, and the channel-end condition for the last operation generates an interruption condition even if the device provides the signal immediately upon receipt of the command code.

Whenever immediate completion of an I/O operation is signaled, no data have been transferred to or from the device. The data address in the CCW is not checked for validity, except that it may not exceed the addressing capacity of the model.

Since a count of zero is not valid, any CCW specifying an immediate operation must contain a nonzero count. When an immediate operation is executed, however, incorrect length is not indicated to the program, and command chaining is performed when so specified.

Programming Note

Control operations for which the entire operation is specified in the command code may be executed as immediate operations. Whether the control function is executed as an immediate operation depends on the operation and type of device and is specified in the SRL publication for the device.

Termination of Data Transfer

When the device accepts a command, the subchannel is set up for data transfer. The subchannel is said to be working during this period. Unless the channel detects equipment malfunctioning or, on the selector channel, the operation is terminated by HALT I/O, the working state lasts until the channel receives the channel-end signal from the device. When no command chaining is specified or when chaining is suppressed because of unusual conditions, the channel-end condition causes the operation at the subchannel to be terminated and an interruption condition to be generated. The status bits in the associated CSW indicate channel end and the unusual conditions, if any. The device can signal channel end at any time after initiation of the operation, and the signal may occur before any data have been transferred.

For operations not involving data transfer, the device normally controls the timing of the channel-end condition. The duration of data transfer operations may be variable and may be controlled by the device or the channel.

Excluding equipment errors and HALT I/O, the channel signals the device to terminate data transfer whenever any of the following conditions occurs:

The storage areas specified for the operation are exhausted or filled.

Program-check condition is detected.

Protection-check condition is detected.

Chaining-check condition is detected.

The first of these conditions occurs when the channel has stepped the count in the last CCW associated with the operation to zero. A count of zero indicates that the channel has transferred all information specified by the program. The other three conditions are due to errors and cause premature termination of data transfer. In either case, the termination is signaled in response to a service request from the device and causes data transfer to cease. If the device has no blocks defined for the operation (such as writing on magnetic tape), it terminates the operation and generates the channel-end condition.

The device can control the duration of an operation and the timing of channel end by blocking of data. On certain operations for which blocks are defined (such as reading on magnetic tape), the device does not provide the channel-end signal until the end of the block is reached, regardless of whether or not the device has been previously signaled to terminate data transfer.

The channel suppresses initiation of an I/O operation when the data address in the first CCW associated with the operation exceeds the addressing capacity of the model. Complete check for the validity of the data address is performed only as data are transferred to or from main storage. When the initial data address in the CCW is invalid, no data are transferred during the operation, and the device is signaled to terminate the operation in response to the first service request. On writing, devices such as magnetic tape units request the first byte of data before any mechanical motion is started and, if the initial data address is invalid, the operation is terminated before the recording medium has been advanced. However, since the operation has been initiated, the device provides channel end, and an interruption condition is generated. Whether a block at the device is advanced when no data are transferred depends on the type of device and is specified in the SRL publication for the device.

When command chaining takes place, the subchannel appears in the working state from the time the first operation is initiated until the device signals the channel-end condition of the last operation of the chain. On the selector channel, the device executing the operation stays connected to the channel and the whole channel appears to be in the working state for the duration of the execution of the chain of operations. On the multiplexor channel an operation in the burst mode causes the channel to appear to be in the working state only for the duration of the transfer of the burst of data. If channel end and device end do not occur concurrently, the device disconnects from the channel after providing channel end, and the channel can in the meantime communicate with other devices on the interface.

Any unusual conditions cause command chaining to be suppressed and an interruption condition to be generated. The unusual conditions can be detected by

either the channel or the device, and the device can provide the indications with channel end, control unit end, or device end. When the channel is aware of the unusual condition by the time the channel-end signal for the operation is received, the chain is terminated as if the operation during which the condition occurred were the last operation of the chain. The device-end signal subsequently is processed as an interruption condition. When the device signals unit check or unit exception with control unit end or device end, the subchannel terminates the working state upon receipt of the signal from the device. The channel-end indication in this case is not made available to the program.

Termination by HALT I/O

The instruction HALT I/O causes the current operation at the addressed channel or subchannel to be terminated immediately. The method of termination differs from that used upon exhaustion of count or upon detection of programming errors to the extent that termination by HALT I/O is not contingent on the receipt of a service request from the device.

When HALT I/O is issued to a channel operating in the burst mode, the channel issues the halt-I/O signal to the device regardless of the current activity in the channel and on the interface. If the channel is involved in the data-transfer portion of an operation, data transfer is immediately terminated, and the device is disconnected from the channel. If HALT I/O is addressed to a selector channel executing a chain of operations and the device has already provided channel end for the current operation, the instruction causes the device to be disconnected and the chain-command flag to be removed.

When HALT I/O is issued to the multiplexor channel and the channel is not operating in the burst mode, HALT I/O causes the device to be selected, and the halt-I/O signal is issued as the device responds. When command chaining is indicated in the subchannel, HALT I/O causes the chain-command flag to be turned off.

Termination of an operation by HALT I/O on the selector channel results in up to four distinct interruption conditions. The first one is generated by the channel upon execution of the instruction and is not contingent on the receipt of status from the device. The command address and count in the associated CSW indicate how much data have been transferred, and the channel-status bits reflect the unusual conditions, if any, detected during the operation. If HALT I/O is issued before all data specified for the operation have been transferred, incorrect length is indicated, subject to the control of the SLI flag in the current CCW. The execution of HALT I/O itself is not reflected in CSW status, and all status bits in a CSW due to this interruption condition can be zero. The channel is available for the initiation of a new I/O operation as soon as the interruption condition is cleared.

The second interruption condition on the selector channel occurs when the control unit generates the channel-end condition. The selector channel handles this condition as any other interruption condition from the device after the device has been disconnected from the channel, and provides zeros in the protection key, command address, count, and channel status fields of the associated CSW. The channel-end condition is not made available to the program when HALT I/O is issued to a channel executing a chain of operations and the device has already provided channel end for the current operation.

Finally, the third and fourth interruption conditions occur when control unit end, if any, and device end are generated. These conditions are handled as for any other I/O operation.

Termination of an operation by HALT I/O on the multiplexor channel causes the normal interruption conditions to be generated. If the instruction is issued when the subchannel is in the data-transfer portion of an operation, the subchannel remains in the working state until channel end is signaled by the device, at which time the subchannel is placed in the interruption-pending state. If HALT I/O is issued after the device has signaled channel end and the subchannel is executing a chain of operations, the channel-end condition is not made available to the program, and the subchannel remains in the working state until the next status byte from the device is received. Receipt of a status byte subsequently places the subchannel in the interruption-pending state.

The CSW associated with the interruption condition in the subchannel contains the status bytes provided by the device and the channel, and indicates at what point data transfer was terminated. If HALT I/O is issued before all data areas associated with the current operation have been exhausted or filled, incorrect length is indicated, subject to the control of the SLI flag in the current CCW. The interruption condition is processed as for any other type of termination.

Programming Note

The CSW associated with a write operation terminated by HALT I/O indicates how many bytes the channel has sent to the device. Since the execution of HALT I/O may cause the loss of the byte of data in transit over the I/O interface and may cause the device to suppress recording of data contained in its buffer, if any, all

bytes that have left the channel may not necessarily be recorded at the I/O device.

Termination Due to Equipment Malfunction

When channel equipment malfunctioning is detected or invalid signals are received over the I/O interface, the recovery procedure and the subsequent states of the subchannels and devices on the channel depend on the type of error and on the model. Normally, the program is alerted of the termination by an I/O interruption, and the associated CSW indicates the channel-control-check or interface-control-check condition. In channels sharing common equipment with the CPU, malfunctioning detected by the channel may be indicated by a machine-check interruption, in which case no CSW is stored. Equipment malfunctioning may cause the channel to perform the malfunction-reset function.

Input/Output Interruptions

Input/output interruptions provide a means for the CPU to change its state in response to conditions that occur in I/O devices or channels. These conditions can be caused by the program or by an external event at the device.

Interruption Conditions

The conditions causing requests for I/O interruptions to be initiated are called I/O interruption conditions. An I/O interruption condition can be brought to the attention of the program only once and is cleared when it causes an interruption. Alternatively, an I/O interruption condition can be cleared by TEST I/O, and conditions generated by the I/O device following the termination of the operation at the subchannel can be cleared by START I/O. The latter include the attention, device-end, and control-unit-end conditions, and the channel-end condition when provided by a device on the selector channel after termination of the operation by HALT I/O.

The device attempts to initiate a request to the channel for an interruption whenever it detects any of the following conditions:

Channel end
Control-unit end
Device end
Attention

The device may also, at command chaining, have created an interruption condition at the device, which can be due to the following conditions:

Unit check
Unit exception
Busy indication from device
Program check
Protection check

When an operation initiated by command chaining is terminated because of an unusual condition detected during the command initiation sequence, the interruption condition may remain pending within the channel, or the channel may create an interruption condition at the device. An interruption condition is created at the device in response to presentation of status by the device and causes the device subsequently to present the same status for interruption purposes. The interruption condition at the device may or may not be associated with unit status. If the unusual condition is detected by the device (unit check, unit exception, or busy) the unit-status field of the associated CSW identifies the condition. In the case of program and protection check, the identification of the error condition is preserved in the subchannel, and appears in the channel-status field of the associated CSW. If the associated interruption condition has been queued at the device, the device provides zero status for interruption purposes. When command chaining takes place, channel end and device end do not cause an interruption, and are not made available.

An interruption condition caused by the device may be accompanied by channel and other unit status conditions. Furthermore, more than one interruption condition associated with the same device can be cleared at the same time. As an example, when the channel-end condition is not cleared at the device by the time device end is generated, both conditions may be indicated in the CSW and cleared at the device concurrently.

However, at the time the channel assigns highest priority for interruptions to a condition associated with an operation at the subchannel, the channel accepts the status from the device and clears the condition at the device. The interruption condition and the associated status indication are subsequently preserved in the subchannel. Any subsequent status generated by the device is not included with the condition at the subchannel, even if the status is generated before the CPU accepts the condition.

When the channel detects any of the following conditions, it initiates a request for an I/O interruption without communicating and without having received the status byte from the device:

PCI Flag in a CCW
Execution of HALT I/O on a selector channel

The interruption conditons from the channel can be accompanied by other channel status indications, but none of the device status bits is on when the channel initiates the interruption.

The method of processing a request for interruption due to equipment malfunctioning depends on the model. In channels sharing common equipment with

the CPU, malfunctioning detected by the channel may be indicated by causing a machine-check interruption.

Priority of Interruptions

All requests for I/O interruption are asynchronous to the activity in the CPU, and interruption conditions associated with more than one I/O device can exist at the same time. The priority among requests is controlled by two types of mechanisms — one establishes the priority among interruption conditions associated with devices attached to the same channel, and another establishes priority among requests from different channels. A channel requests an I/O interruption only after it has established priority among requests from its devices. The conditions responsible for the requests are preserved in the devices or channels until accepted by the CPU.

Assignment of priority to requests for interruption associated with devices on any one channel is a function of the type of interruption condition and the position of the device on the I/O interface cable. A device's position on the cable is not related to its address.

The selector channel assigns the highest priority to conditions associated with the portion of the operation in which the channel is involved. These conditions include channel end, program-controlled interruption, execution of HALT I/O in the channel, and errors prematurely terminating a chain of operations. The selector channel cannot handle any interruption conditions other than those due to the PCI flag while operation is in progress.

As soon as the selector channel has cleared the interruption conditions associated with data transfer, it starts monitoring devices for attention, control-unit-end, and device-end conditions and for the channel-end condition associated with operations terminated by HALT I/O. The highest priority is assigned to the I/O device that first identifies itself on the interface.

On the multiplexor channel the priority among requests for interruption is based only on response from devices. The highest priority is assigned to the device that first identifies itself with an interruption condition or that requests service for data transfer and contains the PCI condition in the subchannel. The PCI, as well as any other condition in the subchannel, cannot cause an I/O interruption unless the device initiates a reference to the subchannel.

Except for conditions associated with termination of data transfer, the current assignment of priority for interruption among devices on a channel may be canceled when START I/O, TEST I/O, or HALT I/O is issued to the channel. Whenever the assignment is canceled, the channel resumes monitoring for interruption conditions and reassigns the priority on completion of the activity associated with the I/O instruction.

The assignment of priority among requests for interruption from channels is based on the type of channel and its address assignment. The priorities of channels 1-6 are in the order of their addresses, with channel 1 having the highest priority. The interruption priority of multiplexor channel 0 is not fixed, and depends on the model and on the current activity in the channel. Its priority may be above, below, or between those of channels 1-6.

Interruption Action

An I/O interruption can occur only when the channel accommodating the device is not masked and after the execution of the current instruction in the CPU has been terminated. If a channel has established the priority among requests for interruption from devices while it is masked, the interruption occurs immediately after the termination of the instruction removing the mask and before the next instruction is executed. This interruption is associated with the highest priority condition on the channel. If more than one channel is unmasked concurrently, the interruption occurs from the channel having the highest priority among those requesting interruption.

If the priority among interruption conditions has not yet been established in the channel by the time the mask is removed, the interruption does not necessarily occur immediately after the termination of the instruction removing the mask. This delay can occur regardless of how long the interruption condition has existed in the device or the subchannel.

The interruption causes the current program status word (PSW) to be stored as the old PSW at location 56 and causes the CSW associated with the interruption to be stored at location 64. Subsequently, a new PSW is loaded from location 120, and processing resumes in the state indicated by this PSW. The I/O device or, in the case of control unit end, the control unit causing the interruption is identified by the channel address in bit positions 16-23 and by the device address in bit positions 24-31 of the old PSW. The CSW associated with the interruption identifies the condition responsible for the interruption and provides further details about the progress of the operation and the status of the device.

Programming Note

When a number of I/O devices on a shared control unit are concurrently executing operations such as rewinding tape or positioning a disk access mechanism, the initial device-end signals generated on completion of the operations are provided in the order of generation,

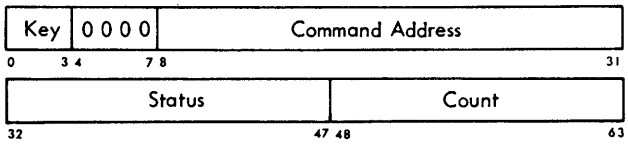
unless command chaining is specified for the operation last initiated. In the latter case, the control unit provides the device-end signal for the last initiated operation first, and the other signals are delayed until the subchannel is freed. Whenever interruptions due to the device-end signals are delayed either because the channel is masked or the subchannel is busy, the original order of the signals is destroyed.

Channel Status Word

The channel status word (CSW) provides to the program the status of an I/O device or the conditions under which an I/O operation has been terminated. The CSW is formed, or parts of it are replaced, in the process of I/O interruptions and during execution of START I/O, TEST I/O, and HALT I/O. The CSW is placed in main storage at location 64 and is available to the program at this location until the time the next I/O interruption occurs or until another I/O instruction causes its content to be replaced, whichever occurs first.

When the CSW is stored as a result of an I/O interruption, the I/O device is identified by the I/O address in the old PSW. The information placed in the CSW by START I/O, TEST I/O, or HALT I/O pertains to the device addressed by the instruction.

The CSW has the following format:



The fields in the CSW are allocated for the following purposes:

Protection Key: Bits 0-3 form the protection key used in the chain of operations at the subchannel.

Command Address: Bits 8-31 form an address that is eight higher than the address of the last CCW used.

Status: Bits 32-47 identify the conditions in the device and the channel that caused the storing of the CSW. Bits 32-39 are obtained over the I/O interface and indicate conditions detected by the device or the control unit. Bits 40-47 are provided by the channel and indicate conditions associated with the subchannel. Each of the 16 bits represents one type of condition, as follows:

BIT	DESIGNATION	BIT	DESIGNATION
32	Attention	40	Program-controlled interruption
33	Status modifier	41	Incorrect length
34	Control unit end	42	Program check
35	Busy	43	Protection check
36	Channel end	44	Channel data check
37	Device end	45	Channel control check
38	Unit check	46	Interface control check
39	Unit exception	47	Chaining check

Count: Bits 48-63 form the residual count for the last CCW used.

Unit Status Conditions

The following conditions are detected by the I/O device or control unit and are indicated to the channel over the I/O interface. The timing and causes of these conditions for each type of device are specified in the SRL publication for the device.

When the I/O device is accessible from more than one channel, status due to channel-initiated operations is signaled to the channel that initiated the associated I/O operation. The handling of conditions not associated with I/O operations, such as attention or device end due to transition from the not-ready to the ready state, depends on the type of device and condition and is specified in the SRL publication for the device.

The channel does not modify the status bits received from the I/O device. These bits appear in the CSW as received over the interface.

Attention

Attention is generated when the device detects an asynchronous condition that is significant to the program. The condition is interpreted by the program and is not associated with the initiation, execution, or termination of an I/O operation.

The device can signal the attention condition to the channel only when no operation is in progress at the I/O device, control unit, or subchannel. Attention can be indicated with device end upon completion of an operation, and it can be presented to the channel during the initiation of a new I/O operation. Otherwise, the handling and presentation of the condition to the channel depends on the type of device.

When the device signals attention during the initiation of an operation, the operation is not initiated. Attention accompanying device end causes command chaining to be suppressed.

Status Modifier

Status modifier is generated by the device when the device cannot provide its current status in response to TEST I/O, to indicate that the control unit is busy, or when the normal sequence of commands has to be modified.

When the status-modifier condition is signaled in response to TEST I/O and the bit appears in the CSW in the absence of any other status bit, presence of the bit indicates that the device cannot execute the instruction and has not provided its current status. The interruption condition, which may be pending at the device or subchannel, has not been cleared, and the CSW stored by TEST I/O contains zeros in the key, command ad-

dress, and count fields. The 2702 Transmission Control is an example of a type of device that cannot execute TEST I/O.

When the status-modifier bit appears in the CSW together with the busy bit, it indicates that the busy condition pertains to the control unit associated with the addressed I/O device. The control unit appears busy when it is executing a type of operation that precludes the acceptance and execution of any command or the instructions TEST I/O and HALT I/O or contains an interruption condition for a device other than one addressed. The interruption condition may be due to control unit end or, on the selector channel, due to channel end following the execution of HALT I/O. The busy state occurs for operations such as backspace tape file, in which case the control unit remains busy after providing channel end, for operations terminated on the selector channel by HALT I/O, and temporarily occurs on the 2702 Transmission Control after initiation of an operation on a device accommodated by the control unit. A control unit accessible from two or more channels appears busy when it is communicating with another channel.

Presence of the status modifier and device end means that the normal sequence of commands must be modified. The handling of this set of bits by the channel depends on the operation. If command chaining is specified in the current CCW and no unusual conditions have been detected, presence of status modifier and device end causes the channel to fetch and chain to the CCW whose main-storage address is 16 higher than that of the current CCW. If the I/O device signals the status-modifier condition at a time when no command chaining is specified, or when any unusual conditions have been detected, no action is taken in the channel, and the status-modifier bit is placed in the CSW.

Programming Note

When the multiplexor channel detects a programming error during command chaining, the interruption condition is queued at the I/O device. On devices such as the 2702 Transmission Control, queuing of the condition may generate the status-modifier indication, which subsequently appears in the CSW associated with the termination of the operation.

Control Unit End

Control unit end indicates that the control unit has become available for use for another operation.

The control-unit-end condition is provided only by control units shared by I/O devices and only when one or both of the following conditions has occurred:

1. The program had previously caused the control unit to be interrogated while the control unit was in the busy state. The control unit is considered to have been interrogated in the busy state when a command or the instructions TEST I/O or HALT I/O had been issued to a device on the control unit, and the control unit had responded with busy and status modifier in the unit status byte. See "Status Modifier."

2. The control unit detected an unusual condition during the portion of the operation after channel end had been signaled to the channel. The indication of the unusual condition accompanies control unit end.

If the control unit remains busy with the execution of an operation after signaling channel end but has not been interrogated by the program, control unit end is not generated. Similarly, control unit end is not provided when the control unit has been interrogated and could perform the indicated function. The latter case is indicated by the absence of busy and status modifier in the response to the instruction causing the interrogation.

When the busy state of the control unit is temporary, control unit end is included with busy and status modifier in response to the interrogation even though the control unit has not yet been freed. The busy condition is considered to be temporary if its duration is commensurate with the program time required to handle an I/O interruption. The 2702 Transmission Control is an example of a device in which the control unit may be busy temporarily and which includes control unit end with busy and status modifier.

The control-unit-end condition can be signaled with channel end, device end, or between the two. When control unit end is signaled by means of an I/O interruption in the absence of any other status conditions, the interruption may be identified by any address assigned to the control unit. A pending control unit end causes the control unit to appear busy for initiation of new operations.

Busy

Busy indicates that the I/O device or control unit cannot execute the command or instruction because it is executing a previously initiated operation or because it contains a pending interruption condition. The interruption condition for the addressed device, if any, accompanies the busy indication. If the busy condition applies to the control unit, busy is accompanied by status modifier.

The following table lists the conditions when the busy bit (B) appears in the CSW and when it is accompanied by the status-modifier bit (SM). A double hyphen (--) indicates that the busy bit is off; an asterisk (*) indicates that CSW status is not stored or an I/O interruption cannot occur; and the (cl) indicates that the interruption condition is cleared and

the status appears in the CSW. The abbreviation DE stands for device end, while CU stands for control unit.

CONDITION	CSW STATUS STORED BY: START I/O	TEST I/O	HALT I/O	I/O INT.
Subchannel available				
DE or attention in device	B,cl	--,cl	*	--,cl
Device working, CU available	B	B	*	*
CU end or channel end in CU:				
for the addressed device	B,cl	--,cl	*	--,cl
for another device	B,SM	B,SM	*	--,cl
CU working	B,SM	B,SM	*	*
Interruption pend. in subchannel				
for the addressed device because of:				
chaining terminated by busy condition	*	B,cl	*	B,cl
other type of termination	*	--,cl	*	--,cl
Subchannel working				
CU available	*	*	--	*
CU working	*	*	B,SM	*

Channel End

Channel end is caused by the completion of the portion of an I/O operation involving transfer of data or control information between the I/O device and the channel. The condition indicates that the subchannel has become available for use for another operation.

Each I/O operation causes a channel-end condition to be generated, and there is only one channel end for an operation. The channel-end condition is not generated when programming errors or equipment malfunctions are detected during initiation of the operation. When command chaining takes place, only the channel end of the last operation of the chain is made available to the program. The channel-end condition is not made available to the program when a chain of commands is prematurely terminated because of an unusual condition indicated with control unit end or device end or during the initiation of a chained command.

The instant within an I/O operation when channel end is generated depends on the operation and the type of device. For operations such as writing on magnetic tape, the channel-end condition occurs when the block has been written. On devices that verify the writing, channel end may or may not be delayed until verification is performed, depending on the device. When magnetic tape is being read, the channel-end condition occurs when the gap on tape reaches the read-write head. On devices equipped with buffers, such as the IBM 1443 N1 Printer (bar line printer), the channel-end condition occurs upon completion of data transfer between the channel and the buffer. During control operations, channel end is generated when the control information has been transferred to the devices, although for short operations the condition may be delayed until completion of the operation. Operations that do not cause any data to be transferred can provide the channel-end condition during the initiation sequence.

A channel-end condition pending in the control unit causes the control unit to appear busy for initiation of new operations. Unless the operation has been performed on the selector channel and has been terminated by HALT I/O, a pending channel end causes the subchannel to be in the interruption-pending state.

Device End

Device end is caused by the completion of an I/O operation at the device or, on some devices, by manually changing the device from the not-ready to the ready state. The condition normally indicates that the I/O device has become available for use for another operation.

Each I/O operation causes a device-end condition, and there is only one device-end to an operation. The device-end condition is not generated when any programming or equipment malfunction is detected during initiation of the operation. When command chaining takes place, only the device-end of the last operation of the chain is made available to the program unless an unusual condition is detected during the initiation of a chained command, in which case the chain is terminated without the device-end indication.

The device-end condition associated with an I/O operation is generated either simultaneously with the channel-end condition or later. On data transfer operations on devices such as magnetic tape units, the device terminates the operation at the time channel end is generated, and both device end and channel end occur together. On buffered devices, such as an IBM 1443 Printer, the device-end condition occurs upon completion of the mechanical operation. For control operations, device end is generated at the completion of the operation at the device. The operation may be completed at the time channel end is generated or later.

When command chaining is specified in the subchannel, receipt of the device-end signal, in the absence of any unusual conditions, causes the channel to initiate a new I/O operation.

Unit Check

Unit check indicates that the I/O device or control unit has detected an unusual condition that is detailed by the information available to a sense command. Unit check may indicate that a programming or an equipment error has been detected, that the not-ready state of the device has affected the execution of the command or instruction, or that an exceptional condition

other than the one identified by unit exception has occurred. The unit-check bit provides a summary indication of the conditions identified by sense data.

An error condition causes the unit-check indication only when it occurs during the execution of a command or TEST I/O, or during some activity associated with an I/O operation. Unless the error condition pertains to the activity initiated by a command and is of immediate significance to the program, the condition does not cause the program to be alerted after device end has been cleared; a malfunction may, however, cause the device to become not ready.

Unit check is indicated when the existence of the not-ready state precludes a satisfactory execution of the command, or when the command, by its nature, tests the state of the device. When no interruption condition is pending for the addressed device at the control unit, the control unit signals unit check when TEST I/O or the no-operation control command is issued to a not-ready device. In the case of no operation, the command is rejected, and channel end and device end do not accompany unit check.

Unless the command is designed to cause unit check, such as rewind and unload on magnetic tape, unit check is not indicated if the command is properly executed even though the device has become not ready during or as a result of the operation. Similarly, unit check is not indicated if the command can be executed with the device not ready. The IBM 2150 Console, for example, accepts and executes the alarm control command when the printer is not ready. Selection of a device in the not-ready state does not cause a unit-check indication when the sense command is issued, and whenever an interruption condition is pending for the addressed device at the control unit.

If the device detects during the initiation sequence that the command cannot be executed, unit check is presented to the channel and appears without channel end, control unit end, or device end. Such unit status indicates that no action has been taken at the device in response to the command. If the condition precluding proper execution of the operation occurs after execution has been started, unit check is accompanied by channel end, control unit end, or device end, depending on when the condition was detected. Any errors associated with an operation, but detected after device end has been cleared, are indicated by signaling unit check with attention.

Errors, such as invalid command code or invalid command code parity, do not cause unit check when the device is working or contains a pending interruption condition at the time of selection. Under these circumstances, the device responds by providing the busy bit and indicating the pending interruption condition, if any. The command code invalidity is not indicated.

Termination of an operation with the unit-check indication causes command chaining to be suppressed.

Programming Note

If a device becomes not ready upon completion of a command, the ending interruption condition can be cleared by TEST I/O without generation of unit check due to the not-ready state, but any subsequent TEST I/O issued to the device causes a unit-check indication.

Unit Exception

Unit exception is caused when the I/O device detects a condition that usually does not occur. Unit exception includes conditions such as recognition of a tape mark and does not necessarily indicate an error. It has only one meaning for any particular command and type of device.

The unit-exception condition can be generated only when the device is executing an I/O operation, or when the device involved with some activity associated with an I/O operation and the condition is of immediate significance to the program. If the device detects during the initiation sequence that the operation cannot be executed, unit exception is presented to the channel and appears without channel end, control unit end, or device end. Such unit status indicates that no action has been taken at the device in response to the command. If the condition precluding normal execution of the operation occurs after the execution has been started, unit exception is accompanied by channel end, control unit end, or device end, depending on when the condition was detected. Any unusual conditions associated with an operation, but detected after device-end has been cleared; is indicated by signaling unit exception with attention.

A command does not cause unit exception when the device responds during the initial selection with busy status to the command.

Termination of an operation with the unit-exception indication causes command chaining to be suppressed.

Channel Status Conditions

The following conditions are detected and indicated by the channel. Except for the conditions caused by equipment malfunctioning, they can occur only while the subchannel is involved with the execution of an I/O operation.

Program-Controlled Interruption

The program-controlled-interruption condition is generated when the channel fetches a CCW with the program-controlled-interruption (PCI) flag on. The interruption due to the PCI flag takes place as soon as

possible after the CCW takes control of the operation but may be delayed an unpredictable amount of time because of masking of the channel or other activity in the system.

Detection of the PCI condition does not affect the progress of the I/O operation.

Incorrect Length

Incorrect length occurs when the number of bytes contained in the storage areas assigned for the I/O operation is not equal to the number of bytes requested or offered by the I/O device. Incorrect length is indicated for one of the following reasons:

Long Block on Input: During a read, read-backward, or sense operation, the device attempted to transfer one or more bytes to storage after the assigned storage areas were filled. The extra bytes have not been placed in main storage. The count in the CSW is zero.

Long Block on Output: During a write or control operation the device requested one or more bytes from the channel after the assigned main-storage areas were exhausted. The count in the CSW is zero.

Short Block on Input: The number of bytes transferred during a read, read backward, or sense operation is insufficient to fill the storage areas assigned to the operation. The count in the CSW is not zero.

Short Block on Output: The device terminated a write or control operation before all information contained in the assigned storage areas was transferred to the device. The count in the CSW is not zero.

The incorrect-length indication is suppressed when the current CCW has the SLI flag and does not have the CD flag. The indication does not occur for immediate operations and for operations rejected during the initiation sequence.

