GA22-7070-1
File No. 4300-01

Systems

IBM 4300 Processors Principles of Operation for ECPS:VSE Mode

IBM

GA22-7070-1
File No. 4300-01

Systems

IBM 4300 Processors Principles of Operation for ECPS:VSE Mode

Second Edition (September 1980)
This major revision obsoletes GA22-7070-0. The document has been revised extensively for clarification and to conform with the wording for common functions in the most recent edition of *IBM System/370 Principles of Operation,* GA22-7000-6. Some material has been rearranged within a chapter, and other material has been moved from one chapter to another.

Changes are identified by a vertical bar in the left margin, except where existing material has been merely rearranged.

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

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Product Publications, Department B98, PO Box 390, Poughkeepsie, NY, U.S.A. 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

This publication provides, for reference purposes, a detailed definition of the architecture of IBM 4300 Processors when operating in the Extended Control Program Support: Virtual Storage Extended (ECPS:VSE) mode.

The publication describes each function of the architecture to the level of detail that must be understood in order to prepare an assembler-language program that relies on that function. It does not, however, describe the notation and conventions that must be employed in preparing such a program.

The information in this publication is provided principally for use by assembler-language programmers, although anyone concerned with the functional details of the IBM 4300 Processors will find it useful.

This publication is written as a reference document and should not be considered an introduction or a textbook.

All facilities discussed in this publication are not necessarily available on every processor. Furthermore, in some instances the definitions have been structured to allow for some degree of extensibility, and therefore certain capabilities may be described or implied that are not offered on any model. An example of such capabilities is the provision for the number of channel-mask bits in the control register. The allowance for this type of extensibility should not be construed as implying any intention by IBM to provide such capabilities. For information about the characteristics and availability of features on a specific processor, use the functional characteristics manual for that processor. The availability of features on processors is summarized in the *IBM 4300 Processors Summary and Input/Output & Data Communications Equipment Configurator,* GA33-1523.

This publication applies only to the ECPS:VSE mode of operation. The *IBM System/370 Principles of Operation,* GA22-7000, should be consulted regarding the functions of the architecture which applies when the processor operates in the System/370 mode.

Size Notation

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

Symbol	Value
K (kilo)	$1{,}024 = 2^{10}$
M (mega)	$1{,}048{,}576 = 2^{20}$

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

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

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

This page left blank intentionally.

Contents

Chapter 1. Introduction

Contents

The IBM 4300 Processors

The IBM 4300 Processors are small and moderately sized processors that have evolved from System/370. They may be used in one of two architectural modes of operation. When operating in the Extended Control Program Support: Virtual Storage Extended (ECPS:VSE) mode, a processor provides new facilities that are designed specifically to enhance the DOS/VSE control program. To run control programs such as VM/370 and OS/VS1, which do not use these facilities, a processor is placed in the System/370 mode. This publication describes the architecture of the 4300 Processors when operating in the ECPS:VSE mode.

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

The ECPS:VSE mode includes a new storage-control facility, called one-level addressing, for creating a single virtual storage of up to 16,777,216 bytes, which both the CPU and the channels address directly using one uniform set of virtual addresses. Mapping the virtual storage onto the real storage is performed internal to the machine.

The one-level-addressing facility provides new instructions and interruptions which the control program uses to determine which parts of virtual storage currently are mapped onto real storage and thereby are made addressable. These instructions and interruptions, and the associated internal address-mapping functions, take the place of dynamic address translation (DAT) and channel indirect data addressing in System/370.

The ECPS:VSE mode also includes a new status-saving function, called machine save, which preserves the entire CPU state and the first 2,048 (2K) bytes of storage. The operator uses machine save in preparation for a complete storage dump. Machine save replaces the store-status function of System/370, which necessarily alters some of the storage to be dumped.

If multiple virtual storages are not required, the ECPS:VSE mode affords the following advantages when compared to System/370:

- Simpler storage-mapping function, with more of the function performed automatically by the machine
- Improved control-program performance, because the control program need not translate the virtual addresses of channel programs

Programming of the machine has been simplified, relative to System/370, by omitting the following functions:

- Multiprocessing and associated instructions
- Machine-check logout and full channel logout

These model-dependent logouts are replaced by internal facilities for diagnosing machine malfunctions. This removes model-dependent error-handling procedures from the control program and improves serviceability.

Compatibility

Compatibility Among 4300 Processors

Although models of the 4300 Processors differ in implementation and physical capabilities, logically they are upward and downward compatible. Compatibility provides for simplicity in education, availability of system backup, and ease in system growth. Specifically, any program will give identical results on any model, provided that it:

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

Compatibility Between 4300 Processors and System/370

Control-Program Compatibility

If the preceding compatibility restrictions are observed, a program written for the 4300 Processors or System/370 will run on the other system. However, because of the compatibility restrictions, control programs cannot be transferred between these systems if they take advantage of facilities that are available on one system but not the other. In particular, the 4300 Processors do not offer the System/370 dynamic-address-translation facility in the ECPS:VSE mode and, hence, cannot execute programs which rely on this particular facility.

To provide full control-program compatibility between System/370 and the 4300 Processors, the 4300 Processors offer an alternate microprogram that causes the machine to assume the characteristics of a System/370 model. When the machine is in this mode, the operation of the machine is as described in the *IBM System/370 Principles of Operation,* GA22-7000.

Problem-State Compatibility

A high degree of compatibility exists at the problem-state level between 4300 Processors operating in the ECPS:VSE mode and System/370. Because the majority of a user's applications are written for the problem state, this problem-state compatibility is useful in many installations.

A program written to run in the problem state on 4300 Processors or System/370 will run on the other system, provided that it:

1. Observes the limitations described in the section "Compatibility Among 4300 Processors."
2. Is not dependent on results defined in this publication or in the *IBM System/370 Principles of Operation,* as appropriate, to be unpredictable or model-dependent (an extension of the fourth rule in the section "Compatibility Among 4300 Processors").
3. Is not dependent on control-program facilities which are unavailable on the system.

To allow the problem programmer to guard against the effects of facilities that are available on System/370 but not on 4300 Processors, this publication in several places describes the results of such effects. For example, when a program is written which shares storage in a multiprogramming environment on a single-CPU configuration, precautions should be taken to allow such a program to run correctly on a multiple-CPU (multiprocessing) configuration.

Specifically, COMPARE AND SWAP, COMPARE DOUBLE AND SWAP, and TEST AND SET are the only instructions which should be used to create interlocks between concurrent programs. These are the only instructions that do not, between fetching and storing of the storage operand, permit another CPU to access the operand location. The instructions AND (NI or NC), EXCLUSIVE OR (XI or XC), and OR (OI or OC) should not be used for such interlocks.

The program may also have to take into account that serialization of CPU operations, which is performed by all interruptions and by the execution of certain instructions, affects the sequence of events as observed by other CPUs in a multiprocessing configuration as well as by channels. (See the section "Serialization" in Chapter 5, "Program Execution.")

Programming Note
This publication assigns meanings to various operation codes, to bit positions in instructions, channel-command words, registers, and table entries, and to fixed locations in the low 512 bytes of storage (addresses 0-511). Other operation codes, bit positions, and low-storage locations are specifically noted as being available for programming use. The remaining ones are unassigned and reserved for future assignment to new facilities and other extensions of the architecture.

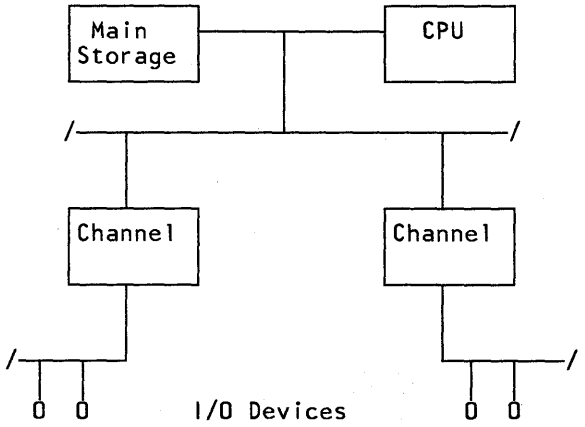
To ensure that existing programs run if and when such new facilities are installed, programs should not depend on an indication of an exception as a result of invalid values that are currently defined as being checked. If a value must be placed in unassigned positions that are not checked, the program should enter zeros. When the machine provides a code or field, the program should take into account that new codes and bits may be assigned in the future. The program should not use unassigned low-storage locations for keeping information since these locations may be assigned in the future in such a way that the machine causes this location to be changed.

Chapter 2. Organization

Contents

Logically, IBM 4300 Processors consist of main storage, a central processing unit (CPU), operator facilities, and channels. The channels allow input/output (I/O) devices to be attached, usually through control units. (See the figure "Logical Structure.")

Specific processors may differ in their internal characteristics, the number and types of channels, the size of main storage, and the representation of the operator facilities. The differences in internal characteristics are apparent to the observer only as differences in machine performance.



Logical Structure

Main Storage

The 4300 Processors provide fast-access main storage and storage-control functions for high-speed processing of data by the CPU and channels. The storage-control functions permit main storage to be controlled at two levels: real storage and virtual storage.

Real storage is the storage where data and instructions actually reside at the time they are accessed by the CPU and channels, but neither CPU programs nor channel programs can address real storage directly. The size of real storage depends on the model.

Virtual storage allows both CPU programs and channel programs to address an apparent main storage of up to 16,777,216 (16M) bytes. Virtual storage may be larger than the underlying real storage. If the virtual storage is larger than the real storage, a supervisory control program using the storage-control functions of the machine is required for controlling which parts of virtual storage are currently mapped onto real storage. This control is dynamic and transparent to the other programs except for the time delay.

Central Processing Unit

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

The physical makeup of the CPU in the various models of the machine may be different, but the logical function remains the same. The result of executing a valid instruction is the same for each model.

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

Instructions which the CPU executes fall into five classes: general, decimal, floating-point, control, and input/output instructions. The general instructions are used in performing fixed-point arithmetic operations and logical, branching, and other nonarithmetic operations. The decimal instructions operate on data in the decimal format, and the floating-point instructions on data in the floating-point format. The control instructions and the input/output instructions are privileged instructions that can be executed only when the CPU is in the supervisor state.

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

The CPU provides registers which are available to programs but do not have addressable representations in main storage. They include the current program-status word (PSW), the general registers, the floating-point registers, the control registers, and the registers for the time-of-day (TOD) clock, the clock comparator, and the CPU timer. The instruction operation code determines which type of register is to be used in an operation. See the figure "General, Floating-Point, and Control Registers" later in this chapter for the format of those registers.

Program-Status Word

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

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

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

General Registers

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

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

In addition to their use as accumulators in general arithmetic and logical operations, 15 of the 16 general registers are also used as base-address and index registers in address generation. In these cases, the registers are designated by a four-bit B field or X field in an instruction. A value of zero in the B or X field specifies that no base or index is

to be applied, and, thus, general register 0 cannot be designated as containing a base address or index.

Floating-Point Registers

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

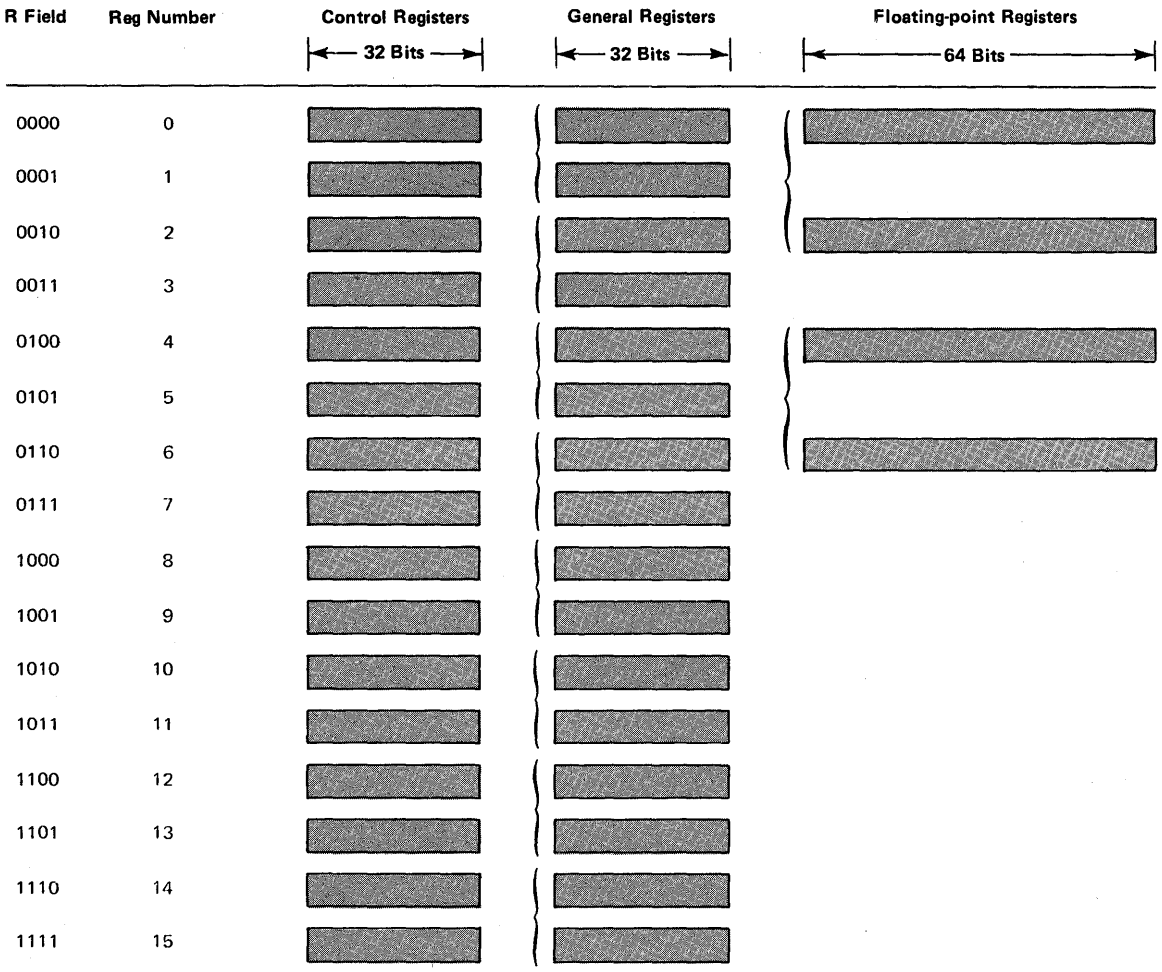
Control Registers

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

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

Input and Output

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



Note: The braces indicate that the two registers may be coupled as a double-register pair, designated by specifying the lower-numbered register in the R field. For example, the general-register pair 0 and 1 is designated in the R field by the number 0.

General, Floating-Point, and Control Registers

Channels

A 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 connects with the CPU, with main storage, and with control units.

A channel may be an independent unit, complete with the necessary logical and internal-storage capabilities, or it may time-share CPU facilities and be physically integrated with the CPU. In either case, the functions performed by a channel are identical. The maximum data-transfer rate may differ, however, depending on the implementation.

There are three types of channels: byte-multiplexer, block-multiplexer, and selector channels.

Input/Output Devices and Control Units

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

Operator Facilities

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

The main functions provided by the operator facilities are system reset, clearing, initial program loading, start, stop, alter, and display.

Chapter 3. Storage

Contents

This chapter discusses the representation of information in storage, how information is addressed, and the one-level-addressing facility for controlling virtual and real storage. The chapter also contains a list of permanently assigned storage locations.

The term "main storage" is used generically to describe both virtual and real storage, in order to distinguish this fast-access storage from auxiliary storage, such as direct-access storage devices. Physically, main storage may be composed of a high-capacity fast storage medium and a smaller but faster buffer storage, sometimes called a cache. The effects, except on performance, of the physical construction and the use of distinct storage media are not observable by the program. Because, in this publication, most references to main storage apply to virtual storage, the abbreviated term "storage" is generally used in place of "virtual storage" when the meaning is clear.

All addresses of storage locations are virtual addresses, because they always refer to virtual storage. Hence, when applied to main storage, address means virtual address in this publication.

Storage Addressing

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

Each byte location in storage is identified by a unique nonnegative integer, which is the address of that byte location or, simply, the byte address. Adjacent byte locations have consecutive addresses, starting with 0 on the left and proceeding in a left-to-right sequence. Addresses are 24-bit unsigned binary integers, which provide 16,777,216 (2^{24} or 16M) byte addresses.

The CPU performs address generation when it forms an operand or instruction address. It also performs address generation when it increments an address to access successive bytes of a field. Similarly, the channel generates an address when it increments an address to fetch a channel-command word (CCW) from a CCW list or to transfer data.

When, during address generation, an address is obtained that exceeds $2^{24} - 1$, the carry out of the high-order bit position of the address is ignored. This handling of an address of excessive size is called *wraparound.*

The effect of wraparound is to make the sequence of addresses appear circular; that is, address 0 appears to follow the maximum byte address, 16,777,215. In 16M-byte storage, information may be located partially in the last and partially in the first locations of storage and is

processed without any special indication of crossing the maximum-address boundary.

Information Formats

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

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

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

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

For purposes of error detection, and in some models for correction, one or more check bits may be transmitted with each byte or with a group of bytes. Such check bits are generated automatically by the machine and cannot be directly controlled by the program. References in this publication to the length of data fields and registers exclude mention of the associated check bits. All storage capacities are expressed in number of bytes.

When the length of an operand field is implied by the operation code of an instruction, the field is said to have a fixed length, which can be one, two, four, or eight bytes.

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

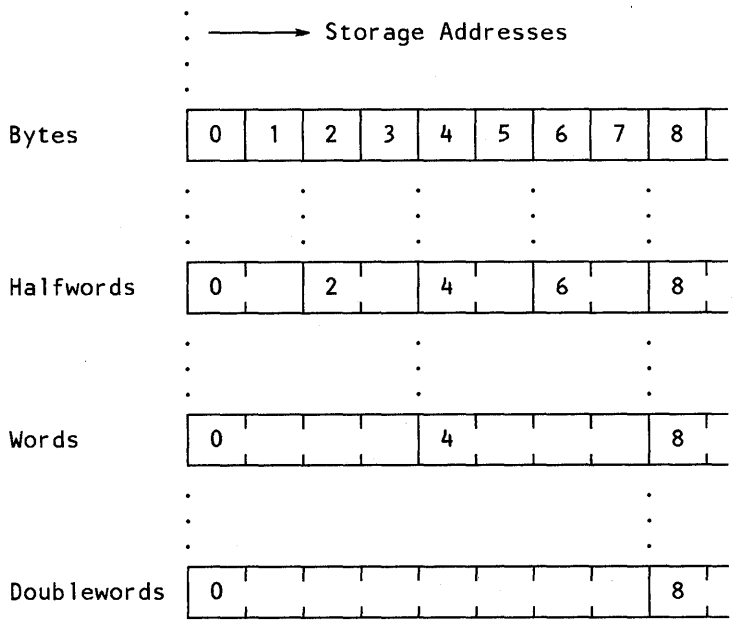
When information is placed in storage, the contents of only those byte locations are replaced that are included in the designated field, even though the width of the physical path to storage may be greater than the length of the field being stored.

Integral Boundaries

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

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

Instructions must appear on two-byte integral boundaries, and channel-command words and the storage operands of certain instructions must appear on other integral boundaries. The storage operands of most instructions do not have boundary-alignment requirements.



Integral Boundaries with Storage Addresses

Programming Note

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

One-Level-Addressing Facility

The one-level-addressing facility may be used by the control program to create virtual storage that is larger than the actual capacity of the underlying real storage. Other programs and channels address this virtual storage directly as if all data and instructions actually resided in virtual storage.

Main storage is volatile; that is, the contents are not preserved when power is off.

Storage Size

The storage size is the number of addressable byte locations provided in virtual storage. A model may allow one or more storage sizes. If more than one storage size is provided, the current storage size is determined by the manual storage-size control during an initial microprogram loading (IML) operation. The storage size cannot be changed by programming.

The storage size is always a multiple of 2,048 (2K) bytes, up to a maximum of 16,777,216 (16M) bytes.

When the storage size exceeds the size of real storage, the parts of virtual storage which are currently not directly accessible may be kept on auxiliary storage, such as direct-access storage devices (DASD). The transfer of the contents of virtual storage to and from auxiliary storage may be controlled by a supervisory control program using I/O instructions in such a way that the remaining CPU programs and channel programs can address any part of virtual storage as if it were all directly accessible.

Storage addresses range from zero to one less than the storage size. If the CPU attempts to refer to a storage location that is not provided or to the corresponding page description (see below), that attempt is indicated by an addressing exception or, in the case of the LOAD FRAME INDEX instruction, by the condition code. If an I/O operation attempts to access a storage location that is not provided, the operation is terminated by an I/O-interruption condition indicating program check.