Presence of the incorrect-length condition suppresses command chaining unless the SLI flag in the CCW is on or unless the condition occurs in an immediate operation. See the table in the Chaining section of this manual for the effect of the CD, CC, and SLI flags on the indication of incorrect length.

Program Check

Program check occurs when programming errors are detected by the channel. The condition can be due to the following causes:

Invalid CCW Address Specification: The CAW or the transfer-in-channel command does not designate the CCW on integral boundaries for double words. The three low-order bits of the CCW address are not zero.

Invalid CCW Address: The channel has attempted to fetch a CCW from a location outside the main storage of the particular installation. An invalid CCW address can occur in the channel because the program has specified an invalid address in the CAW or in the transfer-in-channel command or because on chaining the channel has stepped the address above the highest available location.

Invalid Command Code: The command code in the first CCW designated by the CAW or in a CCW fetched on command chaining has four low-order zeros. The command code is not tested for validity during data chaining.

Invalid Count: A CCW other than a CCW specifying transfer in channel contains the value zero in bit positions 48-63.

Invalid Data Address: The channel has attempted to transfer data to or from a location outside the main storage of the particular installation. An invalid data address can occur in the channel because the program has specified an invalid address in the CCW or because the channel has stepped the address above the highest available address or, on reading backward, below zero.

Invalid Key: The CAW contains a nonzero storage protection key in a model not having the protection feature installed.

Invalid CAW Format: The CAW does not contain zeros in bit positions 4-7.

Invalid CCW Format: A CCW other than a CCW specifying transfer in channel does not contain zeros in bit positions 37-39.

Invalid Sequence: The first CCW designated by the CAW specifies transfer in channel or the channel has fetched two successive CCW's both of which specify transfer in channel.

Detection of the program-check condition during the initiation of an operation causes execution of the operation to be suppressed. When the condition is detected after the device has been started, the device is signaled to terminate the operation the next time it requests or offers a byte of data. The program-check condition causes command chaining to be suppressed.

Protection Check

Protection check occurs when the channel attempts to place data in or fetch data or a CCW from a portion of main storage that is protected for the current operation on the subchannel. The protection key associated with the I/O operation does not match the key of the addressed main-storage location, and the protection key is not zero.

When the protection-check condition occurs during the fetching of a CCW that specifies the initiation of an I/O operation, the operation is not initiated. When protection check is detected after the device has been started, the device is signaled to terminate the operation the next time it requests or offers a byte of data. The condition causes command chaining to be suppressed.

The protection-check condition can be generated only on models having the protection feature installed.

Channel Data Check

Channel data check indicates that the channel has detected a parity error in the information transferred to or from main storage during an I/O operation. This information includes the data read or written, as well as the information transferred as data during a sense or control operation. The error may have been detected anywhere inboard the I/O interface: in the channel, in main storage, or on the path between the two. Channel data check may be indicated for parity errors detected in data that are referred to by the channel but do not participate in the operation.

Whenever a parity error on I/O data is indicated by means of channel data check, the channel forces correct parity on all data received over the I/O interface and, within the limitations of parity checking and correction facilities of the model, correct parity is forced on all data placed in main storage. On an output operation, the parity of the data is not changed when channel data check is indicated.

A condition indicated as channel data check causes command chaining to be suppressed, but does not affect the execution of the current operation. Data transfer proceeds to normal completion, and an I/O interruption condition is generated when the device presents channel end. No log-out or reset occurs, and the detection of the error does not affect the state of the channel or device.

When CPU and channel equipment is integrated to such an extent that a data parity error precludes continuation of the I/O operation or handling of the parity bit as described above, a machine-check condition is generated upon the detection of the error. When a data parity error causes a machine-check interruption, reset and log-out may be performed, and the subsequent recovery procedure depends on the model.

Channel Control Check

Channel control check is caused by any machine malfunctioning affecting channel controls. The condition includes parity errors on CCW and data addresses and parity errors on the contents of the CCW. Conditions responsible for channel control check may cause the contents of the CSW to be invalid and conflicting. The CSW as generated by the channel has correct parity.

Detection of the channel-control-check condition causes the current operation, if any, to be immediately terminated and may cause the channel to perform the malfunction-reset function. The recovery procedure in the channel and the subsequent state of the subchannel upon a malfunction reset depend upon the model.

Interface Control Check

Interface control check is caused by any invalid signal on the I/O interface. The condition is detected by the channel and usually indicates malfunctioning of an I/O device. It can be due to the following reasons:

1. The address or status byte received from a device has invalid parity.
2. A device responded with an address other than the address specified by the channel during initiation of an operation.
3. During command chaining the device appeared not operational.
4. A signal from a device occurred at an invalid time or had invalid duration.

Detection of the interface-control-check condition causes the current operation, if any, to be immediately terminated and may cause the channel to perform the malfunction-reset function. The recovery procedure in the channel and the subsequent state of the subchannel upon a malfunction reset depends on the model.

Chaining Check

Chaining check is caused by channel overrun during data chaining on input operations. The condition occurs when the I/O data rate is too high for the particular resolution of data addresses. Chaining check cannot occur on output operations.

Detection of the chaining-check condition causes the I/O device to be signaled to terminate the operation. It causes command chaining to be suppressed.

Content of Channel Status Word

The content of the CSW depends on the condition causing the storing of the CSW and on the programming method by which the information is obtained. The status portion always identifies the condition that caused storing of the CSW. The protection key, command address, and count fields may contain information pertaining to the last operation or may be set to zero, or the original contents of these fields at location 64 may be left unchanged.

Information Provided by Channel Status Word

Conditions associated with the execution or termination of an operation at the subchannel cause the whole CSW to be replaced. Such a CSW can be stored only by an I/O interruption or by TEST I/O. Except for conditions associated with command chaining and equipment malfunctioning, the storing can be caused by the PCI or channel-end condition and by the execution of HALT I/O on the selector channel. The contents of the CSW are related to the current values of the corresponding quantities, although the count is unpredic-

table after program check, protection check, chaining check, and after an interruption due to the PCI flag.

A CSW stored upon the execution of a chain of operation pertains to the last operation the channel executed or attempted to initiate. Information concerning the preceding operations is not preserved and is not made available to the program.

When an unusual condition causes command chaining to be suppressed, the premature termination of the chain is not explicitly indicated in the CSW. A CSW associated with a termination due to a condition occurring at channel-end time contains the channel-end bit and identifies the unusual condition. When the device signals the unusual condition with control unit end or device end, the channel-end indication is not made available to the program, and the channel provides the current protection key, command address, and count, as well as the unusual indication, with the control-unit-end or device-end bit in the CSW. The command address and count fields pertain to the operation that was executed.

When the execution of a chain of commands is terminated by an unusual condition detected during initiation of a new operation, the command address and count fields pertain to the rejected command. Except for conditions caused by equipment malfunctioning, termination at the initiation time can occur because of attention, unit check, unit exception, or program check, and causes both the channel-end and device-end bits in the CSW to be turned off.

A CSW associated with conditions occurring after the operation at the subchannel has been terminated contains zeros in the protection key, command address, and count fields, provided the conditions are not cleared by START I/O. These conditions include attention, control unit end, and device end (and channel end when it occurs after termination of an operation on the selector channel by HALT I/O).

When the above conditions are cleared by START I/O, only the status portion of the CSW is stored, and the original contents of the protection key, command address, and count fields in location 64 are preserved. Similarly, only the status bits of the CSW are changed when the command is rejected or the operation at the subchannel is terminated during the execution of START I/O or whenever HALT I/O causes CSW status to be stored.

Errors detected during execution of the I/O operation do not affect the validity of the CSW unless the channel-control-check or interface-control-check conditions are indicated. Channel control check indicates that equipment errors have been detected, which can cause any part of the CSW, as well as the address in the PSW identifying the I/O device, to be invalid. Interface control check indicates that the address identifying the device or the status bits received from the device may be invalid. The channel forces correct parity on invalid CSW fields.

Protection Key

A CSW stored to reflect the progress of an operation at the subchannel contains the protection key used in that operation. The content of this field is not affected by programming errors detected by the channel or by the condition causing termination of the operation.

Models in which the protection feature is not implemented cause an all-zero key to be stored.

Command Address

When the CSW is formed to reflect the progress of the I/O operation at the subchannel, the command address is normally eight higher than the address of the last CCW used in the operation.

The following table lists the contents of the command address field for all conditions that can cause the CSW to be stored. The conditions are listed in order of priority; that is, if two conditions are indicated or occur, the CSW appears as indicated for the condition higher on the list. The programming errors listed in the table refer to conditions included in program check.

CONDITION	CONTENT
Channel control check	Unpredictable
Status stored by START I/O	Unchanged
Status stored by HALT I/O	Unchanged
Invalid CCW address spec in Transfer in channel (TIC)	Address of TIC + 8
Invalid CCW address in TIC	Address of TIC + 8
Invalid CCW address generated	First invalid CCW address +8
Invalid command code	Address of invalid CCW + 8
Invalid count	Address of invalid CCW + 8
Invalid data address	Address of invalid CCW + 8
Invalid CCW format	Address of invalid CCW + 8
Invalid sequence — 2 TIC's	Address of second TIC + 8
Protection check	Address of protected CCW + 8
Chaining check	Address of last-used CCW + 8
Termination under count control	Address of last-used CCW + 8
Termination by I/O device	Address of last-used CCW + 8
Termination by HALT I/O	Address of last-used CCW + 8
Suppression of command chaining due to unit check or unit exception with device end or control unit end	Address of last CCW used in the completed operation + 8
Termination on command chaining by busy, unit check, or unit exception	Address of CCW specifying the new operation + 8
PCI flag in CCW	Address of last-used CCW + 8
Interface control check	Address of last-used CCW + 8
Ch end after HIO on sel ch	Zero
Control unit end	Zero
Device end	Zero
Attention	Zero
Busy	Zero
Status modifier	Zero

Count

The residual count, in conjunction with the original count specified in the last CCW used, indicates the number of bytes transferred to or from the area designated by the CCW. When an input operation is terminated, the difference between the original count in the CCW and the residual count in the CSW is equal to the number of bytes transferred to main storage; on an output operation, the difference is equal to the number of bytes transferred to the I/O device.

The following table lists the contents of the count field for all conditions that can cause the CSW to be stored. The conditions are listed in the order of priority; that is, if two conditions are indicated or occur, the CSW appears as for the condition higher on the list.

CONDITION	CONTENT
Channel control check	Unpredictable
Status stored by START I/O	Unchanged
Status stored by HALT I/O	Unchanged
Program check	Unpredictable
Protection check	Unpredictable
Chaining check	Unpredictable
Termination under count control	Correct
Termination by I/O device	Correct
Termination by HALT I/O	Correct
Suppression of command chaining due to unit check or unit exception with device end or control unit end	Correct. Residual count of last CCW used in the completed operation.
Termination on command chaining by busy, unit check, or unit exception	Correct. Original count of CCW specifying the new operation.
PCI flag in CCW	Unpredictable
Interface control check	Correct
Ch end after HIO on sel ch	Zero
Control unit end	Zero
Device end	Zero
Attention	Zero
Busy	Zero
Status Modifier	Zero

Status

The status bits identify the conditions that have been detected during the I/O operation, that have caused a command to be rejected, or that have been generated by external events.

When the channel detects several error conditions, all conditions may be indicated or only one may appear in the CSW, depending on the condition and model. Conditions associated with equipment malfunctioning have precedence, and whenever malfunctioning causes an operation to be terminated, channel control check, interface control check, or channel data check is indicated, depending on the condition. When an operation is terminated by program check, protection check, or chaining check, the channel identifies the condition responsible for the termination and may or may not indicate incorrect length. When a data error has been detected and the operation is terminated prematurely because of a program check, protection check, or chaining check, both data check and the programming error are identified.

If the CCW fetched on command chaining contains the PCI flag but a programming error in the contents of the CCW or an unusual condition signaled by the device precludes the initiation of the operation, the PCI bit appears in the CSW associated with the interruption condition. Similarly, if device status or a programming error in the contents of the CCW causes the command to be rejected during execution of START I/O, the CSW stored by START I/O contains the PCI bit. However, when the channel detects a programming error in the CAW or in the first CCW, the PCI bit may unpredictably appear in a CSW stored by START I/O without the PCI flag being on in the first CCW associated with the START I/O.

Conditions detected by the channel are not related to those identified by the I/O device.

The following table summarizes the handling of status bits. The table lists the states and activities that can cause status indications to be created and the methods by which these indications can be placed in the CSW.

STATUS	WHEN I/O IS IDLE	WHEN SUBCHANNEL WORKING	UPON TERMINATION OF OPERATION AT SUBCHANNEL	AT CONTROL UNIT	AT DEVICE	DURING COMMAND CHAINING	BY START I/O	BY TEST I/O	BY HALT I/O	BY I/O INTER-RUPTION
Attention	C*				C	C*	S	S		S
Status modifier					C	C	C S	C S	C S	S
Control unit end				C*			C S	C S	C S	S
Busy						C	C S	C S	C S	S
Channel end			C*	C* H		C* †	C†S	S		S
Device end	C*				C*	C †	C†S	S		S
Unit check			C	C	C	C*	C S	C S		C S
Unit exception			C	C	C	C*	C S	S		S
Program-controlled interruption		C*	C			C	C S	S		S
Incorrect length		C	C					S		S
Program check		C	C			C*	C S	S		S
Protection check		C	C			C*	C S	S		S
Channel data check		C	C					S		S
Channel control check	C*	C*	C*	C*	C*	C*	C S	C S	C S	C S
Interface control check	C*	C*	C*	C*	C*	C*	C S	C S	C S	C S
Chaining check		C	C					S		S

NOTES

C—The channel or the device can create or present the status condition at the indicated time. A CSW or its status portion is not necessarily stored at this time.

Conditions such as channel end and device end are created at the indicated time. Other conditions may have been created previously, but are made accessible to the program only at the indicated time. Examples of such conditions are program check and channel data check, which are detected while data are transferred, but are made available to the program only with channel end, unless the PCI flag or equipment malfunctioning have caused an interruption condition to be generated earlier.

S—The status indication is stored in the CSW at the indicated time.

An S appearing alone indicates that the condition has been created previously. The letter C appearing with the S indicates that the status condition did not necessarily exist previously in the form that causes the program to be alerted, and may have been created by the I/O instruction or I/O interruption. For example, equipment malfunctioning may be detected during an I/O interruption, causing channel control check or interface control check to be indicated; or a device such as the 2702 Transmission Control Unit may signal the control-unit-busy condition in response to interrogation by an I/O instruction, causing status modifier, busy, and control unit end to be indicated in the CSW.

*—The status condition generates an interruption condition.

Channel end and device end do not result in interruption conditions when command chaining is specified and no unusual conditions have been detected.

†—This status indication can be created at the indicated time only by an immediate operation.

H—When an operation on the selector channel has been terminated by HALT I/O, channel end indicates the termination of the data-handling portion of the operation at the control unit.

Block-Multiplexing

The block-multiplexer channel, like the byte-multiplexer channel, can include both shared and unshared subchannels; but unlike the byte-multiplexer channel, which is optimized for relatively low-speed byte multiplexing, this channel is optimized for relatively high-speed burst operations and is designed to multiplex complete blocks of data. During multiplexing, two or more subchannel programs can be executed in interleaved fashion.

Multiplexing can take place on a block-multiplexer channel when the channel is operating in the block-multiplex mode and one of these two conditions exists:

1. The CCW calls for command chaining and channel end status indication is presented alone.
2. *Command retry* is signaled.

If the conditions described for multiplexing exist when the channel end indication is presented, the channel logically disconnects from the control unit (and disconnects from the subchannel program it has been executing), thus freeing the channel and the I/O interface. Information necessary to restart the disconnected subchannel program is saved in the subchannel. The channel is then available to execute a channel program for a different subchannel, even though the execution of the channel program from which it is disconnected is not complete. When the logically disconnected device signals that it is again ready to use the channel and interface (by presenting the device end indication or device end and status modifier indications), and if the channel and interface are available, the information previously saved is again made active, and execution of the disconnected subchannel program is resumed at the appropriate CCW. If the channel and interface are not available, the device must wait until they are. If, upon reconnection, any status indications are presented other than device end or device end and status modifier, the subchannel program execution is terminated, and an I/O interruption condition is created.

If channel end and device end indications are presented together, no multiplexing takes place and normal command chaining, if called for, is performed. However, if any other status indications are presented along with channel end and device end (except for status modifier when command chaining is called for by the CCW, or status modifier and unit check when it is not), the program execution is terminated, and an I/O interruption condition is created.

During multiplexing on a block-multiplexer channel, the length of the segment of data transferred across the I/O interface is defined by the length specified for the operation in the CCW string being executed, or by the length of the block of data at the I/O device, whichever is shorter. This contrasts with multiplexing on a byte-multiplexer channel, where the data for an operation may be transferred across the I/O interface in a number of relatively small, separate segments as short as a single byte.

Forcing Termination of I/O Operations

A device operating on a channel in burst mode can be disconnected prematurely by issuing a HALT I/O instruction; this instruction causes termination of the data transfer and disconnection from the channel of the device performing the burst operation. In such a case, the subchannel and I/O device addresses specified in the HALT I/O instruction are ignored. On a block-multiplexer channel, which allows the interleaved execution of burst operations for several different devices, the execution of HALT I/O possibly may not stop data transfer across the I/O interface for the addressed device; instead, it may stop data transfer for an unspecified, unidentified device. To ensure that data transfer for the addressed device, and only the addressed device, is terminated, the HALT DEVICE instruction may be used.

Halt Device

HDV $D_1(B_1)$ [*SI*]

The current I/O operation at the addressed I/O device is terminated. The subsequent state of the subchannel depends on the type of channel. HALT DEVICE is executed only when the CPU is in the supervisor state.

Bit positions 16-31 of the sum formed by adding the contents of register B_1 and the contents of the D_1 field identify the channel, the subchannel, and I/O device to which the instruction applies.

When the channel is either available or in the interruption pending state, and the subchannel is available or working with the addressed device, HALT DEVICE causes the addressed device to be selected and to be signaled to terminate the current operation, if any. If the subchannel is working with the addressed device, HALT DEVICE also causes the subchannel to be set up to signal termination of the device operation the next time the device requests or offers a byte of data, if any. Chaining, if indicated in the subchannel, is suppressed. If the subchannel is available, the state of the subchannel is not affected.

When the channel is either available or in the interruption pending state, and the subchannel is either working with a device other than the one addressed or in the interruption pending state, no action is taken.

When the channel is working in burst mode with the addressed device, data transfer across the I/O interface for the operation is immediately terminated, and the device immediately disconnects from the channel. Chaining, if indicated in the subchannel, is suppressed.

When the channel is working in burst mode with a device other than the one addressed, and the subchannel is available, is in the interruption pending state, or is working with a device other than the one addressed, no action is taken. If the subchannel is working with the addressed device, the subchannel is set up to signal termination of the device operation the next time the device requests or offers a byte of data, if any. Chaining, if indicated in the subchannel, is suppressed.

When the channel is working in burst mode with a device other than the one addressed and the subchannel is not operational, is in the interruption pending state, or is working with a device other than the one addressed, the resulting condition code may in some channels be determined by the subchannel state.

Termination of a burst operation by HALT DEVICE on a selector channel causes the channel and subchannel to be placed in the interruption pending state. Generation of the interruption condition is not contingent on the receipt of a status byte from the device. When HALT DEVICE causes a burst operation on a byte-multiplexer channel to be terminated, the subchannel associated with the burst operation remains in the working state until the device provides ending status, whereupon the subchannel enters the interruption pending state. The termination of a burst operation by HALT DEVICE on a block-multiplexer channel may, depending on the model and the type of subchannel, perform as a selector channel or may allow the subchannel to remain in the working state until the device provides ending status.

When any of three conditions occur (described later), HALT DEVICE causes the 16-bit unit and channel status portion of the CSW to be replaced by a new set of status bits. The contents of the other fields of the CSW are not changed. The CSW stored by HALT DEVICE pertains only to the execution of HALT DEVICE and does not describe under what conditions the I/O operation at the addressed subchannel is terminated. The extent of data transfer, and the conditions of termination of the operation at the subchannel, are provided in the CSW associated with the interruption condition caused by the termination. The three conditions are:

1. The addressed device is selected and signaled to terminate the current operation, if any. The CSW then contains zeros in the status field unless a machine malfunction is detected.

2. The control unit is busy, and the device cannot be given the signal to terminate the operation. The CSW unit status field contains the busy and status modifier bits. The channel status field contains zeros unless a machine malfunction is detected.

3. The channel detects a machine malfunction during the execution of HALT DEVICE. The status bits in the CSW then identify the error condition. The state of the channel and the progress of the I/O operation are unpredictable.

When HALT DEVICE causes data transfer to be terminated, the control unit associated with the operation remains unavailable until the data-handling portion of the operation in the control unit is terminated. Termination of the data-transfer portion of the operation is signaled by the generation of channel end. This may occur at the normal time for the operation, or earlier, or later, depending on the operation and type of device. If the control unit is shared, all devices attached to the control unit appear in the working state until the channel end condition is accepted by the CPU. The I/O device executing the terminated operation remains in the working state until termination of the inherent cycle of the operation, at which time device end is generated. If blocks of data at the device are defined, as in read-type operations on magnetic tape, the recording medium is advanced to the beginning of the next block.

If HALT DEVICE is issued when the subchannel is available and no burst operation is in progress, the effect of the HALT DEVICE signal depends partially on the type of device and its state. In all cases, the HALT DEVICE signal has no effect on devices that are not in the working state or are executing a mechanical operation in which data is not transferred across the I/O interface, such as rewinding tape or positioning a disk access mechanism. If the device is executing a type of operation that is unpredictable in duration, or in which

data is transferred across the I/O interface, the device interprets the signal as one to terminate the operation. Pending attention or device end conditions at the device are not reset.

Condition Code:

0 Subchannel busy with another device or interruption pending
1 CSW stored
2 Channel working
3 Not operational

Program Interruptions:

Privileged operation: The CPU is in the problem state. The operation is suppressed.

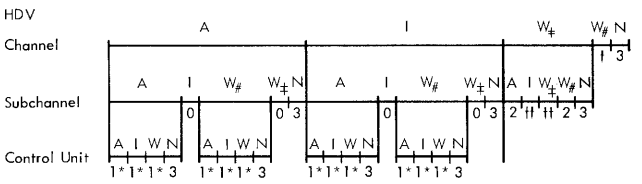
The condition code set by HALT DEVICE for all possible states of the I/O system is shown graphically in Figure 27.1. See "States of the Input/Output System" for a complete definition of the A, I, W, and N states.

Programming Notes

Some selector and byte-multiplexer channels designed prior to the defining of HALT DEVICE, (e.g., the 2860) will execute HALT DEVICE as HALT I/O even though a CPU to which they attach can distinguish between the instructions. A program can ensure complete compatibility between HALT DEVICE and HALT I/O on such channels by observing the following conventions:

1. On a byte-multiplexer channel, do not issue HALT DEVICE to a multiplexing device while a burst operation is in progress on the channel.

2. On a byte-multiplexer channel, do not issue HALT DEVICE to a device on a shared subchannel while that subchannel is working with a device other than the one addressed.



A = Available
I = Interruption Pending
W = Working
W≠ = Working with a device other than the one addressed
W# = Working with the addressed device
N = Not Operational
* = CSW Stored

Notes:

† In the W≠XX state, either a condition code of 1 (with the CSW stored) or a condition code of 2 may be set, depending on the channel and the conditions in the channel. Condition code 1 (with the CSW stored) can only be set if the control unit has received the signal to terminate.

†† In the W≠IX and W≠W≠X states, either condition code 0 or 2 may be set, depending on the channel and the conditions in the channel.

Figure 27.1. Condition Code Set by HALT DEVICE

3. On a selector channel in the working state, do not issue HALT DEVICE to any device other than the one with which the channel is working.

The execution of HALT DEVICE always causes data transfer across the I/O interface for the addressed device to be terminated. The condition code and the CSW (when stored) indicate whether the control unit was signaled to terminate operation during the execution of the instruction. If the control unit was not signaled to terminate operation, the condition code and the CSW (when stored) imply the conditions under which the execution of a HALT DEVICE for the same address will cause the control unit to be signaled to terminate.

Condition Code 0 indicates that HALT DEVICE cannot signal the control unit until an interruption condition on the same subchannel is cleared.

Condition Code 1 with Control-Unit Busy Status In the CSW indicates that HALT DEVICE cannot signal the control unit until the control-unit-end status is received from that control unit.

Condition Code 1 with Zeros In the Status Field of the CSW indicates that the addressed device was selected and signaled to terminate the current operation, if any.

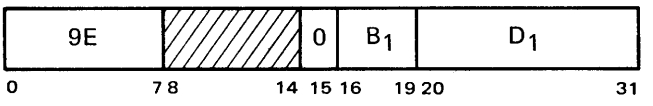
Condition Code 2 indicates that the control unit cannot be signaled until the end of a busy condition in the channel. The end of the busy condition can be detected by noting an interruption from the channel or by noting the results of repeatedly executing HALT DEVICE.

Condition Code 3 indicates that manual intervention is required to allow HALT DEVICE to signal the control unit to terminate.

Halt I/O

When the HALT DEVICE instruction is implemented on a model of System/360, the machine-language format of the HALT I/O instruction is changed to include a zero in bit position 15:

HIO ***$D_1(B_1)$*** ***[SI]***



Channel Available Interruption

The block-multiplexer channel is alternately busy and idle for extended periods of time. To facilitate starting channel programs while the block-multiplexer channel is idle, a new I/O interruption condition, "channel available," is signaled by the channel.

The channel generates this interruption condition (1) if it had previously responded with condition code 2 to a START I/O, TEST I/O, TEST CHANNEL, or HALT DEVICE instruction, (2) if the busy condition indicated by that code no longer exists, and (3) if no interruption other than a program-controlled interruption (PCI) is pending at the channel.

If, after the channel available condition is generated but before it causes an interruption, another interruption condition other than a PCI is generated, the channel available condition is eliminated. If, however, the channel again becomes busy before the channel available condition is presented, the channel available condition is held pending until that busy condition no longer exists.

When the channel available condition causes an interruption, the channel address is stored in the old PSW, with the device address bits set to zero. The entire CSW is set to zero. No special bits are set as a result of the interruption condition.

Command Retry

Command retry (allowed on some channels) is a channel and control-unit procedure that causes a command to be retried without requiring an I/O interruption. This retry is initiated by the control unit by a unique combination of status bits.

When the command being executed encounters a condition requiring retry, the control unit indicates command retry, together with channel end (indicating that the control unit or the device is not ready to retry the command) or with channel end and device end (indicating that the control unit and the device are prepared for immediate retry of the command). Inclusion of any other unit status conditions indicates an unusual situation causing termination of subchannel program execution and creation of an I/O interruption condition.

When command retry is performed, a START I/O initiating an immediate operation for which command chaining is not indicated in the CCW causes condition code 0, rather than condition code 1, to be set. The termination of the channel program causes an interruption condition to be created.

Programming Note

A command retry during the execution of a channel program may cause the following conditions to be detected by the initiating program:

1. A CCW containing a PCI may, if retried because of command retry, cause multiple PCI interruptions to occur.

2. A channel program consisting of a single, unchained CCW specifying an immediate command may cause a condition code of zero rather than one to be set. This setting of the condition code occurs if the control unit signals command retry at the time initial status is presented to the command. The channel program then causes a later interruption upon completion of the operation.

3. If premature termination of the execution of a channel program occurs during the retry of a command, the residual count and command address field in the CSW may not necessarily indicate the extent of main storage used.

4. If a CCW used in an operation is changed before that operation has been successfully completed, the results are unpredictable.

Test I/O

The conditions under which TEST I/O causes the CSW to be stored are changed in this way: when the channel is operating in burst mode and the addressed subchannel contains a pending interruption condition, condition code 1 or 2 may be set, depending on the channel type and system model. If condition code 1 is set, the CSW is stored to identify the interruption condition, and the interruption condition is cleared.

I/O Error Alert

Interface control check may be caused by a condition called I/O Error Alert on a channel that recognizes Disconnect In on the I/O interface. Disconnect In is a new line that provides control units with the capability of alerting the system to a malfunction that prevents the control unit from signaling properly over the I/O interface.

The channel indicates to the program the occurrence of I/O Error Alert by causing an I/O interruption. The CSW stored has the interface control check bit on.

Channel Data Check

To determine the cause and the source of the parity error causing a channel data check indication, more information than that supplied by the CSW is needed. Accordingly, some channels cause a logout to be performed when an interruption is taken that indicates a channel data check has occurred. The logout identifies the source of the error.

Interruptions from More than Seven Channels

On CPU's to which more than seven channels can be attached, PSW bit 6 governs interruptions by channels 6 and up. When the bit position contains a one, the CPU is enabled for interruptions by any channel having an address of 6 or higher. When the bit position contains a zero, interruptions by these channels are disallowed and remain pending.

System Control Panel

The system control panel contains the switches and lights necessary to operate and control the system. The system consists of the CPU, storage, channels, on-line control units, and I/O devices. Off-line control units and I/O devices, although part of the system environment, are not considered part of the system proper.

System controls are divided into three sections: operator control, operator intervention, and customer engineering control. Customer engineering controls are also available on some storage, channel, and control-unit frames.

No provision is made for locking out any section of the system control panel. The conditions under which individual controls are active are described for each case.

System Control Functions

The main functions provided by the system control panel are the ability to reset the system; to store and display information in storage, in registers and in the PSW; and to load initial program information.

System Reset

The system-reset function resets the CPU, the channels, and on-line, nonshared control units and I/O devices.

The CPU is placed in the stopped state and all pending interruptions are eliminated. The parity of general and floating-point registers, as well as the parity of the PSW, may be corrected. All error-status indicators are reset to zero.

In general, the system is placed in such a state that processing can be initiated without the occurrence of machine checks, except those caused by subsequent machine malfunction.

The reset state for a control unit or device is described in the appropriate System Reference Library (SRL) publication. Off-line control units are not reset. A system-reset signal from a CPU resets only the functions in a shared control unit or device belonging to that CPU. Any function pertaining to another CPU remains undisturbed.

The system-reset function is performed when the system-reset key is pressed, when initial program loading is initiated, or when a power-on sequence is performed.

Programming Notes

Because the system reset may occur in the middle of an operation, the contents of the PSW and of result registers or storage locations are unpredictable. If the CPU is in the wait state when the system reset is performed, and I/O is not operating, this uncertainty is eliminated.

Following a system reset, incorrect parity may exist in storage in all models and in the registers in some models. Since a machine check occurs when information with incorrect parity is used, the incorrect information should be replaced by loading new information.

Store and Display

The store-and-display function permits manual intervention in the progress of a program. The store-and-display function may be provided by a supervisor program in conjunction with proper I/O equipment and the interrupt key.

In the absence of an appropriate supervisor program, the controls on the operator intervention panel permit the CPU to be placed in the stopped state, and subsequently to store and display information in main storage, in general and floating-point registers, and in the instruction-address part of the PSW. The stopped state is achieved at the end of the current instruction when the stop key is pressed, when single instruction execution is specified, or when a preset address is reached. Once the desired intervention is completed, the CPU can be started again.

The stopping and starting of the CPU in itself does not cause any alteration in program execution other than the time element involved (the transition from operating to stopped state is described under "Stopped State" in "Status Switching").

All basic store-and-display functions can be simulated by a supervisor program.

Machine checks occurring during store-and-display functions do not interrupt or log immediately but may, in some cases, create a pending interruption. This interruption request can be removed by a system reset. Otherwise, the interruption, when not masked off, is taken when the CPU is again in the operating state.

Initial Program Loading

Initial program loading (IPL) is provided for the initiation of processing when the contents of storage or the PSW are not suitable for further processing.

Initial program loading is initiated manually by selecting an input device with the load-unit switches and subsequently pressing the load key. When facilities for external system initialization are installed, initial program loading may be initiated externally by a signal received on one of the external-start lines.

Pressing the load key causes a system reset, turns on the load light, turns off the manual light, sets the prefix trigger (if present), and subsequently initiates a read operation from the selected input device. When reading is completed satisfactorily, a new PSW is obtained, the CPU starts operating, and the load light is turned off.

When a signal is received on one of the external-start lines, the same sequence of events takes place, except that the read operation is omitted.

System reset suspends all instruction processing, interruptions, and timer updating and also resets all channels, on-line nonshared control units, and I/O devices. The contents of general and floating-point registers remain unchanged, except that the reset procedure may introduce incorrect parity.

The prefix trigger is set after system reset. In manually initiated IPL, the trigger is set according to the state of the prefix-select key switch. When IPL is initiated by a signal on one of the two external-start lines, the trigger is set according to the identity of each line. The prefix trigger is part of the multisystem feature.

Next, if IPL is initiated manually, the selected input device starts reading. The first 24 bytes read are placed in storage locations 0-23. Storage protection, program controlled interruption, and a possible incorrect length indication are ignored. The double-word read into location 8 is used as the channel command word (CCW) for a subsequent I/O operation. When chaining is specified in this CCW, the operation proceeds with the CCW in location 16. Either command chaining or data chaining may be specified.

When the device provides channel end for the last operation of the chain, the I/O address is stored in bits 21-31 of the first word in storage. Bits 16-20 are made zero. Bits 0-15 remain unchanged. The input operation and the storing of the I/O address are not performed when IPL is initiated by means of the external-start lines.

The CPU subsequently fetches the double word in location 0 as a new PSW and proceeds under control of the new PSW. The load light is turned off. No I/O interruption condition is generated. When the I/O operations and PSW loading are not completed satisfactorily, the CPU idles, and the load light remains on.

Programming Notes

Initial program loading resembles a START I/O that specifies the I/O device selected in the load-unit switches and a zero protection key. The CCW for this START I/O is simulated by CPU circuitry and contains a read command, zero data address, a byte count of 24, command-chain flag on, suppress-length-indication flag on, program-controlled-interruption flag off, chain-data flag off, and skip flag off. The CCW has a virtual address of zero.

Initial program loading reads new information into the first six words of storage. Since the remainder of the IPL program may be placed in any desired section of storage, it is possible to preserve such areas of storage as the timer and PSW locations, which may be helpful in program debugging.

If the selected input device is a disk, the IPL information is read from track 0.

The selected input device may be a channel-to-channel adapter involving two CPU's. After a system reset on this adapter, an attention signal is sent to the addressed CPU. That CPU then should issue the write command necessary to load a program into main storage of the requesting CPU.

When the PSW in location 0 has bit 14 set to one, the CPU is in the wait state after the IPL procedure (the manual, the system, and the load lights are off, and the wait light is on). Interruptions that become pending during IPL are taken before instruction execution.

Operator Control Section

This section of the system control panel contains only the controls required by the operator when the CPU is operating under full supervisor control. Under supervisor control, a minimum of direct manual intervention is required since the supervisor performs operations like store and display.

The main functions provided by the operator control section are the control and indication of power, the indication of system status, operator to machine communication, and initial program loading.

The operator control section, with the exception of the emergency pull switch, may be duplicated once as a remote panel on a console.

The following table lists all operator controls by the names on the panel or controls and describes their implementation.

NAME	IMPLEMENTATION
Emergency Pull	Pull switch
Power On	Key, backlighted
Power Off	Key
Interrupt	Key
Wait	Light
Manual	Light
System	Light
Test	Light
Load	Light
Load Unit	Three rotary switches
Load	Key
Prefix Select*	Key switch

* Multisystem feature

Emergency Pull Switch

Pulling this switch turns off all power beyond the power-entry terminal on every unit that is part of the system or that can be switched onto the system. Therefore, the switch controls the system proper and all off-line and shared control units and I/O devices.

The switch latches in the out position and can be restored to its in position by maintenance personnel only.

When the emergency pull switch is in the out position, the power-on key is ineffective.

Power-On Key

This key is pressed to initiate the power-on sequence of the system.

As part of the power-on sequence, a system reset is performed in such a manner that the system performs no instructions or I/O operations until explicitly directed. The contents of main storage are preserved.

The power-on key is backlighted to indicate when the power-on sequence is completed. The key is effective only when the emergency pull switch is in its in position.

Power-Off Key

The power-off key is pressed to initiate the power-off sequence of the system.

The contents of main storage (but not the keys in storage associated with the protection feature) are preserved, provided that the CPU is in the stopped state. The key is effective while power is on the system.

Interrupt Key

The interrupt key is pressed to request an external interruption.

The interruption is taken when not masked off and when the CPU is not stopped. Otherwise, the interruption request remains pending. Bit 25 in the interruption-code portion of the current PSW is made one to indicate that the interrupt key is the source of the external interruption. The key is effective while power is on the system.

Wait Light

The wait light is on when the CPU is in the wait state.

Manual Light

The manual light is on when the CPU is in the stopped state. Several of the manual controls are effective only when the CPU is stopped, that is, when the manual light is on.

System Light

The system light is on when the CPU cluster meter or customer-engineering meter is running.

Programming Note

The states indicated by the wait and manual lights are independent of each other; however, the state of the system light is not independent of the state of these two lights because of the definition of the running condition for the meters. The following table shows possible conditions when power is on.

SYSTEM LIGHT	MANUAL LIGHT	WAIT LIGHT	CPU STATE	I/O STATE
off	off	off	*	*
off	off	on	Wait	Not working
off	on	off	Stopped	Not working
off	on	on	Stopped, wait	Not working
on	off	off	Running	Undetermined
on	off	on	Wait	Working
on	on	off	Stopped	Working
on	on	on	Stopped, wait	Working

* Abnormal condition

Test Light

The test light is on when a manual control is not in its normal position or when a maintenance function is being performed for CPU, channels, or storage.

Any abnormal switch setting on the system control panel or on any separate maintenance panel for the CPU, storage, or channels that can affect the normal operation of a program causes the test light to be on.

The test light may be on when one or more diagnostic functions under control of DIAGNOSE are activated or when certain abnormal circuit breaker or thermal conditions occur.

The test light does not reflect the state of marginal voltage controls.

Load Light

The load light is on during initial program loading; it is turned on when the load key is pressed and is turned off after the loading of the new PSW is completed successfully.

Load-Unit Switches

Three rotary switches provide the 11 rightmost I/O address bits used for initial program loading.

The leftmost rotary switch has eight positions labeled 0-7. The other two are 16-position rotary switches labeled with the hexadecimal characters 0-9, A-F.

Load Key

The load key is pressed to start initial program loading. The key is effective while power is on the system.

Prefix-Select Key Switch

The prefix-select key switch provides the choice between main prefix and alternate prefix during manually initiated initial program loading.

The setting of the switch determines the state of the prefix trigger following the system reset after the load key is pressed.

The switch is part of the multisystem feature.

Operator Intervention Section

This section of the system control panel contains the controls required for the operator to intervene in normal programmed operation. These controls may be intermixed with the customer engineering controls, and additional switch positions and nomenclature may be included, depending on the model.

Operator intervention provides the system-reset and the store-and-display functions. Compatibility in performing these functions is maintained, except that the word size used for store and display depends on the physical word size of storage for the model. Switches for display of the instruction address are absent on models that continuously display the instruction address.

The following table lists all intervention controls by the names on the panel or controls and describes their implementation.

NAME	IMPLEMENTATION
System Reset	Key
Stop	Key
Rate	Rotary switch
Start	Key
Storage Select	Rotary or key switch
Address	Rotary or key switches
Data	Rotary or key switches
Store	Key
Display	Key
Set IC	Key
Address Compare	Rotary or key switches
Alternate Prefix*	Light

* Multisystem feature

System-Reset Key

The system-reset key is pressed to cause a system reset; it is effective while power is on the system. The reset function does not affect any off-line or shared device.

Stop Key

The stop key is pressed to cause the CPU to enter the stopped state. The key is effective while power is on the system.

Programming Note

Pressing the stop key has no effect when a continuous string of interruptions is performed or when the CPU is unable to complete an instruction because of machine malfunction. The effect of pressing the key is indicated by the turn-on of the manual light as the CPU enters the stopped state.

Rate Switch

This rotary switch indicates the manner in which instructions are to be performed.

The switch has two or more positions, depending on model. The vertical position is marked PROCESS. In this position, the system starts operating at normal speed when the start key is pressed. The position left of vertical is marked INSTRUCTION STEP. When the start key is pressed with the rate switch in this position, one complete instruction is performed, and all pending, not masked interruptions are subsequently taken. The CPU next returns to the stopped state.

Any instruction can be executed with the rate switch set to INSTRUCTION STEP. Input/output operations are completed to the interruption point. When the CPU is in the wait state, no instruction is performed, but pending interruptions, if any, are taken before the CPU returns to the stopped state. Initial program loading is completed with the loading of the new PSW before any instruction is performed. The timer is not updated while the rate switch is set to INSTRUCTION STEP.

The test light is on when the rate switch is not set to PROCESS.

The position of the rate switch should be changed only while the CPU is in the stopped state. Otherwise unpredictable results occur.

Start Key

The start key is pressed to start instruction execution in the manner defined by the rate switch.

Pressing the start key after a normal stop causes instruction processing to continue as if no stop had occurred, provided that the rate switch is in the PROCESS or INSTRUCTION-STEP position. Pressing the start key after system reset without first introducing a new instruction address yields unpredictable results.

The key is effective only while the CPU is in the stopped state.

Storage-Select Switch

The storage area to be addressed by the address switches is selected by the storage-select switches.

The switch can select main storage, the general registers, the floating-point registers and, in some cases, the instruction-address part of the PSW.

When the general or floating-point registers are not addressed directly but must be addressed by using another address such as a local-store location, information is included on the panel to enable an operator to compute the required address.

The switch can be manipulated without disrupting CPU operations.

Address Switches

The address switches address a location in a storage area and can be manipulated without disrupting CPU operation. The address switches, with the storage-select switch, permit access to any addressable location. Correct address parity is generated.

Data Switches

The data switches specify the data to be stored in the location specified by the storage-select switch and address switches.

The number of data switches is sufficient to allow storing of a full physical storage word. Correct data parity is generated. Some models generate either correct or incorrect parity under switch control.

Store Key

The store key is pressed to store information in the location specified by the storage-select switch and address switches.

The contents of the data switches are placed in the main storage, general register, or floating-point register location specified. Storage protection is ignored. When the location designated by the address switches and storage-select switch is not available, data are not stored.

The key is effective only while the CPU is in the stopped state.

Display Key

The display key is pressed to display information in the location specified by the storage-select switch and address switches.

The data in the main storage, general register, or floating-point register location, or in the instruction-address part of the PSW specified by the address switches and the storage-select switch, are displayed. When the designated location is not available, the displayed information is unpredictable. In some models, the current instruction address is continuously displayed and hence is not explicitly selected.

The key is effective only while the CPU is in the stopped state.

Set IC Key

This key is pressed to enter an address into the instruction-address part of the current PSW.

The address in the address switches is entered in bits 40-63 of the current PSW. In some models the address is obtained from the data switches.

The key is effective only while the CPU is in the stopped state.

Address-Compare Switch

These rotary or key switches provide a means of stopping the CPU on a successful address comparison.

When these switches are set to the STOP position, the address in the address switches is compared against the value of the instruction address on all models and against all addresses on some models. A match causes the CPU to enter the stopped state. Comparison includes only the part of the instruction address that addresses the physical word size of storage.

Comparison of the entire halfword instruction address is provided in some models, as is the ability to compare data addresses.

The address-compare switches can be manipulated without disrupting CPU operation other than by causing the address-comparison stop. When they are set to any position except NORMAL, the test light is on.

Programming Note

When an address not used in the program is selected in the address switches, the CPU runs as if the address-compare switches were set to normal, except for the reduction in performance which may be caused by the address comparison.

Alternate-Prefix Light

The alternate-prefix light is on when the prefix trigger is in its alternate state. The light is part of the multisystem feature.

Customer Engineering Section

This section of the system control panel contains controls intended only for customer-engineering use.

Appendix A. Instruction Use Examples

The following examples illustrate the use of many System/360 instructions. Before studying one of these examples, the reader should first consult the instruction description in this manual for the particular instruction of interest to him. Note that each instruction description contains the System/360 assembly language mnemonic op code and symbolic operand designation as well as the machine instruction format.

For clarity and for ease in programming, each example in this section presents the instruction both as it is written in an assembly-language statement and as it appears when assembled in storage (hexadecimal machine format). As a rule, all numerical operands are written in hexadecimal format unless otherwise specified. Hexadecimal operands are shown converted into binary and/or decimal if such conversion helps to clarify the example for the reader. Storage addresses are also given in hexadecimal. In the assembly-language statements, registers, lengths, and masks are all presented in decimal, but displacements may be in hexadecimal or decimal. A hexadecimal displacement is indicated by X *'a number'*, where the number can range from 000-FFF$_{16}$. Immediate operands are normally shown in hexadecimal. Whenever the value in a register or storage location is referred to as "not significant," this value is replaced during the execution of the instruction.

When writing ss format instructions in System/360 assembly language, lengths are given as the total number of bytes in the field. This differs from the machine definition regarding lengths which states that the length is the number of bytes to be added to the field address to obtain the address of the last byte of the field. Thus the machine length is one less than the assembly-language length. The assembly program automatically subtracts one from the length specified when the instruction is assembled.

Branching

Branch On Condition (BC, BCR)

The BRANCH ON CONDITION instructions test the condition code to see whether a branch should or should not be taken. The branch is taken only if the condition code is as specified by a mask.

MASK VALUE	CONDITION CODE
8	0
4	1
2	2
1	3

For example, assume that an add (A, AR) operation has been performed and that you wish to branch to address 6050 if the sum is zero or less (condition code = 0 or 1). Also assume:

Register 10 contains 00 00 50 00
Register 11 contains 00 00 10 00

The RX form of the instruction performs the required test (and branch, if necessary) when written as:

Machine Format

OP CODE	M1	X2	B2	D2
47	C	B	A	050

Assembler Format

OP CODE	M1	D2	X2 B2
BC	12,X'50'(11,10)		

A mask of 15 indicates a branch on any condition (an unconditional branch). A mask of zero indicates that no branch is to occur (a no-operation).

Branch and Link (BAL, BALR)

The BRANCH AND LINK instructions are commonly used to branch to a subroutine with the option of later returning to the main instruction sequence. For example, assume that you wish to branch to a subroutine at storage address 1160. Also assume:

The contents of register 2 are not significant
Register 5 contains 00 00 11 50
There is a BAL instruction at address 00 00 C6
(PSW bits 40-63 will contain 00 00 CA after execution of BAL)

The format of the BAL instruction is:

Machine Format

OP CODE	R1	X2	B2	D2
45	2	0	5	010

Assembler Format

OP CODE	R1	D2	X2 B2
BAL	2,X'10'(0,5)		

After the instruction is executed:

Register 2 (bits 8-31) contains 00 00 CA
PSW bits 40-63 contain 00 11 60

The programmer can return to the main instruction sequence at any time with a BRANCH ON CONDITION (BCR) instruction that specifies register 2 and a mask of 15_{10}, provided that register 2 has not meanwhile been disturbed.

The BALR instruction with the R_2 field equal to zero may be used to load a register for use as a base

register. For example, in the assembly language, the sequence of statements:

```
BALR      15, 0
USING     *, 15
```

tells the assembly program that register 15 is to be used as the base register in assembling this program segment and that when the program is executed, the address of the next sequential instruction following the BALR will be placed in the register. (The USING statement is an "assembler instruction" and is thus not a part of the object program.)

At any time the condition code may be preserved for future inspection with BALR R_1,0. Bits 2 and 3 of the register (R_1) contain the condition code.

Branch On Count (BCT, BCTR)

The BRANCH ON COUNT instructions are often used to execute a program loop for a specified number of times. For example, assume that the following represents some lines of coding in an assembly language program:

```
      .
      .
      .
LUPE      AR      8,1
      .
      .
      .
BACK      BCT     6,LUPE
      .
      .
      .
```

where register 6 contains 00 00 00 03 and the address of LUPE is 6826. Also assume that register 10 contains 00 00 68 00.

The format of the BCT instruction is:

Machine Format

OP CODE	R1	X2	B2	D2
46	6	0	A	026

Assembler Format (alternate form to above)

OP CODE	R1	D2	X2	B2
BCT	6,X'26'	(0,10)		

The effect of the coding shown above is to execute three times the loop defined by locations LUPE and BACK.

Branch On Index High (BXH)

The BRANCH ON INDEX HIGH instruction is an index-incrementing and loop-controlling instruction that causes a branch whenever the sum of an index value and an increment value is greater than some comparand. For example, assume that:

Register 4 contains 00 00 00 8A = 138_{10} = the index
Register 6 contains 00 00 00 02 = 2_{10} = the increment
Register 7 contains 00 00 00 AA = 170_{10} = the comparand
Register 10 contains 00 00 71 30 = the branch address

The format of the instruction is:

Machine Format

OP CODE	R1	R3	B2	D2
86	4	6	A	000

Assembler Format

OP CODE	R1,R3,D2	B2
BXH	4,6,0	(10)

When the instruction is executed, first the contents of register 6 are added to register 4, second the sum is compared with the contents of register 7, and third the decision to branch or not to branch is made. After execution:

Register 4 contains 00 00 00 8C = 140_{10}
Registers 6 and 7 are unchanged

Since the new value in register 4 is not greater than the value in register 7, the branch to address 7130 is not taken.

When the register used to contain the increment is odd, that register also becomes the comparand register. The following assembly-language subroutine illustrates how this feature may be used to search a table.

Table

ARG1	FUNCT1
ARG2	FUNCT2
ARG3	FUNCT3
ARG4	FUNCT4
ARG5	FUNCT5
ARG6	FUNCT6
2 bytes	2 bytes

Assume that:

Register 0 contains the search argument
Register 1 contains the width of the table in bytes (00 00 00 04)
Register 2 contains the length of the table in bytes (00 00 00 18)
Register 3 contains the starting address of the table
Register 14 contains the return address in the main program

As the following subroutine is executed, the argument in register 0 is successively compared with the arguments in the table. If an equality is found, the corresponding function replaces the argument in register 0. If an equality is not found, FF_{16} replaces the argument in register 0.

```
SEARCH      LNR     1,1
NOTEQUAL    BXH     2,1,LOOP
NOTFOUND    LA      0,X'FF'
            BCR     15,14
LOOP        CH      0,0(2,3)
            BC      7,NOTEQUAL
            LH      0,2(2,3)
            BCR     15,14
```

Branch On Index Low or Equal (BXLE)

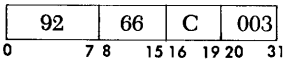
This instruction is similar to BRANCH ON INDEX HIGH except that the branch is successful when the sum is low or equal compared to the comparand.

Execute (EX)

The EXECUTE instruction causes *one* instruction in main storage to be executed out of sequence without actually branching to the object instruction. For example, as-

sume that a MOVE (SI) instruction is located at address 3820, with format as follows:

Machine Format



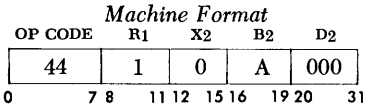
Assembler Format

OP CODE	D1	B1	I2
MVI	3(12),		X'66'

where register 12 contains 00 00 89 16.

Further assume that at storage address 5000, the following EXECUTE instruction is located:

Machine Format



Assembler Format

OP CODE	R1	D2	X2	B2
EX	1,0(	0,10)		

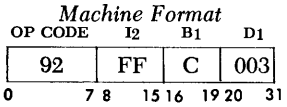
where register 10 contains 00 00 38 20, and register 1 contains 00 0F F0 99.

When the instruction at 5000 is executed, bits 24-31 of register 1 are OR'ed inside the CPU with bits 8-15 of the instruction at 3820:

Bits 8-15: $0110\ 0110_2 = 66$
Bits 24-31: $1001\ 1001_2 = 99$
Result: $1111\ 1111_2 = FF$

causing the instruction at 3820 to be executed *as if* it originally were:

Machine Format



Assembler Format

OP CODE	D1	B1	I2
MVI	3(12),		X'FF'

However, after execution:

Register 1 is unchanged
The instruction at 3820 is unchanged
Storage location 8919 contains FF
The CPU next executes the instruction at address 5004
(PSW bits 40-63 contain 00 50 04)

Fixed-Point Arithmetic

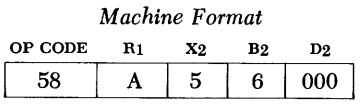
Load (L, LR)

The LOAD instructions place, unchanged, the contents of a word in storage or of a register into another register. For example, assume that the four bytes starting with location 21004 (a full-word boundary) are to be loaded into register 10. Initially:

Register 5 contains 00 02 00 00
Register 6 contains 00 00 10 04
The contents of register 10 are not significant
Storage locations 21004-21007 contain 00 00 AB CD

To load register 10, the RX form of the instruction can be used:

Machine Format



Assembler Format

OP CODE	R1	D2	X2	B2
L	10,0(	5,6)		

After the instruction is executed, register 10 contains 00 00 AB CD.

Load Halfword (LH)

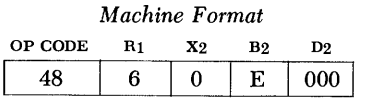
The LOAD HALFWORD instruction places unchanged the contents of a halfword in storage into the right half of a register. The left half of the register is replaced by zeros or ones to reflect the sign (leftmost bit) of the halfword.

For example, assume that the two bytes in storage locations 1802-1803 are to be loaded into register 6. Also assume:

Register 6 contains 7F 12 34 56
Register 14 contains 00 00 18 02
Locations 1802-1803 contain 00 20

The instruction required to load the register is:

Machine Format



Assembler Format

OP CODE	R1	D2	X2	B2
LH	6,0(	0,14)		

After the instruction is executed, register 6 contains 00 00 00 20. If 1802-1803 contained a negative number, for example A7 B6, the sign bit would again be propagated to the left, giving FF FF A7 B6 as the final result in register 6.

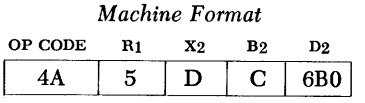
Add Halfword (AH)

The ADD HALFWORD instruction algebraically adds the halfword contents of a storage location to the contents of a register. The halfword storage operand is expanded to 32 bits after it is fetched and before it is used in the add operation. The expansion consists of propagating the leftmost (sign) bit 16 positions to the left. For example, assume that the contents of storage locations 2000-2001 are to be added to register 5. Initially:

Register 5 contains 00 00 00 19 = 25_{10}
Storage locations 2000-2001 contain FF FE = -2_{10}
Register 12 contains 00 00 18 00
Register 13 contains 00 00 01 50

The format of the required instruction is:

Machine Format



Assembler Format

OP CODE	R1	D2	X2	B2
AH	5,X'6B0'(	13,12)		

After the instruction is executed, register 5 contains 00 00 00 17 = 23_{10}

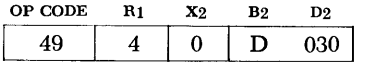
Compare Halfword (CH)

The COMPARE HALFWORD instruction compares a halfword in storage with the contents of a register. For example, assume that:

Register 4 contains FF FF 80 00 = $-32{,}768_{10}$
Register 13 contains 00 01 60 50
Storage locations 16080-16081 contain 8000 = $-32{,}768_{10}$

When the instruction:

Machine Format



Assembler Format

OP CODE	R1	D2	X2	B2
CH	4,X'30'(	0,13)		

is executed, the contents of locations 16080-16081 are fetched, expanded to 32 bits (the sign bit is propagated to the left), and compared with the contents

of register 4. Because the two numbers are equal, the condition code is set to 0.

Multiply (M, MR)

Assume that a number in register 5 is to be multiplied by the contents of a word at address 3750. Initially:

The contents of register 4 are not significant
Register 5 contains 00 00 00 9A = 154_{10} = the multiplicand
Register 11 contains 00 00 30 00
Register 12 contains 00 00 06 00
Storage locations 3750-3753 contain 00 00 00 83 = 131_{10} = the multiplier

The instruction required to perform the multiplication is:

Machine Format

OP CODE	R1	X2	B2	D2
5C	4	B	C	150

Assembler Format

OP CODE	R1	D2	X2	B2
M	4,X'150'(11,12)			

After the instruction is executed:

Register 4 contains 00 00 00 00
Register 5 contains 00 00 4E CE = $20,174_{10}$ } product
Storage locations 3750-3753 are unchanged

The RR format of the instruction can be used to square a number in a register. Assume that register 7 contains 00 00 00 10 = 16_{10}. The instruction:

Machine Format

OP CODE	R1	R2
1C	6	7

Assembler Format

OP CODE	R1R2
MR	6,7

multiplies the number in register 7 by itself. The product, 00 00 00 00 00 00 01 00 = 256_{10}, appears in registers 6 and 7.

Multiply Halfword (MH)

The MULTIPLY HALFWORD instruction is used to multiply a register by a halfword in storage. For example, assume that:

Register 11 contains 00 00 00 15 = 21_{10} = the multiplicand
Register 14 contains 00 00 01 00
Register 15 contains 00 00 20 00
Storage locations 2102-2103 contain FF D9 = -39_{10} = the multiplier

The instruction:

Machine Format

OP CODE	R1	X2	B2	D2
4C	B	E	F	002

Assembler Format

OP CODE	R1	D2	X2	B2
MH	11,2(14,15)			

multiplies the two numbers. The product, FF FF FC CD = -819_{10}, replaces the original contents of register 11.

Only the low-order 32 bits of a product are stored in a register; any higher-order bits are lost. No program interruption occurs upon overflow.

Divide (D, DR)

The DIVIDE instruction divides a dividend in an even/odd register pair by a divisor in a register or in storage. Since the dividend is assumed to be 64 bits long, it is important that the proper sign is first affixed. For example, assume that:

Storage locations 3550-3553 contain 00 00 08 D7 = 2270_{10} = the dividend
Storage locations 3554-3557 contain 00 00 00 32 = 50_{10} = the divisor
Register 6 does not contain all zeros
The initial contents of register 7 are not significant
Register 8 contains 00 00 35 50

The following assembly language statements load the registers properly and perform the divide operation:

		COMMENTS
L	6,0(0,8)	Places 00 00 08 D7 into register 6.
SRDA	6,32(0)	Shifts 00 00 08 D7 into register 7. Register 6 is filled with zeros (sign bits).
D	6,4(0,8)	Performs the division.

The machine format of the above DIVIDE instruction is:

OP CODE	R1	X2	B2	D2
5D	6	0	8	004

After all the above instructions are executed:

Register 6 contains 00 00 00 14 = 20_{10} = the remainder
Register 7 contains 00 00 00 2D = 45_{10} = the quotient

Note that if the dividend had not been first placed in register 6 and shifted into register 7, register 6 would not have been filled with the proper sign bits (zeros in this example) and the DIVIDE instruction would not have given the expected results.