Normally, the indication that a location is not provided is given only when the information associated with that location is actually required, and not when the operation can be completed without that information.

When the storage size is set to the maximum value of 16M bytes, all storage locations are provided; addressing exceptions or program checks for CCW or data locations cannot occur.

Pages

Virtual storage is divided into pages, each page consisting of 2,048 (2^{11}) consecutive bytes on a 2,048-byte address boundary. Virtual storage has up to 8,192 (2^{13}) pages of storage. The size of virtual storage and, hence, the number of pages provided depend on the model and on the setting of the manual storage-size control, if one is provided.

Storage-control instructions, except for INSERT STORAGE KEY and SET STORAGE KEY, refer to a page by the address of any byte in that page. The low-order 11 bits of an operand address referring to a whole page are ignored. The INSERT STORAGE KEY and SET STORAGE KEY instructions also use a page address, but the low-order four bits of their operand address must be zeros.

Page Frames

Real storage is divided into page frames, each capable of containing the data for one page of virtual storage. The size of real storage and, hence, the number of page frames present in the machine depend on the model. Real storage is not explicitly addressable by CPU programs and channel programs.

A page in virtual storage, to be accessible to CPU programs and channel programs, must be associated with a page frame in real storage. An instruction is provided which assigns to a page a free page frame selected by the machine. This instruction is said to connect the page to its assigned frame. Thereafter, the page frame is referred to by the address of the corresponding page. When any previous contents of the page have been retrieved from external storage and the page is ready for accessing by a CPU program, another instruction is used to make the page addressable.

As the supply of free page frames diminishes, the control program may make a page not addressable and, if any bytes in the page have been changed, write the contents of the page on auxiliary storage. An instruction may then be issued to disconnect the page, thus freeing its frame.

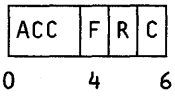
A page frame cannot be assigned to more than one page at a time.

Page Description

Associated with each virtual-storage page which is provided are a seven-bit storage key, three programmable page bits, a page state, and the frame index of the page frame currently assigned to the page, if any. This information, called the page description, is maintained in internal storage.

Storage Key

A storage key is associated with each page that is provided.



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

Access-Control Bits (ACC): The four access-control bits, bits 0-3, are matched with the four-bit access key whenever information is stored, or whenever information is fetched from a location that is protected against fetching.

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

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

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

The entire storage key is set by SET STORAGE KEY and inspected by INSERT STORAGE KEY. The reference and change bits are also set by SET PAGE BITS and inspected by INSERT PAGE BITS. Additionally, the instruction RESET REFERENCE BIT provides a means of inspecting the reference and change bits and of setting the reference bit to zero.

Page Bits

The three programmable page bits associated with each page may be set by the instruction SET PAGE BITS and inspected by INSERT PAGE BITS. The page bits are disregarded by the machine during other operations.

Programming Note

The page bits may be used by the program to assist in managing pages. For example, one of the bits may indicate whether a version of the corresponding page exists on auxiliary storage.

Page States

A page may be in one of three states:

- Disconnected
- Connected
- Addressable

If disconnected, the page does not have a page frame assigned to it. Any attempt by the CPU to access a disconnected page causes a page-access exception or, when a CLEAR PAGE instruction is being executed, a page-state exception. Any attempt by a channel to access a disconnected page, in order to fetch a CCW or to access a data area designated during the execution of a CCW, creates an I/O-interruption condition indicating protection check. However, if a CCW is prefetched, a protection check is not indicated until the CCW is due to be executed.

If connected, the page has a page frame assigned to it. A connected page may be accessed, if the protection mechanism permits:

1. By I/O channels
2. By the CPU as the operand of the CLEAR PAGE instruction

Except when executing CLEAR PAGE, an attempt by the CPU to access a connected page causes a page-access exception.

If addressable, the page has a page frame assigned to it, and the CPU and I/O channels may access the page if the protection mechanism permits.

Although the addressable state implies that the page is also connected, the term "connected state" refers specifically to the state of a page that is neither addressable nor disconnected.

The page state is checked for all storage accesses to locations that are explicitly or implicitly accessed by the CPU or by a channel.

The page state is changed by instructions, which may make the transition from the disconnected to the connected state and from the connected to the addressable state, or vice versa. The instructions are CONNECT PAGE, DECONFIGURE PAGE, DISCONNECT PAGE, MAKE ADDRESSABLE, and MAKE UNADDRESSABLE. Most of these instructions may also be applied to pages which are already in the desired state. An attempt to change directly from disconnected to addressable, or vice versa, causes a page-transition exception. A page-transition exception is also caused by DECONFIGURE PAGE when applied to a disconnected page.

The first page, page 0, containing byte locations 0 to 2047, is always addressable. It cannot be placed in the connected or disconnected state.

The clear-reset function causes a number of consecutive pages, starting with page 0, to be assigned to page frames, cleared, and placed in the addressable state. (See the section "Clear Reset" in Chapter 4, "Control.")

Programming Notes

1. The three page states permit programs to distinguish pages in the disconnected state, which cannot be accessed at all, from pages in the connected state, which are undergoing I/O operations to or from auxiliary page storage, and from pages in the addressable state, which are ready for normal storage access.
2. The storage-control instructions other than CLEAR PAGE operate on page descriptions, not pages. Instructions which operate on page descriptions do not make storage accesses and do not cause page-access or protection exceptions.
3. All channel accesses to storage appear as if they referred to pages and not to the associated page frames. If a page that is being accessed by a channel becomes disconnected and another channel access is attempted, protection check is indicated, and the I/O operation terminates. If the page becomes disconnected and then becomes reconnected before protection check is indicated, protection check may be indicated subsequently, or accesses may continue using the newly assigned page frame.

Frame Index

A unique 16-bit unsigned binary integer is assigned to each page frame existing in the machine. This integer is the frame index of the page frame. The value of the frame index ranges from zero up to, but not including, the existing-frame-capacity count (EFCC) (see the section "Capacity Counts" in this chapter).

When a CONNECT PAGE instruction connects a page to a frame, the frame index of the connected frame is returned by the instruction. The frame index remains associated with that page until the page is disconnected. When the same page is connected again, the new frame index is, in general, different and unpredictable unless there was only one free frame remaining.

The frame index for an already connected or addressable page may be displayed by LOAD FRAME INDEX.

When DECONFIGURE PAGE makes a page frame unavailable, the frame index of that frame will not recur until a clear-reset operation is performed.

The frame index currently associated with a page is part of its page description. A disconnected page has no frame index, and LOAD FRAME INDEX returns no frame-index value for a disconnected page.

Programming Notes

1. The frame index assists the control program in maintaining compact tables of connected or addressable pages. The frame index is not, and should not be interpreted as, the address of a frame in real storage. The algorithm for assigning a frame index to a page frame is implementation-dependent. Programming should not depend on a particular algorithm.
2. The set of frame indexes is dense if the EFCC equals the AFCC, that is, if there are no unavailable page frames. It becomes nondense to the extent that frames are made unavailable by DECONFIGURE PAGE or by maintenance intervention.
3. DECONFIGURE PAGE removes a page frame from contention for connection when a machine check has indicated damage to a page. This can be done only while the frame is connected to a page, because a frame cannot be addressed directly.

Page and Frame Control

Capacity Counts

Four internally maintained counts are defined to assist the program in managing pages and page frames. Each count is a 16-bit unsigned binary integer. The counts are set or updated by the machine. They are displayed by the STORE CAPACITY COUNTS instruction, which stores each count as a 32-bit integer with 16 high-order zero bits.

The page-capacity count (PCC) is the number of virtual-storage pages provided by the machine. The pages have page addresses from 0 to PCC minus one. The value of the PCC is equal to the storage size divided by 2,048; it is set during clear reset according to the current setting of the manual storage-size control, if one is provided.

The existing-frame-capacity count (EFCC) is the number of page frames existing in a particular implementation of the machine. The EFCC reflects the total capacity of real storage. The value of the EFCC is set during clear reset.

The available-frame-capacity count (AFCC) is the number of page frames connected or available for connection to pages. The value of AFCC may be equal to or less than the EFCC. During CPU operation, the AFCC may be decreased by the instruction DECONFIGURE PAGE. The clear-reset function initializes the AFCC to the value of the EFCC less the number of frames that are kept unavailable for connection by maintenance intervention.

The free-frame-capacity count (FFCC) is the number of available page frames that are currently not connected to pages. The value of the FFCC may range from zero to the AFCC minus one. During CPU operation, the value of the FFCC may be changed by the instructions CONNECT PAGE and DISCONNECT PAGE. The clear-reset operation initializes the FFCC to zero or to the value of AFCC minus PCC, depending on whether the AFCC is less than the PCC or not.

Since page 0 must always be addressable, the frame connected to page 0 is considered available but not free. Hence, the minimum value of the AFCC is one, and the maximum value of the FFCC is the AFCC minus one.

Storage-Control Instructions

CONNECT PAGE is used to change a page from the disconnected to the connected state. MAKE ADDRESSABLE changes a page from connected to addressable. MAKE UNADDRESSABLE changes the page state from addressable to connected. DISCONNECT PAGE changes the page state from connected to disconnected. DECONFIGURE PAGE disconnects a connected page and makes the corresponding page frame and its frame index unavailable. LOAD FRAME INDEX tests the page state of a page and displays its frame index, if any. These six instructions do not change or check the storage key of the specified pages.

CLEAR PAGE sets the contents of a page to zero and validates the page.

SET STORAGE KEY replaces the storage key of a page. INSERT STORAGE KEY retrieves the storage key of a page except, in the BC mode, for the reference and change bits. RESET REFERENCE BIT tests the reference and change bits and resets the reference bit to zero.

SET PAGE BITS tests the reference and change bits of a page and then explicitly sets them along with the three programmable page bits of that page. INSERT PAGE BITS retrieves the values of the three page bits, the reference bit, and the change bit of a page.

All storage-control instructions are privileged.

Key-Controlled Protection

Key-controlled protection is provided to protect the contents of storage from destruction or misuse caused by erroneous or unauthorized storing or fetching by the program. It affords protection against improper storing or against both improper storing and fetching, but not against improper fetching alone.

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

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

The protection action is summarized in the figure "Summary of Protection Action."

When the access to storage is initiated by the CPU, and key-controlled protection applies, the PSW key is the access key which is used as the compare value. The PSW key occupies bit positions 8-11 of the current PSW.