Convert to Binary (CVB)

The CONVERT TO BINARY instruction converts an eight-byte, signed, packed-decimal number into a signed binary number and loads the result into a general register. After the conversion operation is completed, the number is in the proper form for use as an operand in fixed-point arithmetic. For example, assume:

Storage locations 7608-760F contain 00 00 00 00 00 25 59 4C, a positive packed-decimal number
The contents of register 7 are not significant
Register 13 contains 00 00 76 00

The format of the conversion instruction is:

Machine Format

OP CODE	R1	X2	B2	D2
4F	7	0	D	008

Assembler Format

OP CODE	R1D2X2	B2
CVB	7,8(0,13)	

After the instruction is executed, register 7 contains 00 00 63 FA = + 25, 594_{10}.

Convert to Decimal (CVD)

The CONVERT TO DECIMAL instruction performs functions exactly opposite to those of the CONVERT TO BINARY instruction. CVD converts a binary number in a

register to packed decimal and stores the result in a double word. For example, assume:

Register 1 contains 00 00 0F 0F = 3855_{10}
Register 13 contains 00 00 76 00
PSW bit 12 = 0 (EBCDIC mode)

The format of the conversion instruction is:

Machine Format

OP CODE	R1	X2	B2	D2
4E	1	0	D	008

Assembler Format

OP CODE	R1D2X2 B2
CVD	1,8(0,13)

After the instruction is executed, location 7608-760F contain 00 00 00 00 00 03 85 5+. The plus sign generated is the standard EBCDIC plus sign, 1100_2.

Shift Left Single (SLA)

Because the sign bit remains unchanged during an SLA operation, this instruction performs an *algebraic* shift. For example, if the contents of register 2 are:

00 7F 0A 72 = 0000 0000 0111 1111 0000 1010 0111 0010_2

the instruction:

Machine Format

OP CODE	R1	R3	B2	D2
8B	2	//////	0	008

Assembler Format

OP CODE	R1D2 B2
SLA	2,8(0)

results in register 2 being shifted left 8 places so that its new contents are:

7F 0A 72 00 = 0111 1111 0000 1010 0111 0010 0000 0000_2

If a left shift of 9 places had been specified, a significant bit would have been shifted out of position 1, and a fixed-point overflow interruption might have occurred (unless PSW bit 36 equaled 0).

Note that register 0 does not participate in the operation and that the contents of the R_3 field are ignored.

Shift Left Double (SLDA)

The SHIFT LEFT DOUBLE instruction is similar to SHIFT LEFT SINGLE except that SLDA shifts the 63 bits (not including the sign) of an even/odd register pair. The R_1 field of this instruction must be even. For example, if the contents of registers 2 and 3 are:

00 7F 0A 72 FE DC BA 98 =
0000 0000 0111 1111 0000 1010 0111 0010 1111 1110 1101 1100 1011 1010 1001 1000_2

the instruction:

Machine Format

OP CODE	R1	R3	B2	D2
8F	2	//////	0	01F

Assembler Format

OP CODE	R1D2 B2
SLDA	2,31(0)

results in registers 2 and 3 both being left-shifted 31 places, so that their new contents are:

7F 6E 5D 4C 00 00 00 00 =
0111 1111 0110 1110 0101 1101 0100 1100 0000 0000 0000 0000 0000 0000 0000 0000_2

In this case, a significant bit is shifted out of position 1, and a fixed-point overflow interruption occurs (unless PSW bit 36 equals 0).

Store Multiple (STM)

Assume that the contents of general registers 14, 15, 0, and 1 are to be stored in consecutive words starting with storage location 4050 and that:

Register 14 contains 00 00 25 63
Register 15 contains 00 01 27 36
Register 0 contains 12 43 00 62
Register 1 contains 73 26 12 57
Register 6 contains 00 00 40 00
The initial contents of locations 4050-405F are not significant

The STORE MULTIPLE instruction allows the use of just one instruction to store the contents of the four registers when it is written as:

Machine Format

OP CODE	R1	R3	B2	D2
90	E	1	6	050

Assembler Format

OP CODE	R1 R3 D2 B2
STM	14,1,X'50'(6)

After the instruction is executed:

Locations 4050-4053 contain 00 00 25 63
Locations 4054-4057 contain 00 01 27 36
Locations 4058-405B contain 12 43 00 62
Locations 405C-405F contain 73 26 12 57

Logical Operations

Move (MVI, MVC)

Move Immediate (MVI)

The MOVE IMMEDIATE instruction can place one byte of information from the instruction stream into any designated location in storage. For example, if the instruction:

Machine Format

OP CODE	I2	B1	D1
92	FA	0	055

Assembler Format

OP CODE	D1 B1	I2
MVI	85(0),	X'FA'

is executed, bits 8-15 of the instruction ($1111\ 1010_2$) are copied in storage location 85_{10}.

Move Characters (MVC)

The MVC instruction can be used to move a data field from one location in storage to another. For example, assume that the following two fields are in storage:

	2048										2052
Field 1	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB

	3840								3848
Field 2	F1	F2	F3	F4	F5	F6	F7	F8	F9

Also assume:

Register 1 contains 00 00 20 48
Register 2 contains 00 00 38 40

With the following instruction the first eight bytes of field 2 replace the first eight bytes of field 1:

Machine Format

OP CODE	L	B1	D1	B2	D2
D2	07	1	000	2	000

Assembler Format

OP CODE	D1	L	B1	D2	B2
MVC	0(8,1),			0(2)	

After the instruction is executed, field 1 becomes:

	2048									2052
Field 1	F1	F2	F3	F4	F5	F6	F7	F8	C9	CA	CB

Field 2 is unchanged.

As indicated in the programming note in the MOVE instruction description, MVC can be used to propagate one character through a field by starting the first operand field one byte to the right of the second operand field. For example, suppose that an area in storage starting with address 358 contains the following data:

358							360
00	F1	F2	F3	F4	F5	F6	F7	F8

With the following MVC instruction, the zeros in location 358 can be propagated throughout the entire field (assume that register 11 contains 00 00 03 58):

Machine Format

OP CODE	L	B1	D1	B2	D2
D2	07	B	001	B	000

Assembler Format

OP CODE	D1	L	B1	D2	B2
MVC	1(8,11),			0(11)	

Because the MVC handles one byte at a time, the above instruction essentially takes the byte at address 358 and stores it at 359 (359 now contains 00), takes the byte at 359 and stores it at 35A, etc., until the entire field is filled with zeros. Note that an MVI instruction could have originally been used to place the byte of zeros in location 358.

NOTES:

1. Although the field occupying locations 358-360 contains nine bytes, the length coded in the assembler format is equal to the number of moves (one less than the field length).
2. The order of operands is important even though only one field is involved.

Move Numerics (MVN)

To illustrate the operation of the MOVE NUMERICS instruction, assume that the following two fields are in storage:

	7090							7097
Field 1	C1	C2	C3	C4	C5	C6	C7	C8

	7041							7049
Field 2	F0	F1	F2	F3	F4	F5	F6	F7	F8

Also assume:

Register 14 contains 00 00 70 90
Register 15 contains 00 00 70 40

After the instruction:

Machine Format

OP CODE	L	B1	D1	B2	D2
D1	03	F	001	E	000

Assembler Format

OP CODE	D1	L	B1	D2	B2
MVN	1(4,15),			0(14)	

is executed, field 2 becomes:

7041							7049
F1	F2	F3	F4	F4	F5	F6	F7	F8

The numeric portions of locations 7090-7093 have been stored in the numeric portions of locations 7041-7044. The contents of locations 7090-7097 and 7045-7049 are unchanged.

Move Zones (MVZ)

The MOVE ZONES instruction, similar to MVC and MVN, can operate on overlapping or nonoverlapping fields. (See the examples for MVC and MVN.) When operating on nonoverlapping fields, MVZ works similar to the MVN instruction in the previous example, except that the MVZ moves the high-order four bits of each byte. To illustrate the use of MVZ with overlapping fields, assume that the following data field is in storage:

800					805
F1	C2	F3	C4	F5	C6

Also assume that register 15 contains 00 00 08 00. The instruction:

Machine Format

OP CODE	L	B1	D1	B2	D2
D3	04	F	001	F	000

Assembler Format

OP CODE	D1	L	B1	D2	B2
MVZ	1(5,15),			0(15)	

propagates the zone from the byte at address 800 through the data field, so that the field becomes:

800					805
F1	F2	F3	F4	F5	F6

Compare Logical (CL, CLR, CLI, CLC)

The COMPARE LOGICAL instructions differ from the algebraic compare instructions (C, CR) in that all quantities are handled as if unsigned.

Compare Logical Registers (CLR)

Assume that:

Register 1 contains 00 00 00 01
Register 2 contains FF FF FF FF

Execution of the instruction:

Machine Format

OP CODE	R1	R2
15	1	2

Assembler Format

OP CODE	R1R2
CLR	1,2

sets the condition code to 1. A condition code of 1 indicates that the first operand is lower than the second. However, if an *algebraic* compare instruction had been executed, the condition code would have been set to 2, indicating that the first operand is higher. During algebraic comparison, the contents of register 1 are interpreted as +1 and the contents of register 2 as −1. During logical comparison, the leftmost byte of register 2 is compared with the leftmost byte of register 1; each byte is interpreted as a binary number. In this case:

Leftmost byte of register 1: $0000\ 0000_2 = 0_{10}$
Leftmost byte of register 2: $1111\ 1111_2 = 255_{10}$

If the two leftmost bytes are equal, the next two bytes will be compared, etc., until either an inequality is discovered or the contents of the registers are exhausted.

Compare Logical Immediate (CLI)

The CLI instruction logically compares a byte from the instruction stream with a byte from storage. For example, assume that:

Register 10 contains 00 00 17 00
Storage location 1703 contains 7E

Execution of the instruction:

Machine Format

OP CODE	I2	B1	D1
95	AF	A	003

Assembler Format

OP CODE	D1 B1	I2
CLI	3(10),	X'AF'

sets the condition code to 1, indicating that the first operand (the quantity in main storage) is lower than the second (immediate) operand.

Compare Logical Characters (CLC)

The COMPARE LOGICAL CHARACTERS instruction can be used to perform the logical comparison of storage fields up to 256 bytes in length. For example, assume that the following two fields of data are in storage:

Field 1

1886										1891
D1	D6	C8	D5	E2	D6	D5	6B	C1	4B	C2	4B

Field 2

1900										190B
D1	D6	C8	D5	E2	D6	D5	6B	C1	4B	C3	4B

Also assume:

Register 6 contains 00 00 18 80
Register 7 contains 00 00 19 00.

Execution of the instruction:

Machine Format

OP CODE	L	B1	D1	B2	D2
D5	0B	6	006	7	000

Assembler Format

OP CODE	D1 L	B1 D2	B2
CLC	6(12,6),	0(7)	

sets the condition code to 1, indicating that field 1 is lower than field 2.

Because CLC compares bytes on an unsigned binary basis, the instruction can be used to collate fields composed of characters from the EBCDIC code. For example, in EBCDIC, the above two data fields are:

Field 1 JOHNSON,A.B.
Field 2 JOHNSON,A.C.

The condition code of 1 tells us that A. B. Johnson precedes A. C. Johnson, thus placing the names in the correct alphabetic order.

AND (N, NR, NI, NC)

When the Boolean operator AND is applied to two bits, the result is 1 when both bits are 1; otherwise, the result is 0. When two bytes are AND'ed in System/360, each pair of bits is handled separately; there is no connection from one bit position to another.

AND (NI)

A frequent use of the AND instruction is to set a particular bit to zero. For example, assume that storage location 4891 contains $0100\ 0011_2$. To set the eighth (rightmost) bit of this byte to 0 without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

OP CODE	I2	B1	D1
94	FE	8	001

Assembler Format

OP CODE	D1 B1	I2
NI	1(8),	X'FE'

When this instruction is executed, the byte in storage is AND'ed with the immediate byte:

Location 4891:	$0100\ 0011_2$
Immediate byte:	$1111\ 1110_2$
Result:	$0100\ 0010_2$

The resulting byte with bit seven set to 0 is stored in location 4891. The condition code is set to 1.

OR (O, OR, OI, OC)

When the Boolean operator OR is applied to two bits, the result is 1 when either bit is 1; otherwise, the result is 0. When two bytes are OR'ed in System/360, each pair of bits is handled separately; there is no connection from one bit position to another.

OR (OI)

A frequent use of the OR instruction is to set a particular bit to 1. For example, assume that storage location 4891 contains 0100 0010_2. To set the eighth (rightmost) bit of this byte to 1 without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

OP CODE	I2	B1	D1
96	01	8	001

Assembler Format

OP CODE	D1 B1	I2
OI	1(8),X'01'	

When this instruction is executed, the byte in storage is OR'ed with the immediate byte:

Location 4891:	0100 0010_2
Immediate byte:	0000 0001_2
Result:	0100 0011_2

The resulting byte with bit seven set to 1 is stored in location 4891. The condition code is set to 1.

Exclusive OR (X, XR, XI, XC)

When the Boolean operator exclusive OR is applied to two bits, the result is 1 when one, and only one, of the two bits is 1; otherwise, the result is 0. When two bytes are exclusive OR'ed in System/360, each pair of bits is handled separately; there is no connection from one bit position to another.

Exclusive OR (XI)

A frequent use of the EXCLUSIVE OR (XI) instruction is to invert a bit (change a 0 bit to a 1 or a 1 bit to a 0). For example, assume that storage location 8082 contains 0110 1001_2. To set the leftmost bit to 1 and the rightmost bit to 0 without affecting any of the other bits, the following instruction can be used (assume that register 9 contains 00 00 80 80):

Machine Format

OP CODE	I2	B1	D1
97	81	9	002

Assembler Format

OP CODE	D1 B1	I2
XI	2(9),X'81'	

When the instruction is executed, the byte in storage is exclusive OR'ed with the immediate byte:

Location 8082:	0110 1001_2
Immediate byte:	1000 0001_2
Result:	1110 1000_2

The resulting byte with the leftmost and rightmost bits inverted is stored in location 8082. The condition code is set to 1.

Exclusive OR (XC)

The EXCLUSIVE OR (XC) instruction can be used to exchange the contents of two areas in storage without the use of an intermediate storage area. For example, assume that two words are in storage:

	358			35B
Word 1	00	00	17	90

	360			363
Word 2	00	00	14	01

Execution of the instruction (assume that register 7 contains 00 00 03 58):

Machine Format

OP CODE	L	B1	D1	B2	D2
D7	03	7	000	7	008

Assembler Format

OP CODE	D1 L	B1 D2	B2
XC	0(4,7),8(7)		

exclusive OR's word 1 with word 2 as follows:

Word 1:	0000 0000 0000 0000 0001 0111 1001 0000_2 = 00 00 17 90
Word 2:	0000 0000 0000 0000 0001 0100 0000 0001_2 = 00 00 14 01
Result:	0000 0000 0000 0000 0000 0011 1001 0001_2 = 00 00 03 91

The result replaces the former contents of word 1.

Now, execution of the instruction:

Machine Format

OP CODE	L	B1	D1	B2	D2
D7	03	7	008	7	000

Assembler Format

OP CODE	D1 L	B1 D2	B2
XC	8(4,7),0(7)		

produces the following result:

Word 1:	0000 0000 0000 0000 0000 0011 1001 0001_2 = 00 00 03 91
Word 2:	0000 0000 0000 0000 0001 0100 0000 0001_2 = 00 00 14 01
Result:	0000 0000 0000 0000 0001 0111 1001 0000_2 = 00 00 17 90

The result of this operation replaces the former contents of word 2. Word 2 now contains the original value of word 1.

Lastly, execution of the instruction:

Machine Format

OP CODE	L	B1	D1	B2	D2
D7	03	7	000	7	008

Assembler Format

OP CODE	D1 L	B1 D2	B2
XC	0(4,7),8(7)		

produces the following result:

Word 1:	0000 0000 0000 0000 0000 0011 1001 0001_2 = 00 00 03 91
Word 2:	0000 0000 0000 0000 0001 0111 1001 0000_2 = 00 00 17 90
Result:	0000 0000 0000 0000 0001 0100 0000 0001_2 = 00 00 14 01

The result of this operation replaces the former contents of word 1. Word 1 now contains the original value of word 2.

NOTES:

1. With the XC instruction, fields up to 256 bytes in length can be exchanged.
2. With the XR instruction, the contents of two registers can be exchanged.
3. Because the x instruction operates storage to register only, an exchange cannot be made solely by the use of x.

4. A field exclusive OR'ed with itself is cleared to zeros.

Test Under Mask (TM)

The TEST UNDER MASK instruction examines specific bits within a byte and sets the condition code according to what it finds. For example, assume that:

Storage location 9999 contains FB
Register 9 contains 00 00 99 90

Execution of the instruction:

Machine Format

OP CODE	I2	B1	D1
91	C3	9	009

Assembler Format

OP CODE	D1 B1	I2
TM	9(9),	X'C3'

produces the following result:

FB = 1111 1011$_2$
Mask (C3) = 1100 0011$_2$
Result = 11xx xx11

The condition code is set to 3: all selected bits are ones.

If location 9999 had contained B9, the result would have been:

B9 = 1011 1001$_2$
Mask (C3) = 1100 0011$_2$
Result = 10xx xx01

The condition code is set to 1: the selected bits are both zeros and ones.

If location 9999 had contained 3C, the result would have been:

3C = 0011 1100$_2$
Mask (C3) = 1100 0011$_2$
Result = 00xx xx00

The condition code is set to 0: all selected bits are zeros.

NOTE: Storage location 9999 remains unchanged.

Load Address (LA)

The LOAD ADDRESS instruction provides a convenient way to place a non-negative number $\leq 4095_{10}$ in a register without first defining the number as a constant and then using it as an operand. For example, assume that the number 2048_{10} is to be placed in register 1. One instruction that will do this is:

Machine Format

OP CODE	R1	X2	B2	D2
41	1	0	0	800

Assembler Format

OP CODE	R1	D2	X2B2
LA	1,	2048	(0,0)

As indicated in the programming note in the instruction description, the LOAD ADDRESS instruction can also be used to increment a register by an amount $\leq 4095_{10}$ specified in the D_2 field. For example, assume that register 5 contains 00 12 34 56.

The instruction:

Machine Format

OP CODE	R1	X2	B2	D2
41	5	0	5	00A

Assembler Format

OP CODE	R1	D2	X2B2
LA	5,	10	(0,5)

adds 10 (decimal) to the contents of register 5 as follows:

Register 5 (old): 00 12 34 56
D_2: 00 00 00 0A
Register 5 (new): 00 12 34 60

Translate (TR)

With the TRANSLATE instruction, System/360 can translate data from any code to any other desired code, provided that each coded character consists of eight bits or fewer. In the following example EBCDIC is translated to USASCII-8. The first step is to create a 256-byte table in storage locations 1000-10FF. This table contains the characters of the code into which you are translating (the function bytes). The table must be in order, not by the binary values of the characters it contains, but by the binary sequence of the characters of the original code (the argument bytes). For example, note in the table below that the characters are in the normal EBCDIC collating sequence.

Translate Table

	100F
1000	
1010	
1020	
1030	
1040	b . (. +
1050	&. $ *)
1060	¬ / , %
1070	# @ ¬ =
1080	a b c d e f g h i
1090	j k l m n o p q r
10A0	s t u v w x y z
10B0	
10C0	A B C D E F G H I
10D0	J K L M N O P Q R
10E0	S T U V W X Y Z
10F0	0 1 2 3 4 5 6 7 8 9
	10FF

Notes:

1. The overbars are used to indicate the USASCII-8 representations of the EBCDIC characters shown.

2. If the character codes in the statement being translated occupy a range smaller than 00_{16} through FF_{16}, a table of less than 256 bytes can be used.

3. The symbol in location 1040 represents the coding for a blank, which is the same in both EBCDIC and USASCII-8, 40_{16}.

Now, assume that starting at storage location 2100 there is a sequence of 20_{10} EBCDIC characters to be translated to USASCII-8:

Locations 2100-2113: JOHNbJONESb257bW.b95

Also assume:

Register 12 contains 00 00 21 00
Register 15 contains 00 00 10 00

As the instruction:

Machine Format

OP CODE	L	B1	D1	B2	D2
DC	13	C	000	F	000

Assembler Format

OP CODE	D1	L	B1	D2	B2
TR	0(20,12),0(15)				

is executed, the binary value of each argument byte is added to the starting address of the table, and the resulting address is used to fetch a function byte:

Table starting address:	1000
First argument byte (J):	D1
Address of function byte:	10D1

Because the table is arranged so that every EBCDIC character is replaced by the corresponding USASCII-8 character, the result is:

Locations 2100-2113: JOHNbJONESb257bW.b95

NOTE: To verify that this example is correct, find in Appendix F the hexadecimal values for the remaining EBCDIC characters and add them to the starting address of the table (1000). The sums should be the addresses within the table of the corresponding USASCII-8 characters.

Translate and Test (TRT)

The TRANSLATE AND TEST instruction is used to scan a data field (the argument bytes) for characters with special meaning. To indicate which characters have special meaning, first set up a table similar to the one used for the TRANSLATE instruction. (See the preceding example.) Once again the table must be in order by the binary sequence of the code of the argument bytes. This time, however, put zeros in the table to indicate characters without any special meaning and nonzero values to indicate characters with special meaning.

This example deals with EBCDIC characters; the characters with special meaning in the argument field are a selected set of punctuation marks. The translate and test table that follows has been set up accordingly.

Now, assume that starting at storage location 3000 you have the following sequence of 30_{10} EBCDIC characters:

Locations 3000-301D:
bbbbbUNPKbbbbbPROUT(9),WORD(5)

Translate and Test Table

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2040	00	00	00	00	00	00	00	00	00	00	00	00	10	20	25	00
2050	90	00	00	00	00	00	00	00	00	00	00	30	35	40	45	00
2060	80	85	00	00	00	00	00	00	00	00	00	50	55	00	00	00
2070	00	00	00	00	00	00	00	00	00	00	00	60	65	70	75	00
2080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Note: If the character codes in the statement being translated occupy a range smaller than 00_{16} through FF_{16}, a table less than 256 bytes can be used.

Also assume:

Register 1 contains 00 00 2F FF
Register 2 contains 00 00 00 00
Register 15 contains 00 00 20 00

As the instruction:

Machine Format

OP CODE	L	B1	D1	B2	D2
DD	1D	1	001	F	000

Assembler Format

OP CODE	D1	L	B1	D2	B2
TRT	1(30,1),0(15)				

is executed, the value of the first argument byte, a blank, is added to the starting address of the table to produce the address of the function byte to be examined:

Table starting address	2000
First argument byte (blank)	40
Address of function byte	2040

Because zeros were originally placed in storage location 2040, no special action occurs, and the operation continues with the second argument byte. The operation will thus continue until it reaches the symbol ((left parenthesis) in location 3013. When this symbol is reached, its value is added to the starting address of the table, as usual:

Table starting address	2000
Argument byte (left parenthesis)	4D
Address of function byte	204D

Because location 204D contains a nonzero value, the following actions occur:

1. The address of the argument byte, 003013, is placed in the low-order 24 bits of register 1.
2. The function byte, 20, is placed in the low-order eight bits of register 2.
3. The condition code is set to 1 (scan not completed).

In general, TRANSLATE AND TEST is executed by use of an EXECUTE instruction, which supplies the length specification from a general register. In this way a complete statement scan can be performed with a single TRANSLATE AND TEST instruction repeated over and over by means of EXECUTE. In the example, after the first execution of TRT, register 1 contains the address of the last argument byte translated. It is then a simple matter to subtract this address from the address of the last argument byte (301D) to produce a length specification. This length minus one is placed in the register that is referenced as the R_1 field of the EXECUTE instruction. (Because the length code in the machine format is one less than the total number of bytes in the field, one must be subtracted from the computed length.) The branch address part of the EXECUTE instruction points to the TRANSLATE AND TEST instruction, which must now appear in the following format:

Machine Format

OP CODE	L	B1	D1	B2	D2
DD	00	1	001	F	000

Assembler Format

OP CODE	D1 LB1	D2 B2
TRT	1(0,1),0(15)	

Now the entire argument field can be scanned, stopping to examine those characters of special interest, without having to modify any of the instructions already written. After a stop is made to examine a character, only a new length need be computed before continuing the scan.

Edit (ED)

Because the decimal feature instructions operate only on packed decimal data, it is necessary to convert the data to the zoned format before a legible report can be printed. Moreover, if the report is to be useful to a great many people, certain punctuation marks, such as commas and decimal points, should be inserted in appropriate places. The highly flexible EDIT instruction performs these two functions in a single execution.

This example shows step-by-step one way in which EDIT can be used. The field to be edited (the source) is four bytes long; it is edited against a pattern 13 bytes long. The following symbols are used:

SYMBOL	MEANING
b (hexadecimal 40)	blank character
((hexadecimal 21)	significance starter
d (hexadecimal 20)	digit selector

Assume that the source and pattern fields are:

Source

1200			1203
02	57	42	6C

+

Pattern

1000											100C
40	20	20	6B	20	20	21	4B	20	20	40	C3	D9
b	d	d	,	d	d	(	.	d	d	b	C	R

Execution of the instruction (assume that register 12 contains 00 00 10 00):

Machine Format

OP CODE	L	B1	D1	B2	D2
DE	0C	C	000	C	200

Assembler Format

OP CODE	D1 L	B1	D2	B2
ED	0(13,12),	X'200'	(12)	

alters the pattern as follows:

PATTERN	DIGIT	SIGNIFICANCE INDICATOR before/after	RULE	LOCATION 1000-100C
b		off/off	leave(1)	bdd,dd(.ddbCR
d	0	off/off	fill	bbd,dd(.ddbCR
d	2	off/on (2)	digit	bb2,dd(.ddbCR
,		on/on	leave	same
d	5	on/on	digit	bb2,5d(.ddbCR
d	7	on/on	digit	bb2,57(.ddbCR
(	4	on/on	digit	bb2,574.ddbCR
.		on/on	leave	same
d	2	on/on	digit	bb2,574.2dbCR
d	6+	on/off (3)	digit	bb2,574.26bCR
b		off/off	fill	same
C		off/off	fill	bb2,574.26bbR
R		off/off	fill	bb2,574.26bbb

Notes:

1. This character becomes the fill character.
2. First nonzero decimal source digit turns on significance indicator.
3. Plus sign in the four low-order bits of the byte turns off significance indicator.

Thus, after the instruction is executed:

Source Unchanged

Pattern

1000												100C
40	40	F2	6B	F5	F7	F4	4B	F2	F6	40	40	40
b	b	2	,	5	7	4		2	6	b	b	b

Condition code = 2: result greater than zero.

When printed, the pattern field, which now contains the result, appears as:

2,574.26

If the number in the source field is changed to 00 00 02 6D, a negative number, and the original pattern is used, the edited result becomes:

Pattern

1000											100C
40	40	40	40	40	40	40	4B	F2	F6	C3	D9

b b b b b b b . 2 6 b C R

Condition code = 1; result less than zero

The significance starter forces the significance indicator to the on state and hence causes the decimal point to be left unchanged. Because the minus sign does not change the significance indicator, the CR symbol is also preserved.

Edit and Mark (EDMK)

After an edit-and-mark operation, a symbol (such as a dollar sign) can be inserted at the appropriate position in the edited result. Usually a currency symbol is inserted to the immediate left of the first significant digit in the amount; however, if a decimal point appears in an amount less than one, the currency symbol must be inserted to the immediate left of the decimal point. A typical operation would leave no blank between the currency symbol and the amount, thus protecting against one form of alteration when the result is printed on a check.

If significance is not forced by the significance starter, the edit-and-mark operation inserts into general register 1 an address one more than the address at which a currency symbol would normally be inserted. After one is subtracted from the value in general register 1 (for example, by using a BRANCH ON COUNT instruction with R_1 set to one and R_2 set to zero), a MOVE instruction (MVI) may be used to position the symbol in main storage.

Machine Format

OP CODE	I_2	B_1	D_1
92	5B	1	000

Assembler Format

OP CODE	D_1 B_1	I_2
MVI	0(1),	C'$'

If significance is forced, general register 1 remains unchanged. Therefore, the address of the character following the significance starter should be placed in that register before the EDIT AND MARK instruction is performed.

Decimal Arithmetic

Add Decimal (AP)

Assume that the signed, packed-decimal field at storage locations 500-503 is to be added to the signed, packed-decimal field at locations 2000-2002. Also assume:

Register 12 contains 00 00 20 00
Register 13 contains 00 00 04 FD
Storage locations 2000-2002 contain 38 46 0D (a neg number)
Storage locations 500-503 contain 01 12 34 5C (a pos number)

After the instruction:

Machine Format

OP CODE	L_1	L_2	B_1	D_1	B_2	D_2
FA	2	3	C	000	D	003

Assembler Format

OP CODE	D_1 L_1 B_1	D_2 L_2 B_2
AP	0(3,12),	3(4,13)

is executed, the storage locations 2000-2002 contain 73 88 5C; the condition code is set to 2 to indicate that the sum is positive. Note that:

1. Although the second operand field is larger than the first operand field, no overflow interruption occurs because the result can be entirely contained within the first operand field.