When the reference is made by a channel, and key-controlled protection applies, the subchannel key associated with the I/O operation is the access key which is used as the compare value. The subchannel key is specified for an I/O operation in bit positions 0-3 of the channel-address word (CAW); the subchannel key is later placed in bit positions 0-3 of the channel-status word (CSW) that is stored as a result of the I/O operation.

When a CPU access is prohibited because of protection, the operation is suppressed or terminated, and a program interruption for a protection exception takes place. When a channel access is prohibited, protection check is indicated in the CSW stored as a result of the operation.

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

Explanation:

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

Yes — Access is permitted.

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

Summary of Protection Action

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

Key-controlled protection is always active, regardless of whether the CPU is in the problem or supervisor state, and regardless of the type of CPU instruction or channel-command word being executed.

All accesses to storage locations that are explicitly designated by the program and that are used by the CPU to store or fetch information are subject to key-controlled protection.

All storage accesses by a channel to fetch a CCW or to access a data area designated during the execution of a CCW are subject to key-controlled protection. However, if a CCW or output data is prefetched, a protection check is not indicated until the CCW is due to be executed or the data is due to be written.

Key-controlled protection is not applied to accesses that are implicitly made by the CPU or channel for such sequences as:

- Interruptions,
- Updating the interval timer,
- Fetching the CAW during the execution of an I/O instruction,
- Storing the CSW by an I/O instruction or interruption,
- Storing channel identification during the execution of STORE CHANNEL ID,
- Limited channel logout, or
- Initial program loading.

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

Reference Recording

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

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

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

CONNECT PAGE
DECONFIGURE PAGE (reference bit is set to zero)
DISCONNECT PAGE (reference bit is set to zero)
INSERT PAGE BITS
INSERT STORAGE KEY
LOAD FRAME INDEX
MAKE ADDRESSABLE
MAKE UNADDRESSABLE
RESET REFERENCE BIT (reference bit is set to zero)
SET PAGE BITS (reference bit is set to a specified value)
SET STORAGE KEY (reference bit is set to a specified value)

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

Change Recording

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

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

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

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

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

Change recording does not take place for the operands of the following instructions since they directly modify a page description without modifying a storage location:

CONNECT PAGE
DECONFIGURE PAGE (change bit is set to zero)
DISCONNECT PAGE (change bit is set to zero)
MAKE ADDRESSABLE
MAKE UNADDRESSABLE
RESET REFERENCE BIT
SET PAGE BITS (change bit is set to a specified value)
SET STORAGE KEY (change bit is set to a specified value)

Change bits are not necessarily restored on CPU retry (see the section "CPU Retry" in Chapter 11, "Machine-Check Handling"). See the section "Exceptions to Nullification and Suppression" in Chapter 5, "Program Execution," for a description of the handling of the change bit in certain unusual situations.

Assigned Storage Locations

Assigned locations in storage have different uses when the CPU is in the operating state or in the load state. This section is summarized in the figure "Assigned Storage Locations."

Programming Note

In the BC mode, there is no implicit storing in locations 128 and above if all of the following conditions are met:

1. The manual check control is set to stop.
2. The MONITOR CALL and STORE CHANNEL ID instructions are not issued.
3. The page-capacity count is equal to or less than the available-frame-capacity count and all pages are addressable.

Storage While CPU is in Operating State

This section shows the format and extent of the assigned storage locations while the CPU is in the operating state. Unless specifically noted, the usage applies to both the EC and BC modes.

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

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

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

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

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

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

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

64-71 *CSW:* The channel-status word (CSW) is stored at locations 64-71 during an I/O interruption. Part or all of it may be stored during the execution of START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, or HALT DEVICE, in which case condition code 1 is set.

72-75 *CAW:* The channel-address word (CAW) is fetched from locations 72-75 during the execution of START I/O and START I/O FAST RELEASE.

80-83 *Interval Timer:* Locations 80-83 contain the interval timer. The interval timer is updated whenever the CPU is in the operating state and the manual interval-timer control is set to enable.

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

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

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

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

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

132-135 *External-Interruption Code:* During an external interruption in the EC mode, the interruption code is stored at locations 134-135, and zeros are stored at locations 132-133.

136-139 *Supervisor-Call-Interruption Identification:* During a supervisor-call

interruption in the EC mode, the instruction-length code is stored in bit positions 5 and 6 of location 137, and the interruption code is stored at locations 138-139. Zeros are stored at location 136 and in the remaining bit positions of 137.

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

144-147 *Access-Exception Address:* During a program interruption due to a page-access exception, the address for which the exception is being indicated is stored at locations 145-147, and zeros are stored at location 144.

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

150-151 *PER Code:* During a program interruption due to a program event, the program-event-recording (PER) code is stored in bit positions 0-3 of location 150, and zeros are stored in bit positions 4-7 and at location 151. This field can be stored only when the instruction causing the PER condition was executed under the control of a PSW specifying the EC mode.

152-155 *PER Address:* During a program interruption due to a program event, the program-event-recording (PER) address is stored at locations 153-155, and zeros are stored at location 152. This field can be stored only when the instruction causing the PER condition was executed under the control of a PSW specifying the EC mode.

156-159 *Monitor Code:* During a program interruption due to a monitor event, the monitor code is stored at locations 157-159, and zeros are stored at location 156.

168-171 *Channel ID:* The four-byte channel-identification information is stored at locations 168-171 during the execution of STORE CHANNEL ID.

176-179 *Limited Channel Logout:* The limited-channel-logout information is stored at locations 176-179. This field may be stored only when the CSW or a portion of the CSW is stored.

185-187 *I/O Address:* During an I/O interruption in the EC mode, the two-byte I/O address is stored at locations 186-187, and zeros are stored at location 185.

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

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

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

248-251 *Failing-Storage Address:* During a machine-check interruption, a failing-storage address, if any, is stored at locations 249-251, and zeros are stored at location 248.

352-383 *Floating-Point-Register Save Area:* During a machine-check interruption, the contents of the floating-point registers are stored at locations 352-383.

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

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

Storage While CPU is in Load State

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

8-15 *IPL CCW1:* Bytes 8-15 read during the IPL initial read operation are stored at locations 8-15. The contents of these locations are ordinarily used as

the next CCW in an IPL CCW chain after completion of the IPL initial-read operation.

16-23 *IPL CCW2:* Bytes 16-23 read during the IPL initial read operation are stored at locations 16-23. The contents of these locations may be used as another CCW in the IPL CCW chain to follow IPL CCW1.

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

Assigned Storage Locations (Part 1 of 3)

Hex	Dec					
80	128					
84	132	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			External-Interruption Code	
88	136	0 0 0 0 0 0 0 0 0 0 0 0 0 0	ILC	0	Superv-Call-Interruption Code	
8C	140	0 0 0 0 0 0 0 0 0 0 0 0 0 0	ILC	0	Program-Interruption Code	
90	144	0 0 0 0 0 0 0 0	Access-Exception Address			
94	148	0 0 0 0 0 0 0 0	Monitor Class #	PER Cde	0 0 0 0 0 0 0 0 0 0 0 0	
98	152	0 0 0 0 0 0 0 0	PER Address			
9C	156	0 0 0 0 0 0 0 0	Monitor Code			
A0	160					
A4	164					
A8	168	Channel ID				
AC	172					
B0	176	Limited Channel Logout				
B4	180					
B8	184		0 0 0 0 0 0 0 0	I/O Address		
BC	188					
C0	192					
C4	196					
C8	200					
CC	204					
D0	208					
D4	212					
D8	216	CPU-Timer Save Area				
DC	220					
E0	224	Clock-Comparator Save Area				
E4	228					
E8	232	Machine-Check Interruption Code				
EC	236					
F0	240					
F4	244					
F8	248	0 0 0 0 0 0 0 0	Failing-Storage Address			

Assigned Storage Locations (Part 2 of 3)

Hex	Dec	
FC	252	
100	256	
104	260	
108	264	
	/	/
154	340	
158	344	
15C	348	
160	352	Floating-Point-Register Save Area
164	356	
168	360	
16C	364	
170	368	
174	372	
178	376	
17C	380	
180	384	General-Register Save Area
184	388	
188	392	
18C	396	
	/	/
1B4	436	
1B8	440	
1BC	444	
1C0	448	Control-Register Save Area
1C4	452	
1C8	456	
1CC	460	
	/	/
1F4	500	
1F8	504	
1FC	508	

Assigned Storage Locations (Part 3 of 3)

Chapter 4. Control

Contents

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

Stopped, Operating, Load, and Check-Stop States

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

A change between these four CPU states cannot be effected by the program. The states are not controlled or identified by bits in the PSW. The stopped, load, and check-stop states are indicated to the operator by means of the manual indicator, load indicator, and check-stop indicator respectively. These three indicators are off when the CPU is in the operating state.

The CPU timer is updated when the CPU is in the operating state or the load state. The time-of-day clock is updated whenever power is on. The interval timer is updated only when the CPU is in the operating state.

Stopped State

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

- The stop key is activated while the CPU is in the operating state.
- The CPU has finished the execution of a unit of operation initiated by performing the start function with the rate control set to instruction step.

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

Before entering the stopped state, all pending allowed interruptions are taken while the CPU is still in the operating state. They cause the old PSW to be stored and the new PSW to be fetched before the stopped state is entered. When the CPU is in the stopped state, interruption conditions remain pending.

The CPU is also placed in the stopped state:

- When a reset is completed, except when the reset operation is performed as part of initial program loading, and
- When an address comparison indicates equality and stopping on the match is specified

The execution of resets is described in the section "Resets" in this chapter, and address comparison is described in the section "Address-Compare Controls" in Chapter 13, "Operator Facilities."

Operating State

The state of the CPU is changed from stopped to operating when the start function is performed or when a restart interruption occurs. However, the effect of performing the start function is unpredictable when the stopped state was entered by means of a reset.

The start function is performed on the CPU in the stopped state when the start key is activated.

When the wait-state bit is one and the rate control is set to instruction step, the start function causes no instruction to be executed, but all pending allowed interruptions are taken before the CPU returns to the stopped state.

Load State

The CPU enters the load state when the load-normal or load-clear key is activated (see the section "Initial Program Loading" in this chapter). When the initial-program-loading operation is completed successfully, the CPU state changes from load to operating, provided the rate control is set to process; if the rate control is set to instruction step, the CPU state changes from load to stopped.

Check-Stop State

The check-stop state, which the CPU enters on certain types of machine malfunction, is described in Chapter 11, "Machine-Check Handling."

Programming Notes

1. Except for the relationship between execution time and real time, the execution of a program is not affected by stopping the CPU.
2. When, because of a machine malfunction, the CPU is unable to end the execution of an instruction, the stop function is ineffective, and a reset function has to be invoked instead. A similar situation occurs when an unending string of interruptions results from a PSW with a PSW-format error of the type that is recognized early, or from a persistent interruption condition, such as one due to the CPU timer.
3. Input/output operations continue to completion after the CPU enters the stopped state. The interruption conditions due to completion of I/O operations remain pending when the CPU is in the stopped state.

Program-Status Word

The current program-status word (PSW) contains information required for the execution of the currently active program. The PSW is 64 bits in length and includes the instruction address, condition code, and other control fields. In general, the PSW is used to control instruction sequencing and to hold and indicate much of the status of the CPU in relation to the program currently being executed. Additional control and status information is contained in control registers and permanently assigned storage locations.

Control is switched during an interruption of the CPU by storing the current PSW, so as to preserve the status of the CPU, and then loading a new PSW.

The status of the CPU can be changed by loading a new PSW or part of a PSW.

The instruction LOAD PSW introduces a new PSW. The instruction address is updated by sequential instruction execution and replaced by successful branches. Other instructions are provided which operate on a portion of the PSW. The figure "Operations on System Mask, PSW Key, and Program Mask" summarizes these instructions.

A new or modified PSW becomes active (that is, the information introduced into the current PSW assumes control over the CPU) when the interruption or the execution of an instruction that changes the PSW is completed. The interruption for program-event recording associated with an instruction that changes the PSW occurs under control of the PER mask that is effective at the beginning of the operation.

Bits 0-7 of the PSW are collectively referred to as the system mask.

EC and BC Modes

Two control modes are provided for the formatting and use of control and status information: the extended-control (EC) mode and the basic-control (BC) mode. Certain functions available in the EC mode are not available, or are available in a restricted form, in the BC mode. The mode currently in effect is specified by PSW bit 12. Bit 12 is one for the EC mode and zero for the BC mode.

Program-event recording can be specified only in the EC mode, because the PSW bit to turn this function on is not available in the BC mode.

In the EC mode, I/O interruptions can be controlled individually for up to 32 channels using the correspondingly numbered 32 mask bits in control register 2; there is also a summary-mask bit for I/O interruptions, bit 6 of the PSW. The BC mode operates in this manner only for channels 6 and up: these channels are individually controlled by the corresponding bits of control register 2, as well as the summary-mask bit, bit 6 of the PSW; channels 0-5 are controlled separately by bits 0-5 of the PSW and are not subject to the summary mask or to mask bits in control register 2.

When interruptions occur while in the EC mode, the interruption code and instruction-length code are stored at various permanently assigned storage locations according to the class of interruptions. In the BC mode, the interruption code and

Instruction	System Mask (PSW bits 0-7)		PSW Key (PSW bits 8-11)		Condition Code and Program Mask*	
	Saved	Set	Saved	Set	Saved	Set
BRANCH AND LINK	No	No	No	No	Yes	No
INSERT PSW KEY	No	No	Yes	No	No	No
SET PROGRAM MASK	No	No	No	No	No	Yes
SET PSW KEY FROM ADDRESS	No	No	No	Yes	No	No
SET SYSTEM MASK	No	Yes	No	No	No	No
STORE THEN AND SYSTEM MASK	Yes	ANDs	No	No	No	No
STORE THEN OR SYSTEM MASK	Yes	ORs	No	No	No	No

Explanation:

* PSW bits 18-23 in EC mode; PSW bits 34-40 in BC mode.

ANDs The logical AND of the immediate field in the instruction and the current system mask replaces the current system mask.

ORs The logical OR of the immediate field in the instruction and the current system mask replaces the current system mask.

Operations on System Mask, PSW Key, and Program Mask

instruction-length code for all except machine-check interruptions are placed in the PSW.

The program-mask and condition-code fields in the PSW are allocated to different bit positions in the two control modes. The instruction INSERT STORAGE KEY provides the reference and change bits when in the EC mode but produces zeros in the corresponding bit positions when in the BC mode.

Programming Notes

1. The BC mode provides a PSW format that is compatible with the PSW of System/360.
2. The choice between EC and BC modes affects only those aspects of operation that are specifically defined to be different for the two modes. It does not affect the operation of any functions that are not associated with the control bits in the PSW provided only in the EC mode, and it does not affect the validity of any instructions. The instructions SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK perform the specified function on the leftmost byte of the PSW regardless of the mode specified by the current PSW. On the other hand, the instruction SET PROGRAM MASK introduces a new program mask regardless of the PSW bit positions occupied by the mask.

Program-Status-Word Format in EC Mode

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

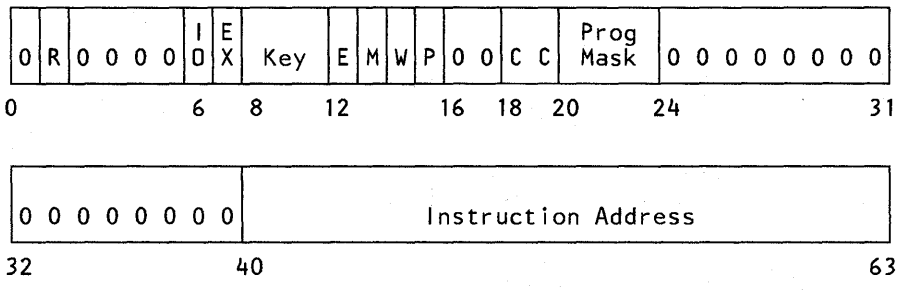
PER Mask (R): Bit 1 controls whether the CPU is enabled for interruptions associated with program-event recording (PER). When the bit is zero, no PER event can cause an interruption. When the bit is one, interruptions are permitted subject to the PER-event-mask bits in control register 9.

I/O Mask (IO): Bit 6 controls whether the CPU is enabled for I/O interruptions. When the bit is zero, an I/O interruption cannot occur. When the bit is one, I/O interruptions are subject to the channel-mask bits in control register 2; when a channel-mask bit is zero, the associated channel cannot cause an I/O interruption; when the channel-mask bit is one, an interruption condition at the channel can cause an interruption.

External Mask (EX): Bit 7 controls whether the CPU is enabled for interruption by conditions included in the external class. When the bit is zero, an external interruption cannot occur. When the bit is one, an external interruption is subject to the corresponding external subclass-mask bits in control register 0; when the subclass-mask bit is zero, conditions associated with the subclass cannot cause an interruption; when the subclass-mask bit is one, an interruption in that subclass can occur.

PSW Key: Bits 8-11 form the access key for storage references by the CPU. This PSW key is matched with a storage key whenever information is stored, or whenever information is fetched from a location that is protected against fetching.

EC Mode (E): Bit 12, which controls the format of the PSW and the mode of operation of the CPU, is one when the CPU is in the extended-control (EC) mode.



PSW Format in EC Mode

Machine-Check Mask (M): Bit 13 controls whether the CPU is enabled for interruption by machine-check conditions. When the bit is zero, a machine-check interruption cannot occur. When the bit is one, machine-check interruptions due to system damage and instruction-processing damage are permitted, but interruptions due to other machine-check-subclass conditions are subject to the subclass-mask bits in control register 14.

Wait State (W): When bit 14 is one, the CPU is waiting; that is, no instructions are processed by the CPU, but interruptions may take place. When bit 14 is zero, instruction fetching and execution occur in the normal manner. The wait indicator is on when the bit is one.

Problem State (P): When bit 15 is one, the CPU is in the problem state. When bit 15 is zero, the CPU is in the supervisor state. In the supervisor state, all instructions are valid. In the problem state, only those instructions are valid that cannot be used to affect the system integrity. The instructions that are not valid in the problem state are called privileged instructions. When a CPU in the problem state attempts to execute a privileged instruction, a privileged-operation exception is recognized, and a program interruption takes place.

Condition Code (CC): Bits 18 and 19 are the two bits of the condition code. The condition code is set to a value of 0, 1, 2, or 3, depending on the result obtained in executing certain instructions. Most arithmetic and logical operations, as well as some other operations, set the condition code. The instruction BRANCH ON CONDITION can specify any selection of the condition-code values as a criterion for branching. A table in Appendix C summarizes the condition-code values that may be set for all instructions which set the condition code of the PSW.

Program Mask: Bits 20-23 are the four program-mask bits. Each bit is associated with a program exception, as follows:

Program-Mask Bit	Program Exception
20	Fixed-point overflow
21	Decimal overflow
22	Exponent underflow
23	Significance

When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The setting of the exponent-underflow-mask bit or the significance-mask bit also determines the manner in which the operation is completed when the corresponding exception occurs.

Instruction Address: Bits 40-63 form the instruction address. This address designates the location of the leftmost byte of the next instruction.

Bit positions 0, 2-5, 16, 17, and 24-39 are unassigned and must contain zeros. A specification exception is recognized when these bit positions do not contain zeros.

Program-Status-Word Format in BC Mode

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

Channel Masks 0-5: Bits 0-5 control whether the CPU is enabled for I/O interruptions from channels 0-5, respectively. When a bit is zero, the associated channel cannot cause an I/O interruption. When the bit is one, an interruption condition at the channel can cause an I/O interruption.

I/O Mask (IO): Bit 6 controls whether the CPU is enabled for I/O interruptions from channels 6 and higher. When the bit is zero, these channels cannot cause I/O interruptions. When the bit is one, I/O interruptions are subject to the channel-mask bits of the corresponding channels in control register 2: when a channel-mask bit is zero, the associated channel cannot cause an I/O interruption; when the channel-mask bit is one, an interruption condition at the channel can cause an interruption.

External Mask (EX): Bit 7 controls whether the CPU is enabled for interruption by conditions included in the external class. When the bit is zero, an external interruption cannot occur. The meaning is the same as in the EC mode.

PSW Key: Bits 8-11 form the access key for storage references by the CPU. The meaning is the same as in the EC mode.

EC Mode (E): Bit 12, which controls the format of the PSW and the mode of operation of the CPU, is zero when the CPU is in the basic-control (BC) mode.

Machine-Check Mask (M): Bit 13 controls whether the CPU is enabled for interruption by machine-check conditions. The meaning is the same as in the EC mode.