2. Because the two numbers had different signs, they were in effect subtracted.

Zero and Add (ZAP)

Assume that the signed, packed-decimal field at storage locations 4500-4502 is to be moved to locations 4000-4004 with four leading zeros in the result field. Also assume:

Register 9 contains 00 00 40 00
Storage locations 4000-4004 contain 12 34 56 78 90
Storage locations 4500-4502 contain 38 46 0D

After the instruction:

Machine Format

OP CODE	L_1	L_2	B_1	D_1	B_2	D_2
F8	4	2	9	000	9	500

Assembler Format

OP CODE	D_1 L_1 B_1	D_2 L_2 B_2
ZAP	0(5,9),	X'500'(3,9)

is executed, the storage locations 4000-4004 contain 00 00 38 46 0D; the condition code is set to 1 to indicate a negative result. Note that because the first operand is not checked for valid sign and digit codes, it may contain any combination of hexadecimal digits.

Compare Decimal (CP)

Assume that the signed, packed-decimal contents of storage locations 700-703 are to be algebraically compared with the signed, packed-decimal contents of locations 500-503. Also assume:

Register 12 contains 00 00 06 00
Register 13 contains 00 00 04 00
Storage locations 700-703 contain 17 25 35 6D
Storage locations 500-503 contain 06 72 14 2D

After the instruction:

Machine Format

OP CODE	L_1	L_2	B_1	D_1	B_2	D_2
F9	3	3	C	100	D	100

Assembler Format

OP CODE	D1 L1 B1	D2 L2 B2
CP	X'100'(4,12),	X'100'(4,13)

is executed, the condition code is set to 1, indicating that the first operand (the contents of locations 700-703) is lower than the second.

Multiply Decimal (MP)

Assume that the signed, packed-decimal field in storage locations 1202-1204 (the multiplicand) is to be multiplied by the signed, packed-decimal field in locations 500-501 (the multiplier):

	1202		1204
Multiplicand	38	46	0D

	500	501
Multiplier	32	1D

Because there are a total of eight significant digits in the multiplier and multiplicand, a field at least five bytes in length must be reserved for the signed result. As indicated in the programming note for MULTIPLY DECIMAL, a ZERO AND ADD into a larger field can provide the required space. If it is assumed:

Register 4 contains 00 00 12 00
Register 6 contains 00 00 05 00

then execution of the assembler instruction:

```
ZAP   X'100'(5,4),2(3,4)
```

sets up a new multiplicand in storage locations 1300-1304:

	1300				1304
Multiplicand (new)	00	00	38	46	0D

Now, after the instruction:

Machine Format

OP CODE	L1	L2	B1	D1	B2	D2
FC	4	1	4	100	6	000

Assembler Format

OP CODE	D1	L1 B1	D2	L2 B2
MP	X'100'(5,4),	0(2,6)		

is executed, storage locations 1300-1304 contain the product: 01 23 45 66 0C.

Divide Decimal (DP)

Assume that the signed, packed-decimal field at storage locations 2000-2004 (the dividend) is to be divided by the signed, packed-decimal field at locations 3000-3001 (the divisor). Also assume:

Register 12 contains 00 00 20 00
Register 13 contains 00 00 30 00
Storage locations 2000-2004 contain 01 23 45 67 8C
Storage locations 3000-3001 contain 32 1D

After the instruction:

Machine Format

OP CODE	L1	L2	B1	D1	B2	D2
FD	4	1	C	000	D	000

Assembler Format

OP CODE	D1 L1 B1	D2 L2 B2
DP	0(5,12),	0(2,13)

is executed, the dividend field is entirely replaced by the signed quotient and remainder fields, as follows:

	2000				2004
Locations 2000-2004	38	46	0D	01	8C
			quotient	remainder	

NOTES:

1. Because the signs of the dividend and divisor are different, the quotient receives a negative sign.
2. The remainder receives the sign of the dividend and the length of the divisor.
3. If an attempt is made to divide the dividend by the one-byte field at location 3001, the quotient will be too long to fit within the four bytes allotted to it. A decimal-divide exception exists, causing a program interruption.

Pack (PACK)

Assume that storage locations 1000-1004 contain the following zoned-decimal field that is to be converted to a packed-decimal field and left in the same location:

	1000				1004
Zoned Field	F1	F2	F3	F4	C5

Also assume that register 12 contains 00 00 10 00. After the instruction:

Machine Format

OP CODE	L1	L2	B1	D1	B2	D2
F2	4	4	C	000	C	000

Assembler Format

OP CODE	D1 L1 B1	D2 L2 B2
PACK	0(5,12),	0(5,12)

is executed, the field in locations 1000-1004 is in the packed-decimal format:

	1000				1004
Packed Field	00	00	12	34	5C

NOTES:

1. This example illustrates the operation of PACK when the first and second operand fields overlap completely.
2. During the operation, the second operand was extended with high-order zeros.

Unpack (UNPK)

Assume that storage locations 2501-2503 contain a signed, packed-decimal field that is to be unpacked and placed in storage locations 1000-1004. Also assume:

Register 12 contains 00 00 10 00
Register 13 contains 00 00 25 00
Storage locations 2501-2503 contain 12 34 5D
The initial contents of storage locations 1000-1004 are not significant
PSW bit 12 = 0 (EBCDIC mode)

After the instruction:

Machine Format

OP CODE	L1	L2	B1	D1	B2	D2
F3	4	2	C	000	D	001

Assembler Format

OP CODE	D1 L1	B1 D2	L2 B2
UNPK	0(5,12),	1(3,13)	

is executed, the storage locations 1000-1004 contain F1 F2 F3 F4 D5. Because the CPU was in EBCDIC mode, the zone $1111_2 = F_{16}$ was attached to all digits except the digit occupying the same byte as the sign.

Move with Offset (MVO)

Assume that the unsigned three-byte field in storage locations 4500-4502 is to be moved to locations 5600-5603 and given the sign of the one-byte field located at 5603. Also assume:

Register 12 contains 00 00 56 00
Register 15 contains 00 00 45 00
Storage locations 5600-5603 contain 77 88 99 0C
Storage locations 4500-4502 contain 12 34 56

After the instruction:

Machine Format

OP CODE	L1	L2	B1	D1	B2	D2
F1	3	2	C	000	F	000

Assembler Format

OP CODE	D1 L1	B1 D2	L2 B2
MVO	0(4,12),	0(3,15)	

is executed, the storage locations 5600-5603 contain 01 23 45 6C. Note that the second operand was extended with one high-order zero to fill out the first operand field.

NOTE: The section "Shifting of Decimal Fields" shows how MOVE WITH OFFSET can be used in shifting a decimal field an odd number of places.

Shifting of Decimal Fields

No instructions have been specifically provided to perform shifting of decimal fields in storage. However, various combinations of System/360 instructions may be used to accomplish in effect this type of shift. The following assembly-language examples illustrate some of the methods for shifting decimal numbers. These examples additionally illustrate how the assembly language facilitates coding with symbolic operands.

Decimal Right Shift (Even Number of Places)

Assume that symbolic storage location SOURCE contains 12 34 56 78 9C, and you wish to shift the contents of SOURCE two places to the right (to drop the rightmost two digits, thereby dividing SOURCE by 100_{10}). The MOVE NUMERICS (MVN) instruction can be used to accomplish this:

```
MVN   SOURCE+3(1),SOURCE+4
```

After the MVN instruction is executed, SOURCE contains 12 34 56 7C.9C. Instructions referencing SOURCE should now use a length of 4 instead of 5.

Decimal Right Shift (Odd Number of Places)

Assume that symbolic storage location SOURCE contains 12 34 56 78 9C, and you wish to shift the contents of SOURCE three places to the right (to drop the rightmost three digits, thereby dividing SOURCE by 1000_{10}). The MOVE WITH OFFSET (MVO) instruction can be used to accomplish this:

```
MVO   SOURCE(5),SOURCE(3)
```

After this instruction is executed, SOURCE contains 00 01 23 45 6C.

Decimal Left Shift (Even Number of Places)

Assume that symbolic location ZERO contains 00 00 and that SOURCE contains 12 34 56 78 9C. The contents of SOURCE can be shifted four places to the left by using the following group of instructions:

```
                                    SOURCE
MVC   SOURCE+5(2),ZERO              12 34 56 78 9C 00 00
MVN   SOURCE+6(1),SOURCE+4          12 34 56 78 9C 00 0C
NI    SOURCE+4,240                  12 34 56 78 90 00 0C
```

Note that the number 240_{10} in the AND (NI) instruction provides a mask of $1111\ 0000_2$, which is used to make the old sign position zero.

Decimal Left Shift (Odd Number of Places)

Assume that symbolic location ZERO contains 00 00 and that SOURCE contains 12 34 56 78 9C. The contents of SOURCE can be shifted three places to the left by using the following group of instructions:

```
                                    SOURCE
MVC   SOURCE+5(2),ZERO              12 34 56 78 9C 00 00
MVN   SOURCE+6(1),SOURCE+4          12 34 56 78 9C 00 0C
NI    SOURCE+4,240                  12 34 56 78 90 00 0C
MVO   SOURCE(6),SOURCE(5)           01 23 45 67 89 00 0C
```

Floating-Point Arithmetic

In this section, the abbreviations FPR0, FPR2, FPR4, and FPR6 stand for floating-point registers 0, 2, 4, and 6, respectively.

Add Normalized (AE, AER, AD, ADR)

The ADD NORMALIZED instructions perform the addition of two floating-point numbers and place the normalized result in a floating-point register. Neither of the two numbers to be added must necessarily be normalized before addition occurs. For example, assume that:

FPR6 contains 43 08 21 00 00 00 00 00 = $82.1_{16} \cong 130.06_{10}$ (unnormalized)

Storage locations 2000-2007 contain 41 12 34 56 00 00 00 00 = $1.23456_{16} \cong 1.13_{10}$ (normalized)

Register 13 contains 00 00 20 00

The instruction:

Machine Format

OP CODE	R1	X2	B2	D2
7A	6	0	D	000

Assembler Format

OP CODE	R1D2 X2B2
AE	6,0(0,13)

can be used to perform the short-precision addition of the two operands. In this example the instruction operates as follows:

The characteristics of the two numbers are compared. Since the number in storage has a characteristic that is smaller by 2, it is right-shifted after fetching until the characteristics agree. The two numbers are then added:

		GUARD DIGIT
FPR6:	43 08 21 00	
Shifted number from storage:	43 00 12 34	5
Intermediate sum:	43 08 33 34	5

Because the intermediate sum is unnormalized, it is left-shifted to form the normalized floating-point number 42 83 33 45 ($=83.3345_{16} = 131.2_{10}$). This number replaces the high-order portion of FPR6. The low-order portion of FPR6 and the contents of storage locations 2000-2007 are unchanged.

If the long-precision instruction AD is used, the result in FPR6 will be 42 83 33 45 60 00 00 00. Note that in this case, the use of the long-precision instruction provides one additional hexadecimal digit of precision.

Add Unnormalized (AU, AUR, AW, AWR)

The ADD UNNORMALIZED instructions operate identically to the ADD NORMALIZED instructions, except that the final result is not normalized when ADD UNNORMALIZED is used. For example, using the same operands as in the example for ADD NORMALIZED, when the short-precision instruction:

Machine Format

OP CODE	R1	X2	B2	D2
7E	6	0	D	000

Assembler Format

OP CODE	R1D2 X2B2
AU	6,0(0,13)

is executed, the two numbers are added as follows:

		GUARD DIGIT
FPR6:	43 08 21 00	
Shifted number from storage:	43 00 12 34	5
Sum:	43 08 33 34	5

The guard digit participates in the addition but is discarded. The unnormalized sum replaces the high-order portion of FPR6.

If the result in FPR6 is converted to a normalized number (42 83 33 40 00 00 00 00) and is compared to the result in FPR6 when ADD NORMALIZED was used (42 83 33 45 00 00 00 00), in this case it is apparent that the use of ADD NORMALIZED (with the retention of the guard digit) has preserved some additional significance in the result.

Compare (CE, CER, CD, CDR)

Assume that FPR4 contains 43 00 00 00 00 00 00 00 (= 0), and FPR6 contains 34 12 34 56 78 9A BC DE (a positive number). The contents of the two registers are to be compared with the following long-precision instruction:

Machine Format

OP CODE	R1	R2
29	4	6

Assembler Format

OP CODE	R1R2
CDR	4,6

When this instruction is executed, the number with the smaller characteristic is taken from the register and right-shifted until the two characteristics agree. The shifted contents of the FPR6 are:

		GUARD DIGIT
FPR6:	43 00 00 00 00 00 00 00	0

Therefore, when the two numbers are compared the condition code is set to 0, indicating an equality.

As the above example implies, when floating-point numbers are compared, more than two numbers may compare equally if one of the numbers is unnormalized. For example, the unnormalized floating-point number 41 00 12 34 56 78 9A BC compares equally with all numbers of the form 3F 12 34 56 78 9A BC 0X (X represents any hexadecimal number). When the COMPARE instruction is executed, the two low-order digits are shifted right two places; the 0 becomes the guard digit, and the X does not participate in the comparison.

Note, however, that when two normalized floating-point numbers are compared, the relationship between numbers that compare equally is unique: each digit in one number must be identical to the corresponding digit in the other number.

Status Switching

Supervisor Call (SVC)

The SUPERVISOR CALL instruction allows a program that is operating in the problem state to switch the CPU to the supervisor state. At the same time, the problem program can make a byte of information available to the supervisor program. For example, the instruction:

Machine Format

OP CODE	I
0A	01

Assembler Format

OP CODE	I
SVC	1

causes a supervisor-call interruption. The byte of information ($0000\ 0001_2$) is placed in the interruption-code field of the SUPERVISOR CALL old PSW (storage location 23_{16}), and a new PSW is fetched from location 60_{16}. The information byte may indicate, for example, that certain conditions encountered during processing require further attention (e.g., the job has been completed and a printout of storage is desired).

Set Storage Key (SSK)

Assume that the storage block corresponding to addresses 800-FFF has bits 11110 set into its storage key (that is, only programs with a protection key of 0 or 15_{10} can store data in this block, but any program can fetch data). Also assume that:

Register 5 contains 00 00 0A 60
Register 6 contains 00 00 00 F0

When the instruction:

Machine Format

OP CODE	R1	R2
08	6	5

Assembler Format

OP CODE	R1R2
SSK	6,5

is executed, bits 8-20 of register 5 are examined; their value indicates which block of $2{,}048_{10}$ bytes is to have its key set:

Register 5 (bits 8-20): 0000 0000 0000 1

In this case register 5 indicates that the "first" block (addresses 800-FFF) is the block being addressed. Note that register 5 will contain all zeros if the block containing addresses 000-7FF is being addressed. Also note that it is not necessary for R_2 to contain the exact address of the first byte in the block (i.e., 00 00 08 00) because only bits 8-20 of R_2 are examined.

The key setting for the storage block indicated by register 5 is obtained from bits 24-28 of register 6:

Register 6 (bits 24-28) 1111 0

If the fetch protection feature is installed, and it is desired to prevent fetching as well as storing of data in locations 800-FFF, the low-order bit of the storage key must be set to 1. This bit can be set to 1 if bit 28 of register 6 is set to 1 before the execution of SSK. (The register could contain 00 00 00 F8, for example.)

Insert Storage Key (ISK)

Assume that the key of the storage block containing addresses 800-FFF is to be inspected and that:

Register 5 contains 00 00 08 00
Register 6 contains FF FF FF FF

Execution of the instruction:

Machine Format

OP CODE	R1	R2
09	6	5

Assembler Format

OP CODE	R1R2
ISK	6,5

changes the contents of register 6 to:
1111 1111 1111 1111 1111 1111 MMMM M000 where MMMMM represents the five-bit storage key. Note that the last M is set to 0 if fetch protection is not installed.

Test and Set (TS)

The TEST AND SET instruction can be used to control the sharing of a storage area that is used in common by more than one program. Assume that the convention has been established that when the leftmost bit of an indicator byte is set to 1, it is a signal to all other programs not to attempt to access the storage area. When a program has finished using the storage area, it can then set the leftmost bit of the indicator byte to 0, indicating that other programs may now access the area.

For example, assume that register 10 contains the address of the indicator byte (00 00 34 56) and that the indicator byte itself initially contains the bits 0000 0000. After the instruction:

Machine Format

OP CODE	B1	D1
93	A	000

Assembler Format

OP CODE	D1	B1
TS	0(10)	

is executed:

The indicator byte (location 3456) contains bits 1111 1111.
The condition code is set to zero (indicating that the test revealed the leftmost bit of the indicator byte was zero).

Appendix B. Fixed-Point and Two's Complement Notation

A fixed-point number is a signed value, recorded as a binary integer. It is called fixed point because the programmer determines the fixed positioning of the binary point.

Fixed-point operands may be recorded in halfword (16 bits) or word (32 bits) lengths. In both lengths, the first bit position (0) holds the sign of the number, with the remaining bit positions (1-15 for halfwords and 1-31 for fullwords) used to designate the magnitude of the number.

Positive fixed-point numbers are represented in true binary form with a zero sign bit. Negative fixed-point numbers are represented in two's complement notation with a one bit in the sign position. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit (i.e. all zeros for positive numbers, all ones for negative numbers).

Negative fixed-point numbers are formed in two's complement notation by inverting each bit of the positive binary number and adding one. For example, the true binary form of the decimal value (plus 26) is made negative (minus 26) in the following manner:

	S INTEGER	
+26	0 0000000 00011010	
Invert	1 1111111 11100101	
Add 1	1	
−26	1 1111111 11100110	(Two's complement form)

This is equivalent to subtracting the number: 00000000 00011010 from 1 00000000 00000000.

The following addition examples illustrate two's complement arithmetic. Only eight bit positions are used. All negative numbers are in two's complement form.

			COMMENTS
+57	=	00111001	
+35	=	00100011	
+92	=	01011100	
+57	=	00111001	
−35	=	11011101	No overflow
+22	=	00010110	Ignore carry – carry into high order position and carry out.
+35	=	00100011	
−57	=	11000111	
−22	=	11101010	Sign change only; no carry.
−57	=	11000111	
−35	=	11011101	No overflow
−92	=	10100100	Ignore carry – carry into high order position and carry out.
−57	=	11000111	
−92	=	10100100	
−149	=	*01101011	*Overflow – no carry into high order position but carry out.
+57	=	00111001	
+92	=	01011100	
149	=	*10010101	*Overflow – carry into high order position, no carry out.

The following are 16-bit fixed-point numbers. The first is the largest positive number and the last, the largest negative number.

NUMBER		DECIMAL	S INTEGER
$2^{15}-1$	=	32,767	=0 1111111 11111111
2^{0}	=	1	=0 0000000 00000001
0	=	0	=0 0000000 00000000
-2^{0}	=	−1	=1 1111111 11111111
-2^{15}	=	−32,768	=1 0000000 00000000

The following are 32-bit fixed-point numbers. The first is the largest positive number that can be represented by 32 bits, and the last is the largest negative number.

NUMBER		DECIMAL	S INTEGER
$2^{31}-1$	=	2 147 483 647	=0 1111111 11111111 11111111 11111111
2^{16}	=	65 536	=0 0000000 00000001 00000000 00000000
2^{0}	=	1	=0 0000000 00000000 00000000 00000001
0	=	0	=0 0000000 00000000 00000000 00000000
-2^{0}	=	−1	=1 1111111 11111111 11111111 11111111
-2^{1}	=	−2	=1 1111111 11111111 11111111 11111110
-2^{16}	=	−65 536	=1 1111111 11111111 00000000 00000000
$-2^{31}+1$	=	−2 147 483 647	=1 0000000 00000000 00000000 00000001
-2^{31}	=	−2 147 483 648	=1 0000000 00000000 00000000 00000000

Appendix C. Floating-Point Arithmetic

Floating-point arithmetic simplifies programming by automatically maintaining binary point placement (scaling) during computations in which the range of values used vary widely or are unpredictable.

The key to floating-point data representation is the separation of the significant digits of a number from the size (scale) of the number. Thus, the number is expressed as a fraction times a power of 16.

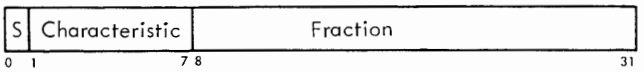
A floating-point number has two associated sets of values. One set represents the significant digits of the number and is called the fraction. The second set specifies the power (exponent) to which 16 is raised and indicates the location of the binary point of the number.

These two numbers (the fraction and exponent) are recorded in a single word or double word.

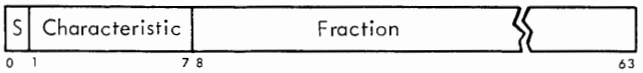
Since each of these two numbers is signed, some method must be employed to express two signs in an area that provides for a single sign. This is accomplished by having the fraction sign use the sign associated with the word (or double word) and expressing the exponent in excess 64 arithmetic; that is, the exponent is added as a signed number to 64. The resulting number is called the characteristic. Since 64 uses 7 bits, the characteristic can vary from 0 to 127, permitting the exponent to vary from –64 through 0 to +63. This provides a decimal range of n x 10^{75} to n x 10^{-78}.

Floating-point data in the System/360 may be recorded in short or long formats, depending on the precision required. Both formats use a sign bit in bit position 0, followed by a characteristic in bit positions 1-7. Short-precision floating-point data operands contain the fraction in bit positions 8-31; long-precision operands have the fraction in bit positions 8-63.

Short-Precision Floating-Point Format



Long-Precision Floating-Point Format



The sign of the fraction is indicated by a zero or one bit in bit position 0 to denote a positive or negative fraction, respectively.

Within a given fraction length (24 or 56 bits), a floating-point operation will provide the greatest precision if the fraction is normalized. A fraction is normalized when the high-order digit (bit positions 8, 9, 10 and 11) is not zero. It is unnormalized if the high-order digit contains all zeros.

If normalization of the operand is desired, the floating-point instructions that provide automatic normalization are used. This automatic normalization is accomplished by left-shifting the fraction (four bits per shift) until a nonzero digit occupies the high-order digit position. The characteristic is reduced by one for each digit shifted.

Conversion Example

Convert the decimal number 149.25 to a short-precision floating-point operand. (Appendix E provides tables for conversion of hexadecimal and decimal integers and fractions.)

1. The number is decomposed into a decimal integer and a decimal fraction.

$$149.25 = 149 \text{ plus } 0.25$$

2. The decimal integer is converted to its hexadecimal representation.

$$149_{10} = 95_{16}$$

3. The decimal fraction is converted to its hexadecimal representation.

$$0.25_{10} = 0.4_{16}$$

4. Combine the integral and fractional parts and express as a fraction times a power of 16 (exponent).

$$95.4_{16} = (0.954 \times 10^{2})_{16}$$

5. The characteristic is developed from the exponent and converted to binary.

base	+	exponent		characteristic
64	+	2	= 66 =	1000010

6. The fraction is converted to binary and grouped hexadecimally.

$$.954_{16} = .1001\ 0101\ 0100$$

7. The characteristic and the fraction are stored in short precision format. The sign position contains the sign of the fraction.

S	Char	Fraction
0	1000010	1001 0101 0100 0000 0000 0000

The following are sample normalized short floating-point numbers. The last two numbers represent the smallest and the largest positive normalized numbers.

NUMBER	POWERS OF 16	S CHAR	FRACTION
1.0	$= +1/16 \times 16^{1}$	=0 1000001	0001 0000 0000 0000 0000 0000
0.5	$= +8/16 \times 16^{0}$	=0 1000000	1000 0000 0000 0000 0000 0000
1/64	$= +4/16 \times 16^{-1}$	=0 0111111	0100 0000 0000 0000 0000 0000
0.0	$= +0 \times 16^{-64}$	=0 0000000	0000 0000 0000 0000 0000 0000
−15.0	$= -15/16 \times 16^{1}$	=1 1000001	1111 0000 0000 0000 0000 0000
5.4×10^{-79}	$\cong +1/16 \times 16^{-64}$	≅0 0000000	0001 0000 0000 0000 0000 0000
7×10^{75}	$\cong (1\text{-}16^{-6}) \times 16^{63}$	≅0 1111111	1111 1111 1111 1111 1111 1111

Appendix D. Powers of Two Table

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25

Appendix E. Hexadecimal Tables

The following tables aid in converting hexadecimal values to decimal values, or the reverse.

Direct Conversion Table

This table provides direct conversion of decimal and hexadecimal numbers in these ranges:

HEXADECIMAL	DECIMAL
000 to FFF	0000 to 4095

To convert numbers outside these ranges, use the hexadecimal and decimal conversion tables that follow the Direct Conversion Table in this Appendix.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00_	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01_	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02_	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03_	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04_	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05_	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06_	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07_	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08_	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09_	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A_	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B_	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C_	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D_	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E_	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F_	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10_	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11_	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12_	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13_	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14_	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15_	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16_	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17_	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18_	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19_	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A_	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B_	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C_	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D_	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E_	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F_	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20_	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21_	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22_	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23_	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24_	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25_	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26_	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27_	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28_	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29_	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A_	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B_	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C_	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D_	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E_	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F_	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30_	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31_	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32_	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33_	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34_	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35_	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36_	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37_	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38_	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39_	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A_	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B_	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C_	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D_	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E_	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F_	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40_	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41_	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42_	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43_	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44_	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45_	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46_	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47_	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48_	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49_	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A_	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B_	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C_	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D_	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E_	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F_	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50_	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51_	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52_	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53_	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54_	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55_	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56_	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57_	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58_	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59_	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A_	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B_	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C_	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D_	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E_	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F_	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60_	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61_	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62_	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63_	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64_	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65_	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66_	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67_	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68_	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69_	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A_	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B_	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C_	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D_	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E_	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F_	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
70_	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71_	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72_	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73_	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74_	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75_	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76_	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77_	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78_	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79_	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A_	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B_	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C_	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D_	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E_	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F_	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80_	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81_	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82_	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83_	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84_	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85_	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86_	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87_	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88_	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89_	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A_	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B_	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C_	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D_	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E_	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F_	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90_	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91_	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92_	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93_	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94_	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95_	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96_	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97_	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98_	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99_	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A_	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B_	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C_	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D_	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E_	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F_	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0_	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1_	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2_	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3_	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4_	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5_	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6_	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7_	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8_	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9_	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA_	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB_	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC_	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD_	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE_	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF_	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0_	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1_	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2_	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3_	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4_	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5_	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6_	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7_	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8_	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9_	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA_	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB_	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC_	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD_	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE_	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF_	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0_	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1_	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2_	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3_	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4_	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5_	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6_	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7_	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8_	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9_	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA_	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB_	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC_	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD_	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE_	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF_	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0_	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1_	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2_	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3_	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4_	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5_	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6_	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7_	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8_	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9_	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA_	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB_	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC_	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD_	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE_	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF_	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0_	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1_	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2_	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3_	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4_	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5_	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6_	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7_	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8_	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9_	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA_	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB_	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC_	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED_	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE_	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF_	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0_	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1_	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2_	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3_	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4_	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5_	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6_	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7_	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8_	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9_	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA_	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB_	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC_	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD_	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE_	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF_	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Hexadecimal and Decimal Integer Conversion Table

HALFWORD								HALFWORD							
BYTE				BYTE				BYTE				BYTE			
BITS: 0123		4567		0123		4567		0123		4567		0123		4567	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
8		7		6		5		4		3		2		1	

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position.
4. Add the numbers selected from the table to form the decimal number.

EXAMPLE

Conversion of Hexadecimal Value	D34
1. D	3328
2. 3	48
3. 4	4
4. Decimal	3380

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.
   (b) Record the hexadecimal of the column containing the selected number.
   (c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1(c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder).
3. Using the remainder from step 2 repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine terms to form the hexadecimal number.

EXAMPLE

Conversion of Decimal Value	3380
1. D	-3328
	52
2. 3	-48
	4
3. 4	-4
4. Hexadecimal	D34

To convert integer numbers greater than the capacity of table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D34_{16} = 3380_{10}$

D = 13
x16
208
3 = + 3
211
x16
3376
4 = +4
3380

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

16 | 3380		remainder
16 | 211	→	4
16 | 13	→	3
	→	D

$3380_{10} = D34_{16}$

POWERS OF 16 TABLE

Example: $268,435,456_{10} = (2.68435456 \times 10^8)_{10} = 1000\ 0000_{16} = (10^7)_{16}$

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10 = A
17 592 186 044 416	11 = B
281 474 976 710 656	12 = C
4 503 599 627 370 496	13 = D
72 057 594 037 927 936	14 = E
1 152 921 504 606 846 976	15 = F

Decimal Values

Hexadecimal and Decimal Fraction Conversion Table

HALFWORD													
BYTE					BYTE								
BITS 0123		4567			0123				4567				
Hex	Decimal	Hex	Decimal		Hex	Decimal			Hex	Decimal Equivalent			
.0	.0000	.00	.0000	0000	.000	.0000	0000	0000	.0000	.0000	0000	0000	0000
.1	.0625	.01	.0039	0625	.001	.0002	4414	0625	.0001	.0000	1525	8789	0625
.2	.1250	.02	.0078	1250	.002	.0004	8828	1250	.0002	.0000	3051	7578	1250
.3	.1875	.03	.0117	1875	.003	.0007	3242	1875	.0003	.0000	4577	6367	1875
.4	.2500	.04	.0156	2500	.004	.0009	7656	2500	.0004	.0000	6103	5156	2500
.5	.3125	.05	.0195	3125	.005	.0012	2070	3125	.0005	.0000	7629	3945	3125
.6	.3750	.06	.0234	3750	.006	.0014	6484	3750	.0006	.0000	9155	2734	3750
.7	.4375	.07	.0273	4375	.007	.0017	0898	4375	.0007	.0001	0681	1523	4375
.8	.5000	.08	.0312	5000	.008	.0019	5312	5000	.0008	.0001	2207	0312	5000
.9	.5625	.09	.0351	5625	.009	.0021	9726	5625	.0009	.0001	3732	9101	5625
.A	.6250	.0A	.0390	6250	.00A	.0024	4140	6250	.000A	.0001	5258	7890	6250
.B	.6875	.0B	.0429	6875	.00B	.0026	8554	6875	.000B	.0001	6784	6679	6875
.C	.7500	.0C	.0468	7500	.00C	.0029	2968	7500	.000C	.0001	8310	5468	7500
.D	.8125	.0D	.0507	8125	.00D	.0031	7382	8125	.000D	.0001	9836	4257	8125
.E	.8750	.0E	.0546	8750	.00E	.0034	1796	8750	.000E	.0002	1362	3046	8750
.F	.9375	.0F	.0585	9375	.00F	.0036	6210	9375	.000F	.0002	2888	1835	9375
1		2			3				4				

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

Find .A in position 1 .6250

Find .0B in position 2 .0429 6875

Find .00C in position 3 .0029 2968 7500

.ABC Hex is equal to .6708 9843 7500

TO CONVERT .13 DECIMAL TO HEXADECIMAL

1. Find .1250 next lowest to .1300
   subtract -.1250 = .2 Hex
2. Find .0039 0625 next lowest to .0050 0000
   -.0039 0625 = .01
3. Find .0009 7656 2500 .0010 9375 0000
   -.0009 7656 2500 = .004
4. Find .0001 0681 1523 4375 .0001 1718 7500 0000
   -.0001 0681 1523 4375 = .0007
   .0000 1037 5976 5625 = .2147 Hex
5. .13 Decimal is approximately equal to ——————↑

To convert fractions beyond the capacity of table, use techniques below:

HEXADECIMAL FRACTION TO DECIMAL

Convert the hexadecimal fraction to its decimal equivalent using the same technique as for integer numbers. Divide the results by 16^n (n is the number of fraction positions).