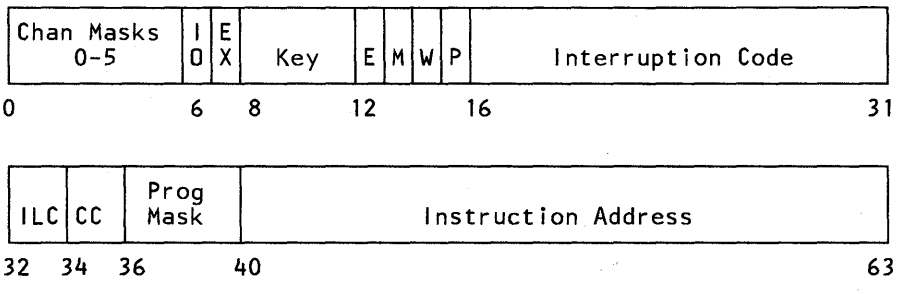
Wait State (W): When bit 14 is one, the CPU is waiting. The meaning is the same as in the EC mode.

Problem State (P): When bit 15 is one, the CPU is in the problem state. When bit 15 is zero, the CPU is in the supervisor state. The meaning is the same as in the EC mode.

Interruption Code: Bits 16-31 in the old PSW, which is stored during a program, supervisor-call, external, or I/O interruption, identify the cause of the interruption. This field is not used or checked in the current PSW. When a new PSW is introduced, the contents of this field are ignored.

Instruction-Length Code (ILC): The code in bit positions 32 and 33 of the old PSW indicates the length of the last-interpreted instruction when a program or supervisor-call interruption occurs. See the section "Instruction-Length Code" in Chapter 6, "Interruptions." When a new PSW is introduced, the contents of this field are ignored.

Condition Code (CC): Bits 34 and 35 are the two bits of the condition code. The meaning is the same as in the EC mode.



PSW Format in BC Mode

Program Mask: Bits 36-39 are the four program-mask bits. Each bit is associated with a program exception, as follows:

Program-Mask Bit	Program Exception
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow
39	Significance

When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The setting of the exponent-underflow-mask bit or the significance-mask bit also determines the manner in which the operation is completed when the corresponding exception occurs.

Instruction Address: Bits 40-63 form the instruction address. This address designates the location of the leftmost byte of the next instruction.

Control Registers

The control registers provide a means for maintaining and manipulating control information that resides outside the PSW. There may be up to sixteen 32-bit control registers.

One or more specific bit positions in control registers are assigned to each facility requiring such register space.

The LOAD CONTROL instruction loads control information from storage into control registers, whereas the STORE CONTROL instruction transfers information from control registers to storage.

The instruction LOAD CONTROL causes all register positions, within those registers designated by the instruction, to be loaded. Information loaded into the control registers becomes active (that is, assumes control over the system) at the completion of the instruction causing the information to be loaded.

When STORE CONTROL is executed, it returns the current value in each register position. Values corresponding to unassigned register positions are unpredictable.

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

Programming Note

To ensure that existing programs run if and when new facilities using additional control-register positions are installed, the program should load zeros in unassigned control-register positions. Although STORE CONTROL may provide zeros in the bit positions corresponding to unassigned register positions, the program should not depend on such zeros. It is permissible, however, for the program to load into the control registers, by LOAD CONTROL, any information previously stored by means of STORE CONTROL.

Ctrl Reg	Bits	Name of Field	Associated with	Initial Value
0	0	Block-multiplexing control	Block-multiplexing channels	0
0	1	SSM-suppression control	SET SYSTEM MASK	0
0	20	Clock-comparator mask	Clock comparator	0
0	21	CPU-timer mask	CPU timer	0
0	24	Interval-timer mask	Interval timer	1
0	25	Interrupt-key mask	Interrupt key	1
0	26	External-signal mask	External signal	1
2	0-31	Channel masks	Channels	1
8	16-31	Monitor masks	MONITOR CALL	0
9	0	Successful-branching-event mask	Program-event recording	0
9	1	Instruction-fetching-event mask	Program-event recording	0
9	2	Storage-alteration-event mask	Program-event recording	0
9	3	GR-alteration-event mask	Program-event recording	0
9	16-31	PER[1] general-register masks	Program-event recording	0
10	8-31	PER starting address	Program-event recording	0
11	8-31	PER ending address	Program-event recording	0
14	0	Unused[2]	Machine-check handling	1
14	4	Recovery-report mask	Machine-check handling	0
14	5	Degradation-report mask	Machine-check handling	0
14	6	External-damage-report mask	Machine-check handling	1
14	7	Warning mask	Machine-check handling	0

Explanation:

The fields not listed are unassigned.
1 PER means program-event recording.
2 This bit is not used but is initialized to one for consistency with the System/370 definition.

Assignment of Control-Register Fields

Program-Event Recording

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

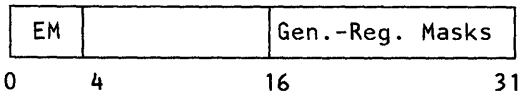
- Execution of a successful branch instruction.
- Fetching of an instruction from the designated storage area.
- Alteration of the contents of the designated storage area.
- Alteration of the contents of designated general registers.

The program can selectively specify one or more of the above types of events to be monitored. The information concerning a PER event is provided to the program by means of a program interruption, with the cause of the interruption being identified in the interruption code. Program-event recording is only available in the EC mode.

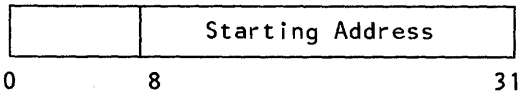
Control-Register Allocation

The information for controlling program-event recording resides in control registers 9, 10, and 11 and consists of the following fields:

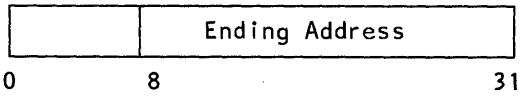
Control Register 9:



Control Register 10:



Control Register 11:



PER-Event Masks (EM): Bits 0-3 of control register 9 specify which types of events are monitored. The bits are assigned as follows:

Bit 0: Successful-branching event
Bit 1: Instruction-fetching event
Bit 2: Storage-alteration event
Bit 3: General-register-alteration event

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

PER General-Register Masks: Bits 16-31 of control register 9 specify which general registers are monitored for replacement of their contents. The 16 bits, in the order of ascending bit numbers, correspond one for one with the 16 registers, in the order of ascending register numbers. When a bit is one, the associated register is monitored for replacement; if zero, the register is not monitored.

PER Starting Address: Bits 8-31 of control register 10 are the address of the beginning of the monitored storage area.

PER Ending Address: Bits 8-31 of control register 11 are the address of the end of the monitored storage area.

Programming Note

Models may operate at reduced performance while the CPU is enabled for PER events. To ensure that CPU performance is not degraded because of the operation of the program-event-recording facility, programs that do not use it should disable the facility by setting the PER mask in the EC-mode PSW to zero. No degradation due to program-event recording occurs in the BC mode or when the PER mask in the EC-mode PSW is zero. Disabling of program-event recording in the EC mode by means of the masks in control register 9 does not necessarily prevent performance degradation due to the facility.

Operation

Program-event recording (PER) is under control of bit 1 of the EC-mode PSW, the PER mask. When the mask is zero, no PER event can cause an interruption. When the mask is one, a monitored event, as specified by the contents of control registers 9, 10, and 11, causes a program interruption. In BC mode, program-event recording is disabled.

An interruption due to a PER event is taken after the execution of the instruction responsible for the event. The occurrence of the event does not affect the execution of the instruction, which may be either completed, terminated, suppressed, or nullified.

When the CPU is disabled for a particular PER event at the time it occurs, either by the mask in the PSW or by the masks in control register 9, the event is not recognized.

A change to the PER mask in the PSW or to the PER control fields in control registers 9, 10, and 11 affects program-event recording starting with the execution of the immediately following instruction. If the CPU is enabled for some PER event but an instruction causes the CPU to be disabled for that particular event, the event causes a PER condition to be recognized if it occurs during the execution of the instruction.

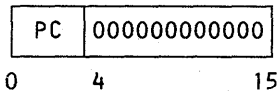
When LOAD PSW or SUPERVISOR CALL causes a PER condition and at the same time changes CPU operation from the EC mode to the BC mode, the PER interruption is taken with the old PSW specifying the BC mode and with the interruption code stored in the old PSW. The additional information identifying the PER condition is stored in its regular format at locations 150-155.

Program-event recording applies to emulation instructions in the following way. Emulation instructions indicate all events that have occurred and may additionally indicate events that did not occur and were not called for in the instruction, provided monitoring was enabled for the type of event by the PER mask in the PSW and the PER-event masks, bits 0-3 in control register 9. In such cases, the contents of the remaining positions in control registers 9, 10, and 11 may be ignored. Thus, for example, an emulation instruction may cause general-register alteration to be indicated even though no general registers are altered and even though bits 16-31 of control register 9 are all zeros.

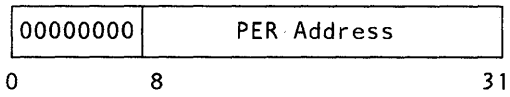
Identification of Cause

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

Locations 150-151:



Locations 152-155:



The event causing a PER interruption is identified by a one in bit positions 0-3 of location 150, the PER code (PC), with the rest of the bits in the code set to zeros. The bit position in the PER code for a particular event is the same as the bit position for that event in the PER event-mask field in control register 9.

The PER address at locations 153-155 is the address of the instruction causing the event. When the instruction is executed by means of EXECUTE, the address of the location containing the EXECUTE instruction is placed in the PER-address field. In either case, the address of the instruction to be executed next is placed in the PSW. Zeros are stored in bit positions 4-7 of location 150 and at locations 151 and 152.

Priority of Indication

When a PER interruption occurs and more than one designated PER event has been recognized, all recognized PER events are concurrently indicated in the PER code. Additionally, if another program interruption condition concurrently exists, the interruption code for a program interruption indicates both the PER condition and the other condition.

Except as listed below, a PER event does not cause premature interruption of the interruptible instruction, and the PER condition is held pending until the completion of the instruction.

- When the execution of an interruptible instruction is due to be interrupted by an I/O, external, or repressible machine-check condition, an interruption for a pending PER condition occurs first, and the I/O, external, or machine-check interruption is subsequently subject to the control of mask bits in the new PSW.
- Similarly, when the CPU is placed in the stopped state during the execution of an interruptible instruction, an interruption for a pending PER condition occurs before the stopped state is entered.
- When any program exception is encountered, the pending PER condition is indicated concurrently.
- Depending on the model, in certain situations, a PER condition may cause the execution of an interruptible instruction to be interrupted without an associated asynchronous condition or program exception.