Example: $.8A7 = .540771_{10}$

$8A7_{16} = 2215_{10}$

$16^3 = 4096$

$$4096\overline{)2215.000000}\quad .540771$$

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

.5408
x16
8 ← [8].6528
x16
A ← [10].4448
x16
7 ← [7].1168

Hexadecimal Addition and Subtraction Table

Example: 6 + 2 = 8, 8 - 2 = 6, and 8 - 6 = 2

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Hexadecimal Multiplication Table

Example: 2 x 4 = 08, F x 2 = 1E

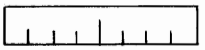
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Appendix F. USASCII-8 and EBCDIC Charts

Charts and supporting text appear on the following pages.

Code Structures

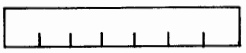
Both USASCII-8 and EBCDIC provide for 256 possible characters. Each character is composed of eight bits (one byte), and each bit position is assigned a number. The bit positions are numbered as follows:



USASCII-8	8 7 6 5 4 3 2 1
EBCDIC	0 1 2 3 4 5 6 7

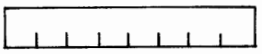
USA Standard Code for Information Interchange (USASCII) Embedded in USASCII-8

The seven-bit USASCII has its bits numbered as follows:



USASCII b_7 b_6 b_5 b_4 b_3 b_2 b_1

The seven-bit USASCII can be extended to eight bits and embedded in USASCII-8 as follows:



USASCII	b_7 b_6 b_7 b_5 b_4 b_3 b_2 b_1
USASCII-8	8 7 6 5 4 3 2 1

The 256-position table at the right, outlined by the heavy black lines, shows the graphic characters and control character representations for USASCII-8. The bit-position numbers, bit patterns, and hexadecimal representations for these and other possible USASCII-8 characters are also shown.

Bit positions 4, 3, 2, 1	Second Hexadecimal Digit	00				01				10				11				Bit Positions 8, 7
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	Bit Positions 6, 5
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	First Hexadecimal Digit
0000	0	NUL	DLE			SP	0					@	P			`	P	
0001	1	SOH	DC1			! (1)	1					A	Q			a	q	
0010	2	STX	DC2			"	2					B	R			b	r	
0011	3	ETX	DC3			#	3					C	S			c	s	
0100	4	EOT	DC4			$	4					D	T			d	t	
0101	5	ENQ	NAK			%	5					E	U			e	u	
0110	6	ACK	SYN			&	6					F	V			f	v	
0111	7	BEL	ETB			'	7					G	W			g	w	
1000	8	BS	CAN			(	8					H	X			h	x	
1001	9	HT	EM			)	9					I	Y			i	y	
1010	A	LF	SUB			*	:					J	Z			j	z	
1011	B	VT	ESC			+	;					K	[			k	{	
1100	C	FF	FS			,	<					L	\			l	¦	
1101	D	CR	GS			-	=					M	]			m	}	
1110	E	SO	RS			.	>					N	^ (2)			n	~	
1111	F	SI	US			/	?					O	_			o	DEL	

(1) In the event that IBM equipment implementing USASCII-8 is provided, the graphic | (Logical OR) will be used instead of ! (Exclamation Point).

(2) In the event that IBM equipment implementing USASCII-8 is provided, the graphic ¬ (Logical NOT) will be used instead of ^ (Circumflex).

Note: Current activities in committees under the auspices of the United States of America Standards Institute may result in changes to the characters and/or structure of the eight-bit representation of USASCII devised by the Institute. Such changes may cause the eight-bit representation of USASCII implemented in System/360 (USASCII-8) to be different from a future USA Standard. Since a difference of this nature may eventually lead to a modification of System/360, it is recommended that users avoid: (1) operation with PSW bit 12 set to 1, and (2) the use of any sign codes in decimal data other than those preferred for EBCDIC.

Control Character Representations

NUL	Null
SOH	Start of Heading (CC)
STX	Start of Text (CC)
ETX	End of Text (CC)
EOT	End of Transmission (CC)
ENQ	Enquiry (CC)
ACK	Acknowledge (CC)
BEL	Bell
BS	Backspace (FE)
HT	Horizontal Tabulation (FE)
LF	Line Feed (FE)
VT	Vertical Tabulation (FE)
FF	Form Feed (FE)
CR	Carriage Return (FE)
SO	Shift Out
SI	Shift In
DLE	Data Link Escape (CC)
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
NAK	Negative Acknowledge (CC)
SYN	Synchronous Idle (CC)
ETB	End of Transmission Block (CC)
CAN	Cancel
EM	End of Medium
SUB	Substitute
ESC	Escape
FS	File Separator (IS)
GS	Group Separator (IS)
RS	Record Separator (IS)
US	Unit Separator (IS)
DEL	Delete

(CC) Communication Control
(FE) Format Effector
(IS) Information Separator

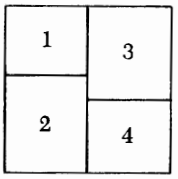
Special Graphic Characters

SP	Space	<	Less Than
!	Exclamation Point	=	Equals
¦	Logical OR	>	Greater Than
"	Quotation Marks	?	Question Mark
#	Number Sign	@	Commercial At
$	Dollar Sign	[	Opening Bracket
%	Percent	\	Reverse Slant
&	Ampersand	]	Closing Bracket
'	Apostrophe	^	Circumflex
(	Opening Parenthesis	¬	Logical NOT
)	Closing Parenthesis	_	Underline
*	Asterisk	`	Grave Accent
+	Plus	{	Opening Brace
,	Comma	¦	Vertical Line (This graphic is stylized to distinguish it from Logical OR)
-	Hyphen (Minus)		
.	Period (Decimal Point)		
/	Slant	}	Closing Brace
:	Colon	~	Tilde
;	Semicolon		

Extended Binary-Coded-Decimal Interchange Code (EBCDIC)

The 256-position table at the right, outlined by the heavy black lines, shows the graphic characters and control character representations for EBCDIC. The bit-position numbers, bit patterns, hexadecimal representations and card hole patterns for these and other possible EBCDIC characters are also shown.

To find the card hole patterns for most characters, partition the 256-position table into four blocks as follows:



Block 1: Zone punches at top of table; digit punches at left
Block 2: Zone punches at bottom of table; digit punches at left
Block 3: Zone punches at top of table; digit punches at right
Block 4: Zone punches at bottom of table; digit punches at right

Fifteen positions in the table are exceptions to the above arrangement. These positions are indicated by small numbers in the upper right corners of their boxes in the table. The card hole patterns for these positions are given at the bottom of the table. Bit-position numbers, bit patterns, and hexadecimal representations for these positions are found in the usual manner.

Following are some examples of the use of the EBCDIC chart:

Character	Type	Bit Pattern	Hex	Hole Pattern	
				Zone Punches	Digit Punches
PF	Control Character	00 00 0100	04	12 - 9	- 4
%	Special Graphic	01 10 1100	6C	0	- 8 - 4
R	Upper Case	11 01 1001	D9	11	- 9
a	Lower Case	10 00 0001	81	12 - 0	- 1
	Control Character, function not yet assigned	00 11 0000	30	12 - 11 - 0 - 9	- 8 - 1

Bit Positions
01 23 4567

Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	Digit Punches	00				01				10				11				
			00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	Bit Positions 0, 1
			00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	Bit Positions 2, 3
			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	First Hexadecimal Digit
			12				12	12		12	12	12		12	12				Zone Punches
				11				11	11	11		11	11	11		11			
					0		0		0	0	0		0	0			0		
			9	9	9	9	9	9	9	9									Digit Punches
0000	0	8-1	(1) NUL	(2) DLE	(3) DS	(4)	(5) SP	(6) &	(7) -	(8)					(9)	(10)	(11)	(12) 0	8-1
0001	1	1	SOH	DC1	SOS				(13) /		a	j			A	J	(14)	1	1
0010	2	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2	2
0011	3	3	ETX	TM							c	l	t		C	L	T	3	3
0100	4	4	PF	RES	BYP	PN					d	m	u		D	M	U	4	4
0101	5	5	HT	NL	LF	RS					e	n	v		E	N	V	5	5
0110	6	6	LC	BS	ETB	UC					f	o	w		F	O	W	6	6
0111	7	7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7	7
1000	8	8		CAN							h	q	y		H	Q	Y	8	8
1001	9	8-1		EM							i	r	z		I	R	Z	9	9
1010	A	8-2	SMM	CC	SM		¢	!	(15)	:									8-2
1011	B	8-3	VT	CU1	CU2	CU3		$	,	#									8-3
1100	C	8-4	FF	IFS		DC4	<	*	%	@									8-4
1101	D	8-5	CR	IGS	ENQ	NAK	(	)	_	'									8-5
1110	E	8-6	SO	IRS	ACK		+	;	>	=									8-6
1111	F	8-7	SI	IUS	BEL	SUB	|	¬	?	"									8-7
			12				12				12	12		12	12	12		12	Zone Punches
				11				11				11	11	11		11	11	11	
					0				0		0		0	0	0		0	0	
			9	9	9	9									9	9	9	9	

Card Hole Patterns

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

Control Character Representations

ACK	Acknowledge	EOT	End of Transmission	PF	Punch Off
BEL	Bell	ESC	Escape	PN	Punch On
BS	Backspace	ETB	End of Transmission Block	RES	Restore
BYP	Bypass	ETX	End of Text	RS	Reader Stop
CAN	Cancel	FF	Form Feed	SI	Shift In
CC	Cursor Control	FS	Field Separator	SM	Set Mode
CR	Carriage Return	HT	Horizontal Tab	SMM	Start of Manual Message
CU1	Customer Use 1	IFS	Interchange File Separator	SO	Shift Out
CU2	Customer Use 2	IGS	Interchange Group Separator	SOH	Start of Heading
CU3	Customer Use 3	IL	Idle	SOS	Start of Significance
DC1	Device Control 1	IRS	Interchange Record Separator	SP	Space
DC2	Device Control 2	IUS	Interchange Unit Separator	STX	Start of Text
DC4	Device Control 4	LC	Lower Case	SUB	Substitute
DEL	Delete	LF	Line Feed	SYN	Synchronous Idle
DLE	Data Link Escape	NAK	Negative Acknowledge	TM	**Tape Mark**
DS	Digit Select	NL	New Line	UC	**Upper Case**
EM	End of Medium	NUL	Null	VT	**Vertical Tab**
ENQ	Enquiry				

Special Graphic Characters

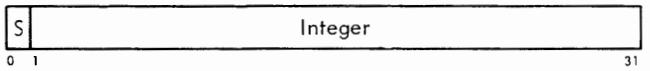
¢	Cent Sign	-	Minus Sign, Hyphen
.	Period, Decimal Point	/	Slash
<	Less-than Sign	,	Comma
(	Left Parenthesis	%	Percent
+	Plus Sign	_	Underscore
|	Logical OR	>	Greater-than Sign
&	Ampersand	?	Question Mark
!	Exclamation Point	:	Colon
$	Dollar Sign	#	Number Sign
*	Asterisk	@	At Sign
)	Right Parenthesis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

Appendix G. Formats and Tables

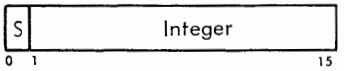
Data Formats

Fixed-Point Numbers

Full Word Fixed-Point Number

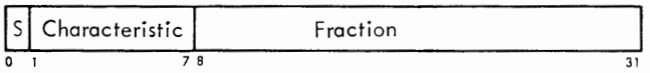


Halfword Fixed-Point Number

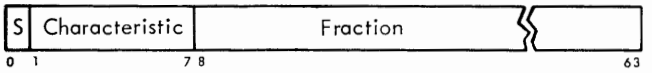


Floating-Point Numbers

Short Floating-Point Number

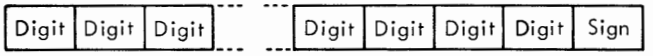


Long Floating-Point Number

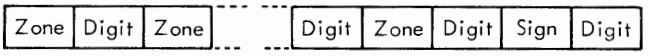


Decimal Numbers

Packed Decimal Number

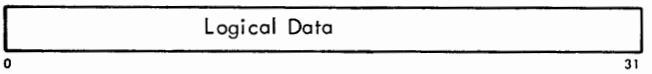


Zoned Decimal Number

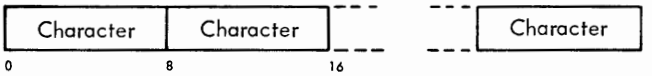


Logical Information

Fixed-Length Logical Information



Variable-Length Logical Information



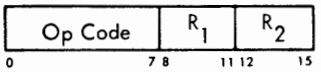
Hexadecimal Representation

HEXADECIMAL CODE	PRINTED GRAPHIC	EBCDIC* CODE	ASCII-8† CODE
0000	0	1111 0000	0101 0000
0001	1	1111 0001	0101 0001
0010	2	1111 0010	0101 0010
0011	3	1111 0011	0101 0011
0100	4	1111 0100	0101 0100
0101	5	1111 0101	0101 0101
0110	6	1111 0110	0101 0110
0111	7	1111 0111	0101 0111
1000	8	1111 1000	0101 1000
1001	9	1111 1001	0101 1001
1010	A	1100 0001	1010 0001
1011	B	1100 0010	1010 0010
1100	C	1100 0011	1010 0011
1101	D	1100 0100	1010 0100
1110	E	1100 0101	1010 0101
1111	F	1100 0110	1010 0110

*Extended Binary-Coded-Decimal Interchange Code.
†An eight-bit representation for American Standard Code for Information Interchange for use in eight-bit environments.

Instructions by Format Type

RR Format

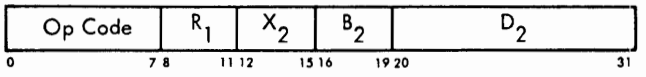


Fixed Point		*Floating Point*	
Load		Load S/L	
Load and Test		Load and Test S/L	
Load Complement		Load Complement S/L	
Load Positive		Load Positive S/L	
Load Negative		Load Negative S/L	
Add		Add Normalized S/L	
Add Logical		Add Unnormalized S/L	
Subtract		Subtract Normalized S/L	
Subtract Logical		Subtract Unnormalized S/L	
Compare		Compare S/L	
Multiply	E	Halve S/L	
Divide	E	Multiply S/L	
		Divide S/L	

Logical		*Status Switching*	
Compare		Set Program Mask	2
AND		Supervisor Call	3
OR		Set Storage Key	Z
Exclusive OR		Insert Storage Key	Z

Branching	
Branch on Condition	1
Branch and Link	
Branch on Count	

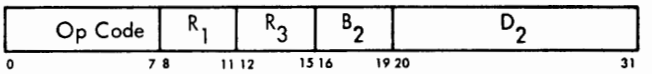
RX Format



Fixed Point		*Floating Point*	
Load H/F		Load S/L	
Add H/F		Add Normalized S/L	
Add Logical		Add Unnormalized S/L	
Subtract H/F		Subtract Normalized S/L	
Subtract Logical		Subtract Unnormalized S/L	
Compare H/F		Compare S/L	
Multiply H		Multiply S/L	
Multiply F	E	Store S/L	
Divide F	E	Divide S/L	
Convert to Binary			
Convert to Decimal			
Store H/F			

Logical		*Branching*	
Compare		Branch on Condition	1
Load Address		Branch and Link	
Insert Character		Branch on Count	
Store Character		Execute	
AND			
OR			
Exclusive OR			

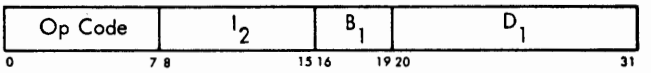
RS Format



Fixed Point		*Logical*	
Load Multiple		Shift Left Single	2
Store Multiple		Shift Right Single	2
Shift Left Single	2	Shift Left Double	E,2
Shift Right Single	2	Shift Right Double	E,2
Shift Left Double	E,2		
Shift Right Double	E,2		

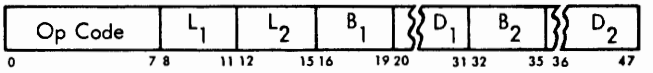
Branching
Branch on High
Branch on Low-Eq

SI Format



Input/Output		*Status Switching*	
Start I/O	4	Load PSW	4
Test I/O	4	Set System Mask	4
Halt I/O	4	Write Direct	Y
Test Channel	4	Read Direct	Y
		Diagnose	
Logical		Test and Set	4
Move			
Compare			
AND			
OR			
Exclusive OR			
Test Under Mask			

SS Format



Decimal		*Logical*	
Pack		Move	5
Unpack		Move Numeric	5
Move With Offset		Move Zone	5
Zero and Add	T	Compare	5
Add	T	AND	5
Subtract	T	OR	5
Compare	T	Exclusive OR	5
Multiply	T	Translate	5
Divide	T	Translate and Test	5
		Edit	T,5
		Edit and Mark	T,5

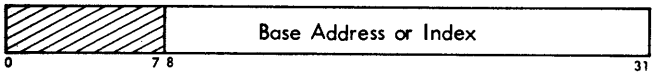
FORMAT NOTES

E	R_1 must be even
F	Fullword
H	Halfword
L	Long
S	Short
T	Decimal feature
Y	Direct control feature
Z	Protection feature
1	R_1 used as mask M_1
2	R_2 or R_3 ignored
3	R_1 and R_2 used as immediate information
4	I_2 ignored
5	L_1 and L_2 used as eight-bit L field

All floating-point instructions are part of the floating-point feature.

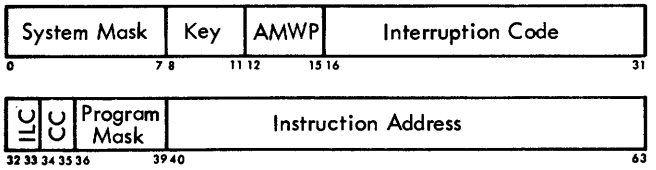
Control Word Formats

Base and Index Registers



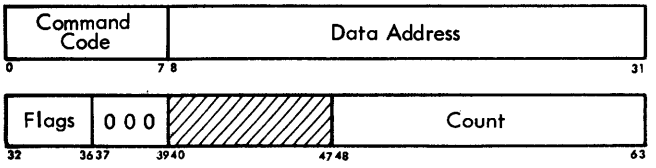
0-7 Ignored
8-31 Base address or index

Program Status Word



0-7 System mask
0 Channel 0 mask
1 Channel 1 mask
2 Channel 2 mask
3 Channel 3 mask
4 Channel 4 mask
5 Channel 5 mask
6 Channel 6 mask
7 External mask
8-11 Protection key
12 ASCII-8 mode (A)
13 Machine check mask (M)
14 Wait state (W)
15 Problem state (P)
16-31 Interruption code
32-33 Instruction length code (ILC)
34-35 Condition code (CC)
36-39 Program mask
36 Fixed-point overflow mask
37 Decimal overflow mask
38 Exponent underflow mask
39 Significance mask
40-63 Instruction address

Channel Command Word



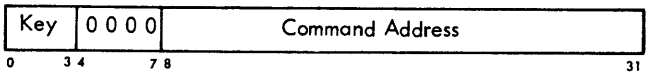
0-7 Command code
8-31 Data address
32-36 Command flags
32 Chain data flag
33 Chain command flag
34 Suppress length indication flag
35 Skip flag
36 Program-controlled interruption flag
37-39 Zero
40-47 Ignored
48-63 Count

Command Code Assignment

NAMES	FLAGS	CODE
Write	CD CC SLI PCI	MMMM MM01
Read	CD CC SLI SKIP PCI	MMMM MM10
Read Backward	CD CC SLI SKIP PCI	MMMM 1100
Control	CD CC SLI PCI	MMMM MM11
Sense	CD CC SLI SKIP PCI	MMMM 0100
Transfer in Channel		xxxx 1000

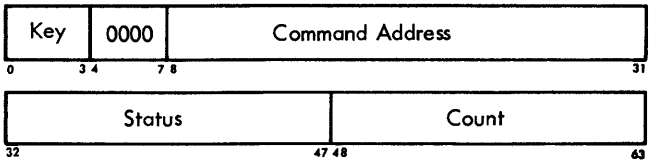
CD = Chain data
CC = Chain command
SLI = Suppress length indication
SKIP = Skip
PCI = Program-controlled interruption

Channel Address Word



0-3 Protection key
4-7 Zero
8-31 Command address

Channel Status Word



0-3 Protection key
4-7 Zero
8-31 Command address
32-47 Status
32 Attention
33 Status modifier
34 Control unit end
35 Busy
36 Channel end
37 Device end
38 Unit check
39 Unit exception
40 Program-controlled interruption
41 Incorrect length
42 Program check
43 Protection check
44 Channel data check
45 Channel control check
46 Interface control check
47 Chaining check
48-63 Count

Operation Codes

RR FORMAT

xxxx	0000xxxx		0001xxxx		0010xxxx		0011xxxx	
	Branching and Status Switching		Fixed-Point Full Word and Logical		Floating-Point Long		Floating-Point Short	
0000			LPR	Load Positive	LPDR	Load Positive	LPER	Load Positive
0001			LNR	Load Negative	LNDR	Load Negative	LNER	Load Negative
0010			LTR	Load and Test	LTDR	Load and Test	LTER	Load and Test
0011			LCR	Load Complement	LCDR	Load Complement	LCER	Load Complement
0100	SPM	Set Program Mask	NR	AND	HDR	Halve	HER	Halve
0101	BALR	Branch and Link	CLR	Compare Logical				
0110	BCTR	Branch on Count	OR	OR				
0111	BCR	Branch/Condition	XR	Exclusive OR				
1000	SSK	Set Key	LR	Load	LDR	Load	LER	Load
1001	ISK	Insert Key	CR	Compare	CDR	Compare	CER	Compare
1010	SVC	Supervisor Call	AR	Add	ADR	Add N	AER	Add N
1011			SR	Subtract	SDR	Subtract N	SER	Subtract N
1100			MR	Multiply	MDR	Multiply	MER	Multiply
1101			DR	Divide	DDR	Divide	DER	Divide
1110			ALR	Add Logical	AWR	Add U	AUR	Add U
1111			SLR	Subtract Logical	SWR	Subtract U	SUR	Subtract U

RX FORMAT

xxxx	0100xxxx		0101xxxx		0110xxxx		0111xxxx	
	Fixed-Point Halfword and Branching		Fixed-Point Full Word and Logical		Floating-Point Long		Floating-Point Short	
0000	STH	Store	ST	Store	STD	Store	STE	Store
0001	LA	Load Address						
0010	STC	Store Character						
0011	IC	Insert Character						
0100	EX	Execute	N	AND				
0101	BAL	Branch and Link	CL	Compare Logical				
0110	BCT	Branch an Count	O	OR				
0111	BC	Branch/Condition	X	Exclusive OR				
1000	LH	Load	L	Load	LD	Load	LE	Load
1001	CH	Compare	C	Compare	CD	Compare	CE	Compare
1010	AH	Add	A	Add	AD	Add N	AE	Add N
1011	SH	Subtract	S	Subtract	SD	Subtract N	SE	Subtract N
1100	MH	Multiply	M	Multiply	MD	Multiply	ME	Multiply
1101			D	Divide	DD	Divide	DE	Divide
1110	CVD	Convert-Decimal	AL	Add Logical	AW	Add U	AU	Add U
1111	CVB	Convert-Binary	SL	Subtract Logical	SW	Subtract U	SU	Subtract U

RS, SI FORMAT

xxxx	1000xxxx		1001xxxx		1010xxxx	1011xxxx
	Branching Status Switching and Shifting		Fixed-Point Logical and Input/Output			
0000	SSM	Set System Mask	STM	Store Multiple		
0001			TM	Test Under Mask		
0010	LPSW	Load PSW	MVI	Move		
0011		Diagnose	TS	Test and Set		
0100	WRD	Write Direct	NI	AND		
0101	RDD	Read Direct	CLI	Compare Logical		
0110	BXH	Branch/High	OI	OR		
0111	BXLE	Branch/Low-Equal	XI	Exclusive OR		
1000	SRL	Shift Right SL	LM	Load Multiple		
1001	SLL	Shift Left SL				
1010	SRA	Shift Right S				
1011	SLA	Shift Left S				
1100	SRDL	Shift Right DL	SIO	Start I/O		
1101	SLDL	Shift Left DL	TIO	Test I/O		
1110	SRDA	Shift Right D	HIO	Halt I/O		
1111	SLDA	Shift Left D	TCH	Test Channel		

SS FORMAT

xxxx	1100xxxx		1101xxxx		1110xxxx	1111xxxx	
			Logical			Decimal	
0000							
0001			MVN	Move Numeric		MVO	Move with Offset
0010			MVC	Move		PACK	Pack
0011			MVZ	Move Zone		UNPK	Unpack
0100			NC	AND			
0101			CLC	Compare Logical			
0110			OC	OR			
0111			XC	Exclusive OR			
1000						ZAP	Zero and Add
1001						CP	Compare
1010						AP	Add
1011						SP	Subtract
1100			TR	Translate		MP	Multiply
1101			TRT	Translate and Test		DP	Divide
1110			ED	Edit			
1111			EDMK	Edit and Mark			

Note: N = Normalized
SL = Single Logical
DL = Double Logical
U = Unnormalized
S = Single
D = Double

Permanent Storage Assignment

	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 on the particular model and I/O channels.

Condition Code Setting

Fixed-Point Arithmetic

	0	1	2	3
Add H/F	zero	< zero	> zero	overflow
Add Logical	zero, no carry	not zero, no carry	zero, carry	not zero, carry
Compare H/F	equal	low	high	--
Load and Test	zero	< zero	> zero	--
Load Complement	zero	< zero	> zero	overflow
Load Negative	zero	< zero	--	--
Load Positive	zero	--	> zero	overflow
Shift Left Double	zero	< zero	> zero	overflow
Shift Left Single	zero	< zero	> zero	overflow
Shift Right Double	zero	< zero	> zero	--
Shift Right Single	zero	< zero	> zero	--
Subtract H/F	zero	< zero	> zero	overflow
Subtract Logical	--	not zero, no carry	zero, carry	not zero, carry

Decimal Arithmetic

Add Decimal	zero	< zero	> zero	overflow
Compare Decimal	equal	low	high	--
Subtract Decimal	zero	< zero	> zero	overflow
Zero and Add	zero	< zero	> zero	overflow

Floating-Point Arithmetic

Add Normalized S/L	zero	< zero	> zero	--
Add Unnormalized S/L	zero	< zero	> zero	--
Compare S/L	equal	low	high	--
Load and Test S/L	zero	< zero	> zero	--
Load Complement S/L	zero	< zero	> zero	--
Load Negative S/L	zero	< zero	--	--
Load Positive S/L	zero	--	> zero	--
Subtract Normalized S/L	zero	< zero	> zero	--
Subtract Unnormalized S/L	zero	< zero	> zero	--

Logical Operations

AND	zero	not zero	--	--
Compare Logical	equal	low	high	--
Edit	zero	< zero	> zero	--
Edit and Mark	zero	< zero	> zero	--
Exclusive OR	zero	not zero	--	--
OR	zero	not zero	--	--
Test Under Mask	zero	mixed	--	one
Translate and Test	zero	incomplete	complete	--

Status Switching

Test and Set	zero	one	--	--

Input/Output Operations

Halt I/O	interruption pending	CSW stored	burst op stopped	not operational
Start I/O	successful	CSW stored	busy	not operational
Test I/O	available	CSW stored	busy	not operational
Test Channel	available	interruption pending	burst mode	not operational

CONDITION CODE SETTING NOTES

available	Unit and channel available
burst op stopped	Burst operation stopped
busy	Unit or channel busy
carry	A carry out of the sign position occurred
complete	Last result byte nonzero
CSW stored	Channel status word stored
equal	Operands compare equal
F	Fullword
> zero	Result is greater than zero
H	Halfword
halted	Data transmission stopped. Unit in halt-reset mode
high	First operand compares high
incomplete	Nonzero result byte; not last
L	Long precision
< zero	Result is less than zero
low	First operand compares low
mixed	Selected bits are both zero and one
not operational	Unit or channel not operational
not zero	Result is not all zero
one	Selected bits are one
overflow	Result overflows
S	Short precision
zero	Result or selected bits are zero

The condition code also may be changed by LOAD PSW, SET PROGRAM MASK, DIAGNOSE, and by an interruption.

Interruption Action

SOURCE IDENTIFICATION	INTERRUPTION CODE PSW BITS 16-31	MASK BITS	ILC SET	EXECUTION
Input/Output (old PSW 56, new PSW 120, Priority 4)				
Channel 0	00000000 aaaaaaaa	0	x	completed
Channel 1	00000001 aaaaaaaa	1	x	completed
Channel 2	00000010 aaaaaaaa	2	x	completed
Channel 3	00000011 aaaaaaaa	3	x	completed
Channel 4	00000100 aaaaaaaa	4	x	completed
Channel 5	00000101 aaaaaaaa	5	x	completed
Channel 6	00000110 aaaaaaaa	6	x	completed

Program (old PSW 40, new PSW 104, priority 2)

Operation	00000000 00000001		1,2,3	suppressed
Privileged operation	00000000 00000010		1,2	suppressed
Execute	00000000 00000011		2	suppressed
Protection	00000000 00000100		0,2,3	suppressed or terminated
Addressing	00000000 00000101		0,1,2,3	suppressed or terminated
Specification	00000000 00000110		1,2,3	suppressed
Data	00000000 00000111		2,3	terminated
Fixed-point overflow	00000000 00001000	36	1,2	completed
Fixed-point divide	00000000 00001001		1,2	suppressed or completed
Decimal overflow	00000000 00001010	37	3	completed
Decimal divide	00000000 00001011		3	suppressed
Exponent overflow	00000000 00001100		1,2	terminated
Exponent underflow	00000000 00001101	38	1,2	completed
Significance	00000000 00001110	39	1,2	completed
Floating-point divide	00000000 00001111		1,2	suppressed

Supervisor Call (old PSW 32, new PSW 96, priority 2)

Instruction bits	00000000 rrrrrrrr		1	completed

External (old PSW 24, new PSW 88, priority 3)

Timer	00000000 1nnnnnnn	7	x	completed
Interrupt key	00000000 n1nnnnnn	7	x	completed
External signal 2	00000000 nn1nnnnn	7	x	completed
External signal 3	00000000 nnn1nnnn	7	x	completed
External signal 4	00000000 nnnn1nnn	7	x	completed
External signal 5	00000000 nnnnn1nn	7	x	completed
External signal 6	00000000 nnnnnn1n	7	x	completed
External signal 7	00000000 nnnnnnn1	7	x	completed

Machine Check (old PSW 48, new PSW 112, priority 1)

Machine malfunction	cccccccc cccccccc	13	x	terminated

NOTES

a	Device address bits
c	Bits of model-dependent code
n	Other external-interruption conditions
r	Bits of R_1 and R_2 field of SUPERVISOR CALL
x	Unpredictable

Instruction Length Recording

CODE	PSW BITS 32-33	INSTRUCTION BITS 0-1	INSTRUCTION LENGTH	FORMAT
0	00		Not available	
1	01	00	One halfword	RR
2	10	01	Two halfwords	RX
2	10	10	Two halfwords	RS or SI
3	11	11	Three halfwords	SS

Program Interruptions

Exceptions resulting from improper specification or use of an instruction and data cause a program interruption.

Operation (OP)

The operation code is not assigned or the assigned operation is not available on the particular CPU.

The operation is suppressed.

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

Privileged Operation (M)

A privileged instruction is encountered in the problem state.

The operation is suppressed.

The instruction-length code is 1 or 2.

NAME	MNEMONIC	FORMAT	ACTION
Diagnose		SI	Suppressed
Halt I/O	HIO	SI	Suppressed
Insert Storage Key	ISK	RR	Suppressed
Load PSW	LPSW	SI	Suppressed
Read Direct	RDD	SI	Suppressed
Set Storage Key	SSK	RR	Suppressed
Set System Mask	SSM	SI	Suppressed
Start I/O	SIO	SI	Suppressed
Test Channel	TCH	SI	Suppressed
Test I/O	TIO	SI	Suppressed
Write Direct	WRD	SI	Suppressed

Execute (EX)

The subject instruction of EXECUTE is another EXECUTE.

The operation is suppressed.

The instruction-length code is 2.

NAME	MNEMONIC	FORMAT	ACTION
Execute	EX	RX	Suppressed

Protection (P)

The key of an instruction or operand location does not match the protection key in the PSW.

The instruction length-code is 0, 2, or 3.

Instructions Subject to Store Protection

When the protection feature contains only the facility to monitor operations changing storage contents, the check is made for every use of the following instructions. In addition, these instructions are subject to a protection exception when both store and fetch violations are monitored. The operation is suppressed on a store violation, except in the case of STORE MULTIPLE, READ DIRECT, TEST AND SET, and variable-length operations, which are terminated.

NAME	MNEMONIC	FORMAT	ACTION
Add Decimal	AP	SS	Terminated
AND	NC	SS	Terminated
AND	NI	SI	Suppressed
Convert to Decimal	CVD	RX	Suppressed
Diagnose	--	SI	Unpredictable
Divide Decimal	DP	SS	Terminated
Edit	ED	SS	Terminated
Edit and Mark	EDMK	SS	Terminated
Exclusive OR	XC	SS	Terminated
Exclusive OR	XI	SI	Suppressed
Move	MVC	SS	Terminated
Move	MVI	SI	Suppressed
Move Numerics	MVN	SS	Terminated
Move with Offset	MVO	SS	Terminated
Move Zones	MVZ	SS	Terminated
Multiply Decimal	MP	SS	Terminated
OR	OC	SS	Terminated
OR	OI	SI	Suppressed
Pack	PACK	SS	Terminated
Read Direct	RRD	SI	Terminated
Store	ST	RX	Suppressed
Store (long)	STD	RX	Suppressed
Store (short)	STE	RX	Suppressed
Store Character	STC	RX	Suppressed
Store Halfword	STH	RX	Suppressed
Store Multiple	STM	RS	Terminated
Test and Set	TS	SI	Terminated
Translate	TR	SS	Terminated
Unpack	UNPK	SS	Terminated
Zero and Add	ZAP	SS	Terminated

Instructions Subject to Fetch Protection

When the protection feature contains the facility to monitor for both store and fetch violations, the following instructions can cause the exception only by a fetch violation. The operation is terminated, except in the case of EXECUTE, which is suppressed.

NAME	MNEMONIC	FORMAT	ACTION
Add	A	RX	Terminated
Add Halfword	AH	RX	Terminated
Add Logical	AL	RX	Terminated
Add Normalized (long)	AD	RX	Terminated
Add Normalized (short)	AE	RX	Terminated
Add Unnormalized (long)	AW	RX	Terminated
Add Unnormalized (short)	AU	RX	Terminated
AND	N	RX	Terminated
Compare	C	RX	Terminated
Compare	CL	RX	Terminated
Compare	CLI	SI	Terminated
Compare	CLC	SS	Terminated
Compare (long)	CD	RX	Terminated
Compare (short)	CE	RX	Terminated
Compare Decimal	CP	SS	Terminated
Compare Halfword	CH	RX	Terminated
Convert to Binary	CVB	RX	Terminated
Divide	D	RX	Terminated
Divide (long)	DD	RX	Terminated
Divide (short)	DE	RX	Terminated
Exclusive OR	X	RX	Terminated
Execute	EX	RX	Suppressed
Insert Character	IC	RX	Terminated
Load	L	RX	Terminated
Load (long)	LD	RX	Terminated
Load (short)	LE	RX	Terminated
Load Multiple	LM	RS	Terminated
Load PSW	LPSW	SI	Terminated
Multiply	M	RX	Terminated
Multiply (long)	MD	RX	Terminated
Multiply (short)	ME	RX	Terminated
Multiply Halfword	MH	RX	Terminated
OR	O	RX	Terminated
Set System Mask	SSM	SI	Terminated
Subtract	S	RX	Terminated
Subtract Halfword	SH	RX	Terminated
Subtract Logical	SL	RX	Terminated
Subtract Normalized (long)	SD	RX	Terminated
Subtract Normalized (short)	SE	RX	Terminated
Subtract Unnormalized (long)	SW	RX	Terminated
Subtract Unnormalized (short)	SU	RX	Terminated
Test Under Mask	TM	SI	Terminated
Translate And Test	TRT	SS	Terminated
Write Direct	WRD	SI	Terminated

Instructions Subject to Store and Fetch Protection

When the protection feature contains the facility to monitor for both store and fetch violations, the following instructions can cause the exception either by a store or fetch violation. The operation is terminated, except in the case of DIAGNOSE, which is unpredictable.

NAME	MNEMONIC	FORMAT	ACTION
Add Decimal	AP	SS	Terminated
AND	NC	SS	Terminated
Diagnose	--	SI	Unpredictable
Divide Decimal	DP	SS	Terminated
Edit	ED	SS	Terminated
Edit And Mark	EDMK	SS	Terminated
Exclusive OR	XC	SS	Terminated
Move	MVC	SS	Terminated
Move Numerics	MVN	SS	Terminated
Move with Offset	MVO	SS	Terminated
Move Zones	MVZ	SS	Terminated
Multiply Decimal	MP	SS	Terminated
OR	OC	SS	Terminated
Pack	PACK	SS	Terminated
Subtract Decimal	SP	SS	Terminated
Test and Set	TS	SI	Terminated
Translate	TR	SS	Terminated
Unpack	UNPK	SS	Terminated
Zero and Add	ZAP	SS	Terminated

Addressing (A)

An address specifies any part of data, instruction, or control words outside the available storage for the particular installation.

In most cases, the operation is terminated. Data in storage remain unchanged, except when designated by valid addresses. The operation is suppressed for CONVERT TO DECIMAL, DIAGNOSE, EXECUTE, immediate instructions (NI, XI, MVI, OI), and certain store-type operations (ST, STC, STH, STD, and STE).

The instruction-length code is normally 1, 2, or 3; however, it may be 0 in the case of a data address.

NAME	MNEMONIC	FORMAT	ACTION
Add	A	RX	Terminated
Add Decimal	AP	SS	Terminated
Add Halfword	AH	RX	Terminated
Add Logical	AL	RX	Terminated
Add Normalized (Long)	AD	RX	Terminated
Add Normalized (Short)	AE	RX	Terminated
Add Unnormalized (Long)	AW	RX	Terminated
Add Unnormalized (Short)	AU	RX	Terminated
AND	N	RX	Terminated
AND	NI	SI	Suppressed
AND	NC	SS	Terminated
Compare	C	RX	Terminated
Compare Decimal	CP	SS	Terminated
Compare Halfword	CH	RX	Terminated
Compare Logical	CL	RX	Terminated
Compare Logical	CLI	SI	Terminated
Compare Logical	CLC	SS	Terminated
Compare (Long)	CD	RX	Terminated
Compare (Short)	CE	RX	Terminated
Convert to Binary	CVB	RX	Terminated
Convert to Decimal	CVD	RX	Suppressed
Diagnose		SI	Suppressed
Divide	D	RX	Terminated
Divide Decimal	DP	SS	Terminated
Divide (Long)	DD	RX	Terminated
Divide (Short)	DE	RX	Terminated
Edit	ED	SS	Terminated
Edit and Mark	EDMK	SS	Terminated
Exclusive OR	X	RX	Terminated
Exclusive OR	XI	SI	Suppressed
Exclusive OR	XC	SS	Terminated
Execute	EX	RX	Suppressed
Insert Character	IC	RX	Terminated
Insert Storage Key	ISK	RR	Terminated
Load	L	RX	Terminated
Load Halfword	LH	RX	Terminated
Load (Long)	LD	RX	Terminated
Load Multiple	LM	RS	Terminated
Load PSW	LPSW	SI	Terminated
Load (Short)	LE	RX	Terminated
Move	MVI	SI	Suppressed
Move	MVC	SS	Terminated
Move Numerics	MVN	SS	Terminated
Move with Offset	MVO	SS	Terminated
Move Zones	MVZ	SS	Terminated
Multiply	M	RX	Terminated
Multiply Decimal	MP	SS	Terminated
Multiply Halfword	MH	RX	Terminated
Multiply (Long)	MD	RX	Terminated
Multiply (Short)	ME	RX	Terminated
OR	O	RX	Terminated
OR	OI	SI	Suppressed
OR	OC	SS	Terminated
Pack	PACK	SS	Terminated
Read Direct	RDD	SI	Terminated
Set Storage Key	SSK	RR	Suppressed
Set System Mask	SSM	SI	Terminated
Store	ST	RX	Suppressed
Store Character	STC	RX	Suppressed
Store Halfword	STH	RX	Suppressed
Store (Long)	STD	RX	Suppressed
Store Multiple	STM	RS	Terminated
Store (Short)	STE	RX	Suppressed
Subtract	S	RX	Terminated
Subtract Decimal	SP	SS	Terminated
Subtract Halfword	SH	RX	Terminated
Subtract Logical	SL	RX	Terminated
Subtract Normalized (Long)	SD	RX	Terminated
Subtract Normalized (Short)	SE	RX	Terminated
Subtract Unnormalized (Long)	SW	RX	Terminated
Subtract Unnormalized (Short)	SU	RX	Terminated
Test Under Mask	TM	SI	Terminated
Test and Set	TS	SI	Terminated
Translate	TR	SS	Terminated
Translate and Test	TRT	SS	Terminated
Unpack	UNPK	SS	Terminated
Write Direct	WRD	SI	Terminated
Zero and Add	ZAP	SS	Terminated

The addressing interruption can occur in normal sequential operation following branching, LOAD PSW, interruption, or manual operation. Instruction execution is suppressed.

Specification (S)

1. A data, instruction, or control-word address does not specify an integral boundary for the unit of information.

2. The R_1 field of an instruction specifies an odd register address for a pair of general registers that contain a 64-bit operand.

3. A floating-point register address other than 0, 2, 4, or 6 is specified.

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

5. The first operand field is shorter than or equal to the second operand field in decimal multiplication or division.

6. The block address specified in SET STORAGE KEY or INSERT STORAGE KEY has the four low-order bits not all zero.

7. A PSW with a nonzero protection key is encountered when the protection feature is not installed.

In all of these cases the operation is suppressed.

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

NAME	MNEMONIC	FORMAT	ACTION	NOTE
Add	A	RX	Suppressed	4
Add Halfword	AH	RX	Suppressed	2
Add Logical	AL	RX	Suppressed	4
Add Normalized (Long)	ADR	RR	Suppressed	3
Add Normalized (Long)	AD	RX	Suppressed	3,8
Add Normalized (Short)	AER	RR	Suppressed	3
Add Normalized (Short)	AE	RX	Suppressed	3,4

NAME	MNEMONIC	FORMAT	ACTION	NOTE
Add Unnormalized (Long)	AWR	RR	Suppressed	3
Add Unnormalized (Long)	AW	RX	Suppressed	3,8
Add Unnormalized (Short)	AUR	RR	Suppressed	3
Add Unnormalized (Short)	AU	RX	Suppressed	3,4
AND	N	RX	Suppressed	4
Compare	C	RX	Suppressed	4
Compare Halfword	CH	RX	Suppressed	2
Compare Logical	CL	RX	Suppressed	4
Compare (Long)	CDR	RR	Suppressed	3
Compare (Long)	CD	RX	Suppressed	3,8
Compare (Short)	CER	RR	Suppressed	3
Compare (Short)	CE	RX	Suppressed	3,4
Convert to Binary	CVB	RX	Suppressed	8
Convert to Decimal	CVD	RX	Suppressed	8
Diagnose		SI	Suppressed	
Divide	DR	RR	Suppressed	1
Divide	D	RX	Suppressed	1,4
Divide Decimal	DP	SS	Suppressed	5
Divide (Long)	DDR	RR	Suppressed	3
Divide (Long)	DD	RX	Suppressed	3,8
Divide (Short)	DER	RR	Suppressed	3
Divide (Short)	DE	RX	Suppressed	3,4
Exclusive OR	X	RX	Suppressed	4
Execute	EX	RX	Suppressed	2
Halve (Long)	HDR	RR	Suppressed	3
Halve (Short)	HER	RR	Suppressed	3
Insert Storage Key	ISK	RR	Suppressed	7
Load	L	RX	Suppressed	4
Load and Test (Long)	LTDR	RR	Suppressed	3
Load and Test (Short)	LTER	RR	Suppressed	3
Load Complement (Long)	LCDR	RR	Suppressed	3
Load Complement (Short)	LCER	RR	Suppressed	3
Load Halfword	LH	RX	Suppressed	2
Load (Long)	LDR	RR	Suppressed	3
Load (Long)	LD	RX	Suppressed	3,8
Load Multiple	LM	RS	Suppressed	4
Load Negative (Long)	LNDR	RR	Suppressed	3
Load Negative (Short)	LNER	RR	Suppressed	3
Load Positive (Long)	LPDR	RR	Suppressed	3
Load Positive (Short)	LPER	RR	Suppressed	3
Load PSW	LPSW	SI	Suppressed	8
Load (Short)	LER	RR	Suppressed	3
Load (Short)	LE	RX	Suppressed	3,4
Multiply	MR	RR	Suppressed	1
Multiply	M	RX	Suppressed	1,4
Multiply Decimal	MP	SS	Suppressed	5
Multiply Halfword	MH	RX	Suppressed	2
Multiply (Long)	MDR	RR	Suppressed	3
Multiply (Long)	MD	RX	Suppressed	3,8
Multiply (Short)	MER	RR	Suppressed	3
Multiply (Short)	ME	RX	Suppressed	3,4
OR	O	RX	Suppressed	4
Set Storage Key	SSK	RR	Suppressed	7
Shift Left Double	SLDA	RS	Suppressed	1
Shift Left Double Logical	SLDL	RS	Suppressed	1
Shift Right Double	SRDA	RS	Suppressed	1
Shift Right Double Logical	SRDL	RS	Suppressed	1
Store	ST	RX	Suppressed	4
Store Halfword	STH	RX	Suppressed	2
Store (Long)	STD	RX	Suppressed	3,8
Store Multiple	STM	RS	Suppressed	4
Store (Short)	STE	RX	Suppressed	3,4
Subtract	S	RX	Suppressed	4
Subtract Halfword	SH	RX	Suppressed	2
Subtract Logical	SL	RX	Suppressed	4
Subtract Normalized (Long)	SDR	RR	Suppressed	3
Subtract Normalized (Long)	SD	RX	Suppressed	3,8
Subtract Normalized (Short)	SER	RR	Suppressed	3
Subtract Normalized (Short)	SE	RX	Suppressed	3,4
Subtract Unnormalized (Long)	SWR	RR	Suppressed	3
Subtract Unnormalized (Long)	SW	RX	Suppressed	3,8
Subtract Unnormalized (Short)	SUR	RR	Suppressed	3
Subtract Unnormalized (Short)	SU	RX	Suppressed	3,4

The specification interruption can occur in normal sequential operation following branching, LOAD PSW, interruption, or manual operation (Note 1).

SPECIFICATION INTERRUPTION NOTES

1 Even register specification
2 Two-byte unit of information specification
3 Floating-point register specification
4 Four-byte unit of information specification
5 Decimal multiplier or divisor size specification
6 Zero protection key specification
7 Block address specification
8 Eight-byte unit of information specification

Data (D)

1. The sign or digit codes of operands in decimal arithmetic, or editing operations, or CONVERT TO BINARY, are incorrect.

2. Fields in decimal arithmetic overlap incorrectly.

3. The decimal multiplicand has too many high-order significant digits.

The operation is terminated in all three cases.

The instruction-length code is 2 or 3.

NAME	MNEMONIC	FORMAT	ACTION	NOTE
Add Decimal	AP	SS	Terminated	1,2
Compare Decimal	CP	SS	Terminated	1,2
Convert to Binary	CVB	RX	Terminated	1
Divide Decimal	DP	SS	Terminated	1,2
Edit	ED	SS	Terminated	1
Edit and Mark	EDMK	SS	Terminated	1
Multiply Decimal	MP	SS	Terminated	1,2,3
Subtract Decimal	SP	SS	Terminated	1,2
Zero and Add	ZAP	SS	Terminated	1,2

DATA INTERRUPTION NOTES

1 All instructions listed may have incorrect sign or digit codes.
2 Overlapping fields
3 Multiplicand length

Fixed-Point Overflow (IF)

A high-order carry occurs or high-order significant bits are lost in fixed-point addition, subtraction, shifting, or sign-control operations.

The operation is completed by ignoring the information placed outside the register. The interruption may be masked by PSW bit 36.

The instruction-length code is 1 or 2.

NAME	MNEMONIC	FORMAT	ACTION
Add	AR	RR	Completed
Add	A	RX	Completed
Add Halfword	AH	RX	Completed
Load Complement	LCR	RR	Completed
Load Positive	LPR	RR	Completed
Shift Left Double	SLDA	RS	Completed
Shift Left Single	SLA	RS	Completed
Subtract	SR	RR	Completed
Subtract	S	RX	Completed
Subtract Halfword	SH	RX	Completed

Fixed-Point Divide (IK)

1. The quotient exceeds the register size in fixed-point division, including division by zero.

2. The result of CONVERT TO BINARY exceeds 31 bits.

Division is suppressed. Conversion is completed by ignoring the information placed outside the register.

The instruction-length code is 1 or 2.

NAME	MNEMONIC	FORMAT	ACTION
Convert to Binary	CVB	RX	Completed
Divide	DR	RR	Suppressed
Divide	D	RX	Suppressed

Decimal Overflow (DF)

The destination field is too small to contain the result field in decimal operations.

The operation is completed by ignoring the overflow information. The interruption may be masked by PSW bit 37.

The interruption-length code is 3.

NAME	MNEMONIC	FORMAT	ACTION
Add Decimal	AP	SS	Completed
Subtract Decimal	SP	SS	Completed
Zero and Add	ZAP	SS	Completed

Decimal Divide (DK)

The quotient exceeds the specified data field.

The operation is suppressed.

The instruction-length code is 3.

NAME	MNEMONIC	FORMAT	ACTION
Divide Decimal	DP	SS	Suppressed

Exponent Overflow (E)

The result characteristic exceeds 127 in floating-point addition, subtraction, multiplication, or division.

The operation is terminated.

The instruction-length code is 1 or 2.

NAME	MNEMONIC	FORMAT	ACTION
Add Normalized (Long)	ADR	RR	Terminated
Add Normalized (Long)	AD	RX	Terminated
Add Normalized (Short)	AER	RR	Terminated
Add Normalized (Short)	AE	RX	Terminated
Add Unnormalized (Long)	AWR	RR	Terminated
Add Unnormalized (Long)	AW	RX	Terminated
Add Unnormalized (Short)	AUR	RR	Terminated
Add Unnormalized (Short)	AU	RX	Terminated
Divide (Long)	DDR	RR	Terminated
Divide (Long)	DD	RX	Terminated
Divide (Short)	DER	RR	Terminated
Divide (Short)	DE	RX	Terminated
Multiply (Long)	MDR	RR	Terminated
Multiply (Long)	MD	RX	Terminated
Multiply (Short)	MER	RR	Terminated
Multiply (Short)	ME	RX	Terminated
Subtract Normalized (Long)	SDR	RR	Terminated
Subtract Normalized (Long)	SD	RX	Terminated
Subtract Normalized (Short)	SER	RR	Terminated
Subtract Normalized (Short)	SE	RX	Terminated
Subtract Unnormalized (Long)	SWR	RR	Terminated
Subtract Unnormalized (Long)	SW	RX	Terminated
Subtract Unnormalized (Short)	SUR	RR	Terminated
Subtract Unnormalized (Short)	SU	RX	Terminated

Exponent Underflow (U)

The result characteristic is less than zero in floating-point addition, subtraction, multiplication, or division.

The operation is completed by making the result of the operation a true zero. The interruption may be masked by PSW bit 38.

The instruction-length code is 1 or 2.

NAME	MNEMONIC	FORMAT	ACTION
Add Normalized (Long)	ADR	RR	Completed
Add Normalized (Long)	AD	RX	Completed
Add Normalized (Short)	AER	RR	Completed
Add Normalized (Short)	AE	RX	Completed
Divide (Long)	DDR	RR	Completed
Divide (Long)	DD	RX	Completed
Divide (Short)	DER	RR	Completed
Divide (Short)	DE	RX	Completed
Multiply (Long)	MDR	RR	Completed
Multiply (Long)	MD	RX	Completed
Multiply (Short)	MER	RR	Completed
Multiply (Short)	ME	RX	Completed
Subtract Normalized (Long)	SDR	RR	Completed
Subtract Normalized (Long)	SD	RX	Completed
Subtract Normalized (Short)	SER	RR	Completed
Subtract Normalized (Short)	SE	RX	Completed

Significance (LS)

The result of a floating-point addition or subtraction has an all-zero fraction.

The operation is completed. The interruption may be masked by PSW bit 39. The manner in which the operation is completed is determined by the mask bit.

The instruction-length code is 1 or 2.

NAME	MNEMONIC	FORMAT	ACTION
Add Normalized (Long)	ADR	RR	Completed
Add Normalized (Long)	AD	RX	Completed
Add Normalized (Short)	AER	RR	Completed
Add Normalized (Short)	AE	RX	Completed
Add Unnormalized (Long)	AWR	RR	Completed
Add Unnormalized (Long)	AW	RX	Completed
Add Unnormalized (Short)	AUR	RR	Completed
Add Unnormalized (Short)	AU	RX	Completed
Subtract Normalized (Long)	SDR	RR	Completed
Subtract Normalized (Long)	SD	RX	Completed
Subtract Normalized (Short)	SER	RR	Completed
Subtract Normalized (Short)	SE	RX	Completed
Subtract Unnormalized (Long)	SWR	RR	Completed
Subtract Unnormalized (Long)	SW	RX	Completed
Subtract Unnormalized (Short)	SUR	RR	Completed
Subtract Unnormalized (Short)	SU	RX	Completed

Floating-Point Divide (FK)

Division by a floating-point number with zero fraction is attempted.

The operation is suppressed.

The instruction-length code is 1 or 2.

NAME	MNEMONIC	FORMAT	ACTION
Divide (Long)	DDR	RR	Suppressed
Divide (Long)	DD	RX	Suppressed
Divide (Short)	DER	RR	Suppressed
Divide (Short)	DE	RX	Suppressed

Functions that May Differ Among Models

Instruction Execution

In the editing operations, overlapping fields give unpredictable results.

Equipment connected to the hold-in line of READ DIRECT should be so constructed that the hold signal will be removed when READ DIRECT is performed. Excessive duration of this instruction may result in incomplete updating of the timer.

The purpose of the I_2 field and the operand address in the SI format of DIAGNOSE may be further defined for a particular CPU and its appropriate diagnostic

procedures. Similarly the number of low-order address bits that must be zero is further specified for a particular CPU. When the address does not have the required number of low-order zeros, a specification exception is recognized and causes a program interruption.

Whether DIAGNOSE is subject to protection action depends on the model.

The diagnose operation is completed either by taking the next sequential instruction or by obtaining a new PSW from location 112. The diagnostic procedure may affect the problem, supervisor, and interruptable states of the CPU, and the contents of storage registers and timer, as well as the progress of I/O operations.

Instruction Termination

Only one program interruption occurs for a given instruction. The old PSW always identifies a valid cause. This does not preclude simultaneous occurrence of any other causes. Which of several causes is identified may vary from one occasion to the next and from one model to another.

When instruction execution is terminated by an interruption, all, part, or none of the result may be stored. The result data, therefore, are unpredictable. The setting of the condition code, if called for, may also be unpredictable. In general, the results of the operation should not be used for further computation.

Cases of instruction termination for a program interruption are:

Protection: The key of the addressed storage location does not match the protection key in the PSW. A store violation causes the operation to be terminated in the case of STORE MULTIPLE, READ DIRECT, TEST AND SET, and variable-length operations. Protected storage remains unchanged. The timing signals of READ DIRECT may have been made available. The operation is terminated on a fetch violation, except for EXECUTE, which is suppressed.

Addressing: An address specifies any part of data, instruction, or control word outside the available storage for the particular installation. In most cases the operation is terminated. Data in storage remain unchanged, except when designated by valid addresses.

When part of an operand in CLC is specified in an unavailable location, the comparison may end at an inequality, or the operation may be terminated by the addressing exception, even though inequality could have been established from the available operand parts.

Data: The sign or digit codes of operands in decimal arithmetic, CONVERT TO BINARY, or editing operations are incorrect, or fields in decimal arithmetic overlap incorrectly, or the decimal multiplicand has too many high-order significant digits. The operation is terminated in all three cases. The condition code setting, if called for, is unpredictable for protection, addressing, and data exceptions.

Exponent Overflow: The result exponent of an ADD, SUBTRACT, MULTIPLY, or DIVIDE overflows and the result fraction is not zero. The operation is terminated. The condition code is set to 3 for ADD and SUBTRACT, and remains unchanged for MULTIPLY and DIVIDE.