In the case of an instruction-fetching event for SUPERVISOR CALL, the PER interruption occurs immediately after the supervisor-call interruption.

Programming Notes

1. In the following cases an instruction can both cause a program interruption for a PER event and change the value of masks controlling an interruption for PER events. The original mask values determine whether a program interruption takes place for the PER event.
   a. The instructions LOAD PSW, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and SUPERVISOR CALL can cause an instruction-fetching event and disable the CPU for PER interruptions. Additionally, STORE THEN AND SYSTEM MASK can cause a storage-alteration event to be indicated. In

all these cases, the program old PSW associated with the program interruption for the PER event may indicate that the CPU was disabled for that type of PER event.

b. An instruction-fetching event may be recognized during execution of a LOAD CONTROL instruction which also changed the value of the PER-event masks in control register 9 or the addresses in control registers 10 and 11 controlling indication of instruction-fetching events.

2. No instructions can both change the values of general-register-alteration masks and cause a general-register-alteration event to be recognized.
3. When a PER interruption occurs during the execution of an interruptible instruction, the ILC indicates the length of that instruction or EXECUTE, as appropriate. When a PER interruption occurs as a result of LOAD PSW or SUPERVISOR CALL, the ILC indicates the length of these instructions or EXECUTE, as appropriate, unless a concurrent specification exception on LOAD PSW calls for an ILC of 0.
4. When a PER interruption is caused by branching, the PER address identifies the branch instruction (or EXECUTE, as appropriate), whereas the old PSW points to the next instruction to be executed. When the interruption occurs during the execution of an interruptible instruction, the PER address and the instruction address in the old PSW are the same.

Storage-Area Designation

Two of the PER events–instruction fetching and storage alteration–involve the designation of an area in storage. The storage area monitored for the references starts at the location designated by the starting address in control register 10 and extends up to and including the location designated by the ending address in control register 11. The area extends to the right of the starting address.

The set of addresses monitored for instruction-fetching and storage-alteration events wraps around at address 16,777,215; that is, address 0 is considered to follow address 16,777,215. When the starting address is less than the ending address, the area is contiguous. When the starting address is greater than the ending address, the set of locations monitored includes the area from the starting address to address 16,777,215 and the area from address 0 to, and including, the ending address. When the starting address is equal to the ending address, only the location designated by that address is monitored.

The monitoring of storage alteration and instruction fetching is performed by comparing all 24 bits of the monitored address with the starting and ending addresses.

PER Events

Successful Branching

Execution of a successful branch operation causes a program-event interruption if bit 0 of the PER-event-mask field is one and the PER mask in the PSW is one.

A successful branch occurs whenever one of the following instructions causes control to be passed to the instruction designated by the branch address:

BRANCH ON CONDITION
BRANCH AND LINK
BRANCH ON COUNT
BRANCH ON INDEX HIGH
BRANCH ON INDEX LOW OR EQUAL

The branch event is also indicated by an emulation instruction when the emulation instruction itself causes a branch. That is, the branch event is indicated when the location of the next instruction executed by the CPU after leaving emulation mode does not immediately follow the location of the emulation instruction.

The event is indicated by setting bit 0 of the PER code to one.

Instruction Fetching

Fetching the first byte of an instruction from the storage area designated by the contents of control registers 10 and 11 causes a program-event interruption if bit 1 of the PER-event-mask field is one and the PER mask in the PSW is one.

A PER event for instruction fetching is recognized whenever the CPU executes an instruction whose initial byte is located within the monitored area. When the instruction is executed by means of EXECUTE, a PER event is recognized when the first byte of the EXECUTE instruction or the target instruction or both is located in the monitored area.

The event is indicated by setting bit 1 of the PER code to one.

Storage Alteration

Storing of data by the CPU in the storage area designated by the contents of control registers 10 and 11 causes a program-event interruption if bit 2

of the PER-event-mask field is one and the PER mask in the PSW is one.

The contents of storage are considered to have been altered whenever the CPU executes an instruction that causes all or part of an operand to be stored within the monitored area of storage. Alteration is considered to take place whenever storing is considered to take place for purposes of indicating protection exceptions. (See the section "Recognition of Access Exceptions" in Chapter 6, "Interruptions.") Storing constitutes alteration for program-event-recording purposes even if the value stored is the same as the original value.

Implied locations that are referred to by the CPU in the process of interval-timer updating, interruptions, and execution of I/O instructions, including the interval-timer, PSW, and CSW locations, are not monitored. These locations, however, are monitored when information is stored there explicitly by an instruction. Similarly, monitoring does not apply to storing of data by a channel.

The instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP are considered to alter the second-operand location only when storing actually occurs.

The instruction STORE CHARACTERS UNDER MASK is not considered to alter the storage location when the mask is zero.

The event is indicated by setting bit 2 of the PER code to one.

General-Register Alteration

Alteration of the contents of a general register causes a program-event interruption if bit 3 of the PER-event-mask field is one, the alteration mask corresponding to that general register is one, and the PER mask in the PSW is one.

The contents of a general register are considered to have been altered whenever a new value is placed in the register. Recognition of the event is not contingent on the new value being different from the previous one. The execution of an RR-format arithmetic or movement instruction is considered to fetch the contents of the register, perform the indicated operation, if any, and then replace the value in the register. The register can be designated implicitly, such as in TRANSLATE AND TEST and EDIT AND MARK, or explicitly by an RR, RX, or RS instruction, including BRANCH AND LINK, BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL.

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

The instructions MOVE LONG and COMPARE LOGICAL LONG are always considered to alter the contents of the four registers specifying the two operands, including the cases where the padding byte is used, when both operands have zero length, or when condition code 3 is set for MOVE LONG.

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

The instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP are considered to alter the general register, or general-register pair, designated by R_1, only when the contents are actually replaced, that is, when the first and second operands are not equal.

The event is indicated by setting bit 3 of the PER code to one.

Programming Note

The following are some examples of general-register alteration:

1. Register-to-register load instructions are considered to alter the register contents even when both operand addresses designate the same register.
2. Addition or subtraction of zero and multiplication or division by one are considered to constitute alteration.
3. Logical and fixed-point shift operations are considered to alter the register contents even for shift amounts of zero.
4. The branching instructions BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL are considered to alter the first operand even when zero is added to its value.

Indication of Events Concurrently with Other Interruption Conditions

The following rules govern the indication of PER events caused by an instruction that has also caused a program exception or the monitor event to be indicated, or that causes a supervisor-call interruption.

1. The indication of an instruction-fetching event does not depend on whether the execution of the instruction was completed, terminated, suppressed, or nullified. The event, however, is not indicated when an access exception prohibits access to the first byte of the instruction. When the first halfword of the instruction is accessible but an access exception

applies to the second or third halfword of the instruction, it is unpredictable whether the instruction-fetching event is indicated.

2. When the operation is completed, the event is indicated regardless of whether any program exception or the monitoring event is recognized.
3. Successful branching, storage alteration, and general-register alteration are not indicated for an operation or, in case the instruction is interruptible, for a unit of operation that is suppressed or nullified.
4. When the execution of the instruction is terminated, general-register or storage alteration is indicated whenever the event has occurred, and a model may indicate the event if the event would have occurred had the execution of the instruction been completed, even if altering the contents of the result field is contingent on operand values.
5. When LOAD PSW or SUPERVISOR CALL causes a PER condition and at the same time introduces a new PSW with the type of PSW-format error that is recognized immediately after the PSW becomes active, the interruption code identifies both the PER condition and the specification exception. When these instructions introduce a PSW-format error of the type that is recognized as part of the execution of the following instruction, the PSW is stored as the old PSW without the specification exception being recognized.

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

Exception	Type of Ending	PER Event			
		Branch	Instr Fetch	Storage Alter-ation	GR Alter-ation
Operation	S	-	X[1]	-	-
Privileged operation	S	-	X[1]	-	-
Execute	S	-	X[1]	-	-
Protection					
Instruction	S	-	-[1]	-	-
Operand	S or T	-	X	X+	X+
Addressing					
Instruction	S	-	-[1]	-	-
Operand	S or T	-	X	X+	X+
Specification					
Odd instruction address	S	-	-	-	-
Invalid PSW format	C	-	X	-	-
Other	S	-	X	-	-
Data					
Invalid sign	S	-	X	-	-
Other	T	-	X	X+	X+
Fixed-point overflow	C	-	X	-	X
Fixed-point divide					
Division	S	-	X	-	-
Conversion	C	-	X	-	X
Decimal overflow	C	-	X	X	-
Decimal divide	S	-	X	-	-
Exponent overflow	C	-	X	-	-
Exponent underflow	C	-	X	-	-
Significance	C	-	X	-	-
Floating-point divide	S	-	X	-	-
Special operation	S	-	X	-	-
Page access					
Instruction	N	-	-[1]	-	-
Operand	N	-	X	X[2]	X[2]
Page state	S	-	X	-	-
Page transition	S	-	X	-	-
Monitor event	C	-	X	-	-

Indication of PER Events (Part 1 of 2)

Explanation:

C The operation or, in the case of the interruptible instructions, the unit of operation is completed.

N The operation or, in the case of the interruptible instructions, the unit of operation is nullified. The instruction address in the old PSW has not been updated.

S The operation or, in the case of the interruptible instructions, the unit of operation is suppressed.

T The execution of the instruction is terminated.

X The event is indicated with the exception if the event has occurred; that is, the contents of the monitored storage location or general register were altered, or an attempt was made to execute an instruction whose first byte is located in the monitored area.

\+ A model is permitted, but not required, to indicate the event if the event would have occurred had the operation been completed but did not take place because the execution of the instruction was terminated.

\- The event is not indicated.

1 When an access exception applies to the second or third halfword of the instruction but the first halfword is accessible, it is unpredictable whether the instruction-fetching event is indicated.

2 This condition may occur in the case of the interruptible instructions when the event is recognized in the unit of operation that is completed and when the exception causes the next unit of operation to be suppressed or nullified.

Indication of PER Events (Part 2 of 2)

Programming Notes

1. The execution of the interruptible instructions MOVE LONG (MVCL) and COMPARE LOGICAL LONG (CLCL) can cause events for general-register alteration and instruction fetching. Additionally, MVCL can cause the storage-alteration event.