Machine-Check Interruption

For a machine-check interruption, the old PSW is stored at location 48, and depending on the model, the interruption code may identify the type of malfunction. The state of the CPU is scanned out into the storage area starting with location 128 and extending through as many words as are required by the given CPU. The new PSW is fetched from location 112. Proper execution of these steps depends on the nature of the machine check. A change in the machine-check mask bit due to the loading of a new PSW results in a change in the treatment of machine checks. Depending upon the nature of a machine check, the old treatment may still be in force for several cycles. Machine checks that occur in operations executed by I/O channels may either cause a machine-check interruption or are recorded in the CSW for that operation.

Instruction-Length Code

The instruction-length code is predictable only for program and supervisor-call interruptions. For I/O and external interruptions, the interruption is not caused by the last interpreted instruction, and the code is not predictable for these classes of interruptions. For machine-check interruptions, the setting of the code is a function of the malfunction and therefore unpredictable.

For the supervisor-call interruption the instruction-length code is 1, indicating the halfword length of SUPERVISOR CALL; for the program interruptions, the codes 1, 2, and 3 indicate the instruction length in halfwords. The code 0 is reserved for program interruptions where the length of the instruction is not available because of certain overlap conditions in instruction fetching. In those cases, the instruction address in the old PSW does not represent the next instruction address. The instruction-length code 0 can occur only for a program interruption caused by a protected or unavailable data address.

Timer

Updating of the timer may be omitted when I/O data transmission approaches the limit of storage capability and when a channel sharing CPU equipment and operating in burst mode causes CPU activity to be locked out.

When a high-resolution timer is installed, the following rules apply:

1. Use of the contents of location 83 as a source of an instruction yields unpredictable results.
2. The storing of data by the channel at location 83 has an unpredictable effect on the eight low-order bits of the timer value, while fetching of data by the channel from word location 83 yields unpredictable results.
3. In a system having shared storage, storing in the low-order byte of another CPU's timer has an unpredictable effect on the eight low-order bits of the timer value, while a fetch-type operation to the timer location of another CPU causes the content of the low-order byte to be unpredictable.

System Control Panel

The system-reset function may correct the parity of general and floating-point registers, as well as the parity of the PSW.

Pressing the start key after a system reset without first introducing a new instruction address yields unpredictable results.

The number of data switches is sufficient to allow storing of a full physical storage word. Correct parity generation is provided. In some models, either correct or incorrect parity is generated under switch control.

The data in the storage, general register or floating-point register location, or the instruction-address part of the PSW as specified by the address switches and the storage-select switch can be displayed by the display key. When the location designated by the address switches and storage-select switch is not available, the displayed information is unpredictable. In some models, the instruction address is permanently displayed and hence is not explicitly selected.

When the address-comparison switches are set to the stop position, the address in the address switches is compared against the value of the instruction address on some models, and against all addresses on others. Comparison includes only that part of the instruction address corresponding to the physical word size of storage.

Comparison of the entire halfword instruction address is provided in some models, as is the ability to compare data addresses.

The test light may be on when one or more diagnostic functions under control of DIAGNOSE are activated, or when certain abnormal circuit breaker or thermal conditions occur.

Normal Channel Operation

Channel capacity depends on the way I/O operations are programmed and the activity in the rest of the system. In view of this, an evaluation of the ability of a specific I/O configuration to function concurrently must be based on the application. Two systems employing identical complements of I/O devices may be able to execute certain programs in common, but it is possible that other programs requiring, for example, data chaining may not run on one of the systems.

The time when the interruption due to the PCI flag occurs depends on the model and the current activity. The channel may cause the interruption an unpredictable time after control of the operation is taken over by the CCW containing the PCI flag.

The content of the count field in a CSW associated with an interruption due to the PCI flag is unpredictable. The content of the count field depends on the model and its current activity.

When the interruption condition due to the PCI flag has been delayed until the operation at the subchannel has been terminated, two interruptions from the subchannel still may take place, with the first interruption indicating and clearing the PCI condition alone, and the second providing the CSW associated with the ending status. Whether one or two interruptions occur depends on the model, and on whether the PCI condition has been assigned the highest priority for interruption at the time of termination.

When the channel has established which device on the channel will cause the next I/O interruption, the identity of the device is preserved in the channel. Except for conditions associated with termination of an operation at the subchannel, the current assignment of priority for interruptions among devices may or may not be canceled when START I/O or TEST I/O is issued to the channel, depending upon the model.

The assignment of priority among requests for interruption from channels is based on the type of channel. The priorities of selector channels are in the order of their addresses, with channel 1 having the highest priority. The interruption priority of the multiplexor channel is not fixed, and depends on the model and the current activity in the channel.

Channel Programming Errors

A data address referring to a location not provided in the model normally causes program check when the device offers a byte of data to be placed at the nonexistent location or requests a byte from that location. Models in which the channel does not have the capacity to address 16,777,216 bytes of storage cause program check whenever the address is found to exceed the addressing capacity of the channel.

In the following cases, action depends on the addressing capacity of the model.

1. When the data address in the CCW designated by the CAW exceeds the addressing capacity of the model, the I/O operation is not initiated and the CSW is stored during the execution of START I/O. Normally an invalid data address does not preclude the initiation of the operation.

2. When the data address in a CCW fetched during command chaining exceeds the addressing capacity of the model, the I/O operation is not initiated.

3. When a CCW fetched on data chaining contains an address exceeding the addressing capacity of the model and the device signals channel end immediately upon transferring the last byte designated by the preceding CCW, program check is indicated to the program. Normally, program check is not indicated unless the device attempts to transfer one more byte of data.

4. Data addresses are not checked for validity during skipping, except that the initial data address in the CCW cannot exceed the addressing capacity of the model.

When the channel detects chaining check, program check, or protection check, the content of the count field in the associated CSW is unpredictable.

When the channel detects a programming error in the CAW or in the first CCW, the PCI bit may unpredictably appear in a CSW stored by START I/O without the PCI flag being on in the first CCW associated with the START I/O.

When a programming error occurs in the information placed in the CAW or CCW and the addressed channel or subchannel is working, either condition code 1 or 2 may be set, depending on the model. Similarly, either code 1 or 3 may be set when a programming error occurs and a part of the addressed I/O system is not operational.

When a programming error occurs and the addressed device contains an interruption condition, with the channel and subchannel in the available state, START I/O may or may not clear the interruption condition, depending on the type of error and the model. If the instruction has caused the device to be interrogated, as indicated by the presence of the busy bit in the CSW, the interruption condition has been cleared, and the CSW contains program check, as well as the status from the device.

When the channel detects several error conditions, all conditions may be indicated or only one may appear in the CSW, depending on the condition and the model.

Channel Equipment Errors

Parity errors detected by the channel on data sent to or received from the I/O device on some models cause the current operation to be terminated. When the channel and the CPU share common equipment, parity errors on data may cause malfunction reset to be performed. The recovery procedure in the channel and subsequent state of the subchannel upon a malfunction reset depend on the model.

Detection of channel control check or interface control check causes the current operation, if any, to be immediately terminated and causes the channel to perform the malfunction-reset function. The recovery procedure in the channel and the subsequent state of the subchannel upon a malfunction reset depend on the model.

The contents of the CSW, as well as the address in the PSW identifying the I/O device, are unpredictable upon the detection of a channel-control-check condition.

Some channels can tolerate an absence of data transfer during a burst mode operation, such as occurs when reading a long gap on tape, for not more than approximately one-half minute. Equipment malfunction may be indicated when an absence of data transfer exceeds this time.

Execution of malfunction reset in the channel depends on the type of error and model. It may cause all operations in the channel to be terminated and all operational subchannels to be reset to the available state. The channel may send the malfunction-reset signal to the device connected to the channel at the time the malfunctioning is detected, or a channel sharing common equipment with the CPU may send the system-reset signal to all devices attached to the channel.

The method of processing a request for interruption due to equipment malfunctioning, as indicated by the presence of the channel-control-check and interface-control-check conditions, depends on the model. In channels sharing common equipment with the CPU, malfunctioning detected by the channel may be indicated by the machine-check interruption.

Alphabetic List of Instructions

The listings in the TYPE and EXCEPTIONS columns mean:

A	Addressing exception
C	Condition code is set
D	Data exception
DF	Decimal-overflow exception
DK	Decimal-divide exception
E	Exponent-overflow exception
EX	Execute exception
F	Floating-point feature
FK	Floating-point divide exception
IF	Fixed-point overflow exception
IK	Fixed-point divide exception
L	New condition code loaded
LS	Significance exception
M	Privileged-operation exception
P	Protection exception
S	Specification exception
T	Decimal feature
U	Exponent-underflow exception
Y	Direct control feature
Z	Protection feature

NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE	PG.
Add	AR	RR C	IF	1A	28
Add	A	RX C	P,A,S, IF	5A	28
Add Decimal	AP	SS T,C	P,A, D, DF	FA	37
Add Halfword	AH	RX C	P,A,S, IF	4A	28
Add Logical	ALR	RR C		1E	28
Add Logical	AL	RX C	P,A,S	5E	28
Add Normalized (Long)	ADR	RR F,C	S,U,E,LS	2A	45
Add Normalized (Long)	AD	RX F,C	P,A,S,U,E,LS	6A	45
Add Normalized (Short)	AER	RR F,C	S,U,E,LS	3A	45
Add Normalized (Short)	AE	RX F,C	P,A,S,U,E,LS	7A	45
Add Unnormalized (Long)	AWR	RR F,C	S, E,LS	2E	46
Add Unnormalized (Long)	AW	RX F,C	P,A,S, E,LS	6E	46
Add Unnormalized (Short)	AUR	RR F,C	S, E,LS	3E	46
Add Unnormalized (Short)	AU	RX F,C	P,A,S, E,LS	7E	46
AND	NR	RR C		14	55
AND	N	RX C	P,A,S	54	55
AND	NI	SI C	P,A	94	55
AND	NC	SS C	P,A	D4	55
Branch and Link	BALR	RR		05	66
Branch and Link	BAL	RX		45	66
Branch on Condition	BCR	RR		07	65
Branch on Condition	BC	RX		47	65
Branch on Count	BCTR	RR		06	66
Branch on Count	BCT	RX		46	66
Branch on Index High	BXH	RS		86	66
Branch on Index Low or Equal	BXLE	RS		87	66.1
Compare	CR	RR C		19	30
Compare	C	RX C	P,A,S	59	30
Compare Decimal	CP	SS T,C	P,A, D	F9	38
Compare Halfword	CH	RX C	P,A,S	49	30
Compare Logical	CLR	RR C		15	54
Compare Logical	CL	RX C	P,A,S	55	54
Compare Logical	CLI	SI C	P,A	95	54
Compare Logical	CLC	SS C	P,A	D5	54
Compare (Long)	CDR	RR F,C	S	29	47
Compare (Long)	CD	RX F,C	P,A,S	69	47
Compare (Short)	CER	RR F,C	S	39	47
Compare (Short)	CE	RX F,C	P,A,S	79	47
Convert to Binary	CVB	RX	P,A,S,D, IK	4F	31
Convert to Decimal	CVD	RX	P,A,S	4E	32
Diagnose		SI	M,P,A,S	83	76
Divide	DR	RR	S, IK	1D	31
Divide	D	RX	P,A,S, IK	5D	31
Divide Decimal	DP	SS T	P,A,S,D, DK	FD	38
Divide (Long)	DDR	RR F	S,U,E,FK	2D	49
Divide (Long)	DD	RX F	P,A,S,U,E,FK	6D	49
Divide (Short)	DER	RR F	S,U,E,FK	3D	49
Divide (Short)	DE	RX F	P,A,S,U,E,FK	7D	49
Edit	ED	SS T,C	P,A, D	DE	57
Edit and Mark	EDMK	SS T,C	P,A, D	DF	60
Exclusive OR	XR	RR C		17	55
Exclusive OR	X	RX C	P,A,S	57	55
Exclusive OR	XI	SI C	P,A	97	55
Exclusive OR	XC	SS C	P,A	D7	55
Execute	EX	RX	P,A,S, EX	44	67
Halt I/O	HIO	SI CM		9E	96
Halve (Long)	HDR	RR F	S	24	48
Halve (Short)	HER	RR F	S	34	48
Insert Character	IC	RX	P,A	43	56
Insert Storage Key	ISK	RR Z	M, A,S	09	74
Load	LR	RR		18	26
Load	L	RX	P,A,S	58	26
Load Address	LA	RX		41	56
Load and Test	LTR	RR C		12	26
Load and Test (Long)	LTDR	RR F,C	S	22	44
Load and Test (Short)	LTER	RR F,C	S	32	44
Load Complement	LCR	RR C	IF	13	27
Load Complement (Long)	LCDR	RR F,C	S	23	44
Load Complement (Short)	LCER	RR F,C	S	33	44
Load Halfword	LH	RX	P,A,S	48	26
Load (Long)	LDR	RR F	S	28	44
Load (Long)	LD	RX F	P,A,S	68	44
Load Multiple	LM	RS	P,A,S	98	27
Load Negative	LNR	RR C		11	27
Load Negative (Long)	LNDR	RR F,C	S	21	45
Load Negative (Short)	LNER	RR F,C	S	31	45
Load Positive	LPR	RR C	IF	10	27
Load Positive (Long)	LPDR	RR F,C	S	20	44
Load Positive (Short)	LPER	RR F,C	S	30	44
Load PSW	LPSW	SI L	M,P,A,S	82	73
Load (Short)	LER	RR F	S	38	44
Load (Short)	LE	RX F	P,A,S	78	44
Move	MVI	SI	P,A	92	53
Move	MVC	SS	P,A	D2	53
Move Numerics	MVN	SS	P,A	D1	54
Move with Offset	MVO	SS	P,A	F1	40
Move Zones	MVZ	SS	P,A	D3	54

NAME	MNEMONIC	TYPE	EXCEPTIONS	CODE	PG.
Multiply	MR	RR	S	1C	30
Multiply	M	RX	P,A,S	5C	30
Multiply Decimal	MP	SS T	P,A,S,D	FC	38
Multiply Halfword	MH	RX	P,A,S	4C	30
Multiply (Long)	MDR	RR F	S,U,E	2C	48
Multiply (Long)	MD	RX F	P,A,S,U,E	6C	48
Multiply (Short)	MER	RR F	S,U,E	3C	48
Multiply (Short)	ME	RX F	P,A,S,U,E	7C	48
OR	OR	RR C		16	55
OR	O	RX C	P,A,S	56	55
OR	OI	SI C	P,A	96	55
OR	OC	SS C	P,A	D6	55
Pack	PACK	SS	P,A	F2	39
Read Direct	RDD	SI Y	M,P,A	85	75
Set Program Mask	SPM	RR L		04	73
Set Storage Key	SSK	RR Z	M, A,S	08	74
Set System Mask	SSM	SI	M,P,A	80	74
Shift Left Double	SLDA	RS C	S, IF	8F	33
Shift Left Double Logical	SLDL	RS	S	8D	60
Shift Left Single	SLA	RS C	IF	8B	32
Shift Left Single Logical	SLL	RS		89	60
Shift Right Double	SRDA	RS C	S	8E	34
Shift Right Double Logical	SRDL	RS	S	8C	61
Shift Right Single	SRA	RS C		8A	33
Shift Right Single Logical	SRL	RS		88	60
Start I/O	SIO	SI CM		9C	94
Store	ST	RX	P,A,S	50	32
Store Character	STC	RX	P,A	42	56
Store Halfword	STH	RX	P,A,S	40	32
Store (Long)	STD	RX F	P,A,S	60	50
Store Multiple	STM	RS	P,A,S	90	32
Store (Short)	STE	RX F	P,A,S	70	50
Subtract	SR	RR C	IF	1B	29
Subtract	S	RX C	P,A,S, IF	5B	29
Subtract Decimal	SP	SS T,C	P,A, D, DF	FB	37
Subtract Halfword	SH	RX C	P,A,S, IF	4B	29
Subtract Logical	SLR	RR C		1F	29
Subtract Logical	SL	RX C	P,A,S	5F	29
Subtract Normalized (Long)	SDR	RR F,C	S,U,E,LS	2B	46
Subtract Normalized (Long)	SD	RX F,C	P,A,S,U,E,LS	6B	46
Subtract Normalized (Short)	SER	RR F,C	S,U,E,LS	3B	46
Subtract Normalized (Short)	SE	RX F,C	P,A,S,U,E,LS	7B	46
Subtract Unnormalized (Long)	SWR	RR F,C	S, E,LS	2F	47
Subtract Unnormalized (Long)	SW	RX F,C	P,A,S, E,LS	6F	47
Subtract Unnormalized (Short)	SUR	RR F,C	S, E,LS	3F	47
Subtract Unnormalized (Short)	SU	RX F,C	P,A,S, E,LS	7F	47
Supervisor Call	SVC	RR		0A	74
Test and Set	TS	SI C	P,A	93	74
Test Channel	TCH	SI CM		9F	98
Test I/O	TIO	SI CM		9D	95
Test Under Mask	TM	SI C	P,A	91	56
Translate	TR	SS	P,A	DC	57
Translate and Test	TRT	SS C	P,A	DD	57
Unpack	UNPK	SS	P,A	F3	39
Write Direct	WRD	SI Y	M,P,A	84	75
Zero and Add	ZAP	SS T,C	P,A, D, DF	F8	37

List of Instructions by Set and Feature

Standard Instruction Set

NAME	MNEMONIC	TYPE		CODE
Add	AR	RR	C	1A
Add	A	RX	C	5A
Add Halfword	AH	RX	C	4A
Add Logical	ALR	RR	C	1E
Add Logical	AL	RX	C	5E
AND	NR	RR	C	14
AND	N	RX	C	54
AND	NI	SI	C	94
AND	NC	SS	C	D4
Branch and Link	BALR	RR		05
Branch and Link	BAL	RX		45
Branch on Condition	BCR	RR		07
Branch on Condition	BC	RX		47
Branch on Count	BCTR	RR		06
Branch on Count	BCT	RX		46
Branch on Index High	BXH	RS		86
Branch on Index Low or Equal	BXLE	RS		87
Compare	CR	RR	C	19
Compare	C	RX	C	59
Compare Halfword	CH	RX	C	49
Compare Logical	CLR	RR	C	15
Compare Logical	CL	RX	C	55
Compare Logical	CLC	SS	C	D5
Compare Logical	CLI	SI	C	95
Convert to Binary	CVB	RX		4F
Convert to Decimal	CVD	RX		4E
Diagnose		SI		83
Divide	DR	RR		1D
Divide	D	RX		5D
Exclusive OR	XR	RR	C	17
Exclusive OR	X	RX	C	57
Exclusive OR	XI	SI	C	97
Exclusive OR	XC	SS	C	D7
Execute	EX	RX		44
Halt I/O	HIO	SI	C	9E
Insert Character	IC	RX		43
Load	LR	RR		18
Load	L	RX		58
Load Address	LA	RX		41
Load and Test	LTR	RR	C	12
Load Complement	LCR	RR	C	13
Load Halfword	LH	RX		48
Load Multiple	LM	RS		98
Load Negative	LNR	RR	C	11
Load Positive	LPR	RR	C	10
Load PSW	LPSW	SI	L	82
Move	MVI	SI		92
Move	MVC	SS		D2
Move Numerics	MVN	SS		D1
Move with Offset	MVO	SS		F1
Move Zones	MVZ	SS		D3
Multiply	MR	RR		1C
Multiply	M	RX		5C
Multiply Halfword	MH	RX		4C
OR	OR	RR	C	16
OR	O	RX	C	56
OR	OI	SI	C	96
OR	OC	SS	C	D6
Pack	PACK	SS		F2

NAME	MNEMONIC	TYPE		CODE
Set Program Mask	SPM	RR	L	04
Set System Mask	SSM	SI		80
Shift Left Double	SLDA	RS	C	8F
Shift Left Single	SLA	RS	C	8B
Shift Left Double Logical	SLDL	RS		8D
Shift Left Single Logical	SLL	RS		89
Shift Right Double	SRDA	RS	C	8E
Shift Right Single	SRA	RS	C	8A
Shift Right Double Logical	SRDL	RS		8C
Shift Right Single Logical	SRL	RS		88
Start I/O	SIO	SI	C	9C
Store	ST	RX		50
Store Character	STC	RX		42
Store Halfword	STH	RX		40
Store Multiple	STM	RS		90
Subtract	SR	RR	C	1B
Subtract	S	RX	C	5B
Subtract Halfword	SH	RX	C	4B
Subtract Logical	SLR	RR	C	1F
Subtract Logical	SL	RX	C	5F
Supervisor Call	SVC	RR		0A
Test and Set	TS	SI	C	93
Test Channel	TCH	SI	C	9F
Test I/O	TIO	SI	C	9D
Test Under Mask	TM	SI	C	91
Translate	TR	SS		DC
Translate and Test	TRT	SS	C	DD
Unpack	UNPK	SS		F3

Floating-Point Feature Instructions

NAME	MNEMONIC	TYPE	CODE
Add Normalized (Long)	ADR	RR F,C	2A
Add Normalized (Long)	AD	RX F,C	6A
Add Normalized (Short)	AER	RR F,C	3A
Add Normalized (Short)	AE	RX F,C	7A
Add Unnormalized (Long)	AWR	RR F,C	2E
Add Unnormalized (Long)	AW	RX F,C	6E
Add Unnormalized (Short)	AUR	RR F,C	3E
Add Unnormalized (Short)	AU	RX F,C	7E
Compare (Long)	CDR	RR F,C	29
Compare (Long)	CD	RX F,C	69
Compare (Short)	CER	RR F,C	39
Compare (Short)	CE	RX F,C	79
Divide (Long)	DDR	RR F	2D
Divide (Long)	DD	RX F	6D
Divide (Short)	DER	RR F	3D
Divide (Short)	DE	RX F	7D
Halve Long	HDR	RR F	24
Halve (Short)	HER	RR F	34
Load and Test (Long)	LTDR	RR F,C	22
Load and Test (Short)	LTER	RR F,C	32
Load Complement (Long)	LCDR	RR F,C	23
Load Complement (Short)	LCER	RR F,C	33
Load (Long)	LDR	RR F	28
Load (Long)	LD	RX F	68
Load Negative (Long)	LNDR	RR F,C	21
Load Negative (Short)	LNER	RR F,C	31
Load Positive (Long)	LPDR	RR F,C	20
Load Positive (Short)	LPER	RR F,C	30
Load (Short)	LER	RR F	38
Load (Short)	LE	RX F	78
Multiply (Long)	MDR	RR F	2C
Multiply (Long)	MD	RX F	6C
Multiply (Short)	MER	RR F	3C
Multiply (Short)	ME	RX F	7C
Store (Long)	STD	RX F	60
Store (Short)	STE	RX F	70
Subtract Normalized (Long)	SDR	RR F,C	2B
Subtract Normalized (Long)	SD	RX F,C	6B
Subtract Normalized (Short)	SER	RR F,C	3B
Subtract Normalized (Short)	SE	RX F,C	7B
Subtract Unnormalized (Long)	SWR	RR F,C	2F
Subtract Unnormalized (Long)	SW	RX F,C	6F
Subtract Unnormalized (Short)	SUR	RR F,C	3F
Subtract Unnormalized (Short)	SU	RX F,C	7F

Decimal Feature Instructions

NAME	MNEMONIC	TYPE	CODE
Add Decimal	AP	SS T,C	FA
Compare Decimal	CP	SS T,C	F9
Divide Decimal	DP	SS T	FD
Edit	ED	SS T,C	DE
Edit and Mark	EDMK	SS T,C	DF
Multiply Decimal	MP	SS T	FC
Subtract Decimal	SP	SS T,C	FB
Zero and Add	ZAP	SS T,C	F8

Commercial Instruction Set

The commercial instruction set includes the instructions of both the standard instruction set and the decimal feature.

Protection Feature Instructions

NAME	MNEMONIC	TYPE	CODE
Insert Storage Key	ISK	RR Z	09
Set Storage Key	SSK	RR Z	08

Scientific Instruction Set

The scientific instruction set includes the instructions of both the standard instruction set and the floating-point feature.

Universal Instruction Set

When the instructions associated with storage protection are added to the commercial and scientific features, a universal instruction set is obtained.

Direct Control Feature Instructions

NAME	MNEMONIC	TYPE	CODE
Read Direct	RDD	SI Y	85
Write Direct	WRD	SI Y	84

List of Instructions by Op Code

CODE	MNEMONIC	PAGE
04	SPM	73
05	BALR	66
06	BCTR	66
07	BCR	65
08	SSK	74
09	ISK	74
0A	SVC	74
10	LPR	27
11	LNR	27
12	LTR	26
13	LCR	27
14	NR	55
15	CLR	54
16	OR	55
17	XR	55
18	LR	26
19	CR	30
1A	AR	28
1B	SR	29
1C	MR	30
1D	DR	31
1E	ALR	28
1F	SLR	29
20	LPDR	44
21	LNDR	45
22	LTDR	44
23	LCDR	44
24	HDR	48
28	LDR	44
29	CDR	47
2A	ADR	45
2B	SDR	46
2C	MDR	48
2D	DDR	49
2E	AWR	46
2F	SWR	47
30	LPER	44
31	LNER	45
32	LTER	44
33	LCER	44
34	HER	48
38	LER	44
39	CER	47
3A	AER	45
3B	SER	46
3C	MER	48
3D	DER	49
3E	AUR	46
3F	SUR	47
40	STH	32
41	LA	56
42	STC	56
43	IC	56
44	EX	67
45	BAL	66
46	BCT	66
47	BC	65
48	LH	26
49	CH	30
4A	AH	28
4B	SH	29
4C	MH	30
4E	CVD	32
4F	CVB	31
50	ST	32
54	N	55
55	CL	54
56	O	55
57	X	55
58	L	26
59	C	30
5A	A	28
5B	S	29
5C	M	30
5D	D	31
5E	AL	28
5F	SL	29
60	STD	50
68	LD	44
69	CD	47
6A	AD	45
6B	SD	46
6C	MD	48
6D	DD	49
6E	AW	46
6F	SW	47
70	STE	50
78	LE	44
79	CE	47
7A	AE	45
7B	SE	46
7C	ME	48
7D	DE	49
7E	AU	46
7F	SU	47
80	SSM	74
82	LPSW	73
83		76
84	WRD	75
85	RDD	75
86	BXH	66
87	BXLE	66.1
88	SRL	60
89	SLL	60
8A	SRA	33
8B	SLA	32
8C	SRDL	61
8D	SLDL	60
8E	SRDA	34
8F	SLDA	33
90	STM	32
91	TM	56
92	MVI	53
93	TS	74
94	NI	55
95	CLI	54
96	OI	55
97	XI	55
98	LM	27
9C	SIO	94
9D	TIO	95
9E	HIO	96
9F	TCH	98
D1	MVN	54
D2	MVC	53
D3	MVZ	54
D4	NC	55
D5	CLC	54
D6	OC	55
D7	XC	55
DC	TR	57
DD	TRT	57
DE	ED	57
DF	EDMK	60
F1	MVO	40
F2	PACK	39
F3	UNPK	39
F8	ZAP	37
F9	CP	38
FA	AP	37
FB	SP	37
FC	MP	38
FD	DP	38

Index

Where more than one page-reference is given, major references appear first and in italic type.

IBM System/360 Principles of Operation (S360-01) Printed in U.S.A. GA22-6821-8

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

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

READER'S COMMENT FORM

IBM System/360 Principles of Operation GA22-6821-8

Your comments about this publication may be helpful to us. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications; this only delays the response. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes ☐

No ☐

Name _____

Job Title _____

Address _____

_____ Zip _____

WTC users must add postage.

GA22-6821-8

YOUR COMMENTS, PLEASE . . . . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your comments will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

CUT ALONG THIS LINE

fold fold

FIRST CLASS
PERMIT NO. 419
POUGHKEEPSIE, N.Y.

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

POSTAGE WILL BE PAID BY . . . . . .

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N.Y. 12602

ATTENTION: CUSTOMER MANUALS, DEPT. B98

fold fold

IBM System/360 Principles of Operation (S360-01) Printed in U.S.A. GA22-6821-8

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

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

READER'S COMMENT FORM

IBM System/360 Principles of Operation GA22-6821-8

Your comments about this publication may be helpful to us. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications; this only delays the response. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes ☐

No ☐

Name _____

Job Title _____

Address _____

_____ Zip _____

WTC users must add postage.

GA22-6821-8

YOUR COMMENTS, PLEASE . . . . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your comments will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 419
POUGHKEEPSIE, N.Y.

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

POSTAGE WILL BE PAID BY . . . . . .

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N.Y. 12602

ATTENTION: CUSTOMER MANUALS, DEPT. B98

fold fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

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

CUT ALONG THIS LINE

IBM System/360 Principles of Operation (S360-01) Printed in U.S.A. GA22-6821-8

READER'S COMMENT FORM

IBM System/360 Principles of Operation GA22-6821-8

Your comments about this publication may be helpful to us. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications; this only delays the response. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes ☐

No ☐

Name _____

Job Title _____

Address _____

_____ Zip _____

WTC users must add postage.

YOUR COMMENTS, PLEASE . . . . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your comments will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 419
POUGHKEEPSIE, N.Y.

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

POSTAGE WILL BE PAID BY . . . . . .

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N.Y. 12602

ATTENTION: CUSTOMER MANUALS, DEPT. B98

fold fold

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

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

CUT ALONG THIS LINE

IBM System/360 Principles of Operation (S360-01) Printed in U.S.A. GA22-6821-8