   Since the execution of MVCL and CLCL can be interrupted, a program event may be indicated more than once. It may be necessary, therefore, for a program to remove the redundant event indications from the PER data. The following rules govern the indication of the applicable events during execution of these two instructions:

   a. The instruction-fetching event is indicate whenever the instruction is fetched for execution, regardless of whether it is the initial execution or a resumption.
   b. The general-register-alteration event is indicated on the initial execution and on each resumption and does not depend on whether or not the register actually is changed.
   c. The storage-alteration event is indicated only when data has been stored in the monitored area by the portion of the operation starting with the last initiation and ending with the last byte transferred

before the interruption. No special indication is provided on premature interruptions as to whether the event will occur again upon the resumption of the operation. When the storage area designates a single byte location, a storage-alteration event can be recognized only once in the execution of MOVE LONG.

2. The following is an outline of the general action a program must take to delete the redundant entries in the PER data for MOVE LONG and COMPARE LOGICAL LONG so that only one entry for each complete execution of the instruction is obtained:
   a. Check to see if the PER address is equal to the instruction address in the old PSW and if the last instruction executed was MVCL or CLCL.
   b. If both conditions are met, delete instruction-fetching and register-alteration events.
   c. If both conditions are met and the event is storage alteration, delete the event if some part of the remaining destination operand is within the monitored area.

External-Signal Facility

The external-signal facility consists of six signal-in lines and an external-signal mask, which is bit 26 of control register 0. Each of the six signal-in lines, when pulsed, sets up the condition for one of six distinct interruptions (see the section "External Signal" in Chapter 6, "Interruptions").

For a detailed description, see the *System/360 and System/370 Direct Control and External Interruption Features–Original Equipment Manufacturers' Information,* GA22-6845.

Timing

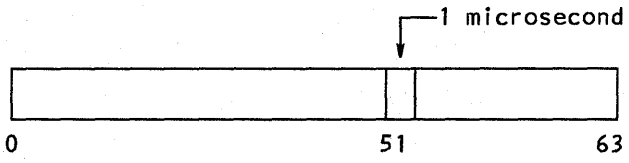
The timing facilities include four facilities for measuring time: the time-of-day clock, the clock comparator, the CPU timer, and the interval timer.

Time-of-Day Clock

The time-of-day (TOD) clock provides a high-resolution measure of real time suitable for the indication of date and time of day. The cycle of the clock is approximately 143 years.

Format

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



In the basic form, the time-of-day clock is incremented by adding a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is incremented at such a frequency that the rate of advancing the clock is the same as if a one were added in bit position 51 every microsecond. The resolution of the time-of-day clock is such that the incrementing rate is comparable to the instruction-execution rate of the model.

When incrementing of the clock causes a carry to be propagated out of bit position 0, the carry is ignored, and counting continues from zero on. The program is not alerted, and no interruption condition is generated as a result of the overflow.

The operation of the clock is not affected by any normal activity or event in the system. Incrementing of the clock does not depend on whether the wait-state bit of the PSW is one or whether the CPU is in the stopped, operating, or load state. Its operation is not affected by program, initial-program, or clear resets or by initial program loading. Operation of the clock is also not affected by the setting of the rate control or by an initial-microprogram-loading operation. The clock is not incremented when the power is off.

States

The following states are distinguished for the time-of-day clock: set, not set, error, and not operational. The state determines the condition code set by execution of STORE CLOCK. The clock is incremented, and is said to be running, when it is in either the set state or the not-set state.

Not-Set State: When the power is turned on, the clock is set to zero, and the clock enters the not-set state. The clock is incremented when in the not-set state. Incrementing begins at zero.

When the clock is in the not-set state, execution of STORE CLOCK causes condition code 1 to be set and the current value of the running clock to be stored.

Set State: Execution of SET CLOCK when the manual TOD-clock control is set to the enable-set position causes the clock to enter the set state from the not-set, set, or error state if no exceptions are encountered.

Incrementing of the clock begins with the first stepping pulse after the clock enters the set state.

When the clock is in the set state, execution of STORE CLOCK causes condition code 0 to be set and the current value of the running clock to be stored.

Error State: The clock enters the error state when a malfunction is detected that is likely to have affected the validity of the clock value. A timing-facility-damage machine-check-interruption condition is generated whenever it enters the error state.

When STORE CLOCK is executed and the clock is in the error state, condition code 2 is set, and the value stored is zero.

Not-Operational State: The clock is in the not-operational state when it is disabled for maintenance. It depends on the model if the clock can be placed in this state. Whenever the clock enters the not-operational state, a timing-facility-damage machine check is generated.

When the clock is in the not-operational state, execution of STORE CLOCK causes condition code 3 to be set, and zero is stored.

Setting and Inspecting the Clock

The clock can be set to a specific value by execution of SET CLOCK if the manual TOD-clock control is set to the enable-set position. Setting the clock replaces the values in all bit positions from bit position 0 through the rightmost position that is incremented when the clock is running.

The time-of-day clock can be inspected by executing STORE CLOCK, which causes a 64-bit value to be stored. Two executions of STORE CLOCK always store different values if the clock is running.

The values stored for a running clock always correctly imply the order of execution of STORE CLOCK. Zeros are stored in positions to the right of the bit position that is incremented.

Programming Notes

1. Bit position 31 of the clock is incremented every 1.048576 seconds; for some applications, reference to the high-order 32 bits of the clock may provide sufficient resolution.
2. Communication between systems is facilitated by establishing a standard time origin, or standard epoch, which is the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 AM Greenwich Mean Time (GMT) is recommended as the standard epoch for the clock.
3. A program using the clock value as a time-of-day and calendar indication must be consistent with the programming support under which the program is to run. If the programming support uses the standard epoch, bit 0 of the clock remains one through the years 1972-2041. Ordinarily, testing the high-order bit for a one is sufficient to determine if the clock value is in the standard epoch.

   In converting to or from the current date or time, the programming support assumes each day to be 86,400 seconds. It does not take into account "leap seconds" inserted or deleted because of time-correction standards.
4. Because of the limited accuracy of manually setting the clock value, the low-order bit positions of the clock, expressing fractions of a second, are normally not valid as indications of the time of day. However, they permit elapsed-time measurements of high resolution.
5. The following chart shows the time interval between instants at which various bit positions of the time-of-day clock are stepped. This time value may also be considered as the weighted time value that the bit, when one, represents.

TOD-Clock Bit	Stepping Interval			
	Days	Hours	Minutes	Seconds
51				0.000 001
47				0.000 016
43				0.000 256
39				0.004 096
35				0.065 536
31				1.048 576
27				16.777 216
23			4	28.435 456
19		1	11	34.967 296
15		19	5	19.476 736
11	12	17	25	11.627 776
7	203	14	43	6.044 416
3	3257	19	29	36.710 656

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

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

7. The stepping value of time-of-day-clock bit position 63, if implemented, is 2^{-12} microseconds, or approximately 244 picoseconds. This value is called a clock unit.

   The following chart shows various time intervals in clock units expressed in hexadecimal notation.

Interval	Clock Units (Hex)
1 microsecond	1000
1 millisecond	3E 8000
1 second	F424 0000
1 minute	39 3870 0000
1 hour	D69 3A40 0000
1 day	1 41DD 7600 0000
365 days	1CA E8C1 3E00 0000
366 days	1CC 2A9E B400 0000
1,461 days[1]	72C E4E2 6E00 0000

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

Clock Comparator

The clock comparator provides a means of causing an interruption when the time-of-day-clock value exceeds a value specified by the program.

The clock comparator has the same format as the time-of-day clock. In the basic form, the clock comparator consists of bits 0-47, which are compared with the corresponding bits of the time-of-day clock. In some models, higher resolution is obtained by providing more than 48 bits. The bits in positions provided in the clock comparator are compared with the corresponding bits of the clock. When the resolution of the clock is less than that of the clock comparator, the contents of the clock comparator are compared with the clock value as this value would be stored by executing STORE CLOCK.

The clock comparator causes an external interruption with the interruption code 1004 (hex). A request for a clock-comparator interruption exists whenever either of the following conditions exists:

1. The time-of-day clock is running and the value of the clock comparator is less than the value in the compared portion of the clock, both values being considered unsigned binary integers. Comparison follows the rules of unsigned binary arithmetic.
2. The time-of-day clock is in the error state or the not-operational state.

A request for a clock-comparator interruption does not remain pending when the value of the clock comparator is made equal to or greater than that of the time-of-day clock or when the value of the time-of-day clock is made less than the clock-comparator value. The latter may occur as a result of the time-of-day clock either being set or wrapping to zero.

The clock comparator can be inspected by executing the instruction STORE CLOCK COMPARATOR and can be set to a specific value by executing the SET CLOCK COMPARATOR instruction.

The contents of the clock comparator are initialized to zero by initial program reset.

Programming Notes

1. An interruption request for the clock comparator persists as long as the clock-comparator value is less than that of the time-of-day clock or as long as the time-of-day clock is in the error or not-operational state. Therefore, one of the following actions must be taken after an external interruption for the clock comparator has occurred and before the CPU is again enabled for external interruptions: the value of the clock comparator has to be replaced, the time-of-day clock has to be set, or the clock-comparator submask has to be set to zero. Otherwise, loops of external interruptions are formed.
2. The instruction STORE CLOCK may store a value which is greater than that in the clock comparator, even though the CPU is enabled for the clock-comparator interruption. This is because the time-of-day clock may be

incremented one or more times between when instruction execution is begun and when the clock value is accessed. In this situation, the interruption occurs when the execution of STORE CLOCK is completed.

CPU Timer

The CPU timer provides a means for measuring elapsed CPU time and for causing an interruption when a prespecified amount of time has elapsed.

The CPU timer is a binary counter with a format which is the same as that of the time-of-day clock, except that bit 0 is considered a sign. In the basic form, the CPU timer is decremented by subtracting a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is decremented at such a frequency that the rate of decrementing the CPU timer is the same as if a one were subtracted in bit position 51 every microsecond. The resolution of the CPU timer is such that the stepping rate is comparable to the instruction-execution rate of the model.

The CPU timer requests an external interruption with the interruption code 1005 (hex) whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). The request does not remain pending when the CPU-timer value is changed to a nonnegative value.

When both the CPU timer and the time-of-day clock are running, the stepping rates are synchronized such that both are stepped at the same rate. Normally, decrementing the CPU timer is not affected by concurrent I/O activity. However, in some models the CPU timer may stop during extreme I/O activity and other similar interference situations. In these cases, the time recorded by the CPU timer provides a more accurate measure of the CPU time used by the program than that which would have been recorded had the CPU timer continued to step.

The CPU timer is decremented when the CPU is in the operating state or the load state. When the manual rate control is set to instruction step, the CPU timer is decremented only during the time in which the CPU is actually performing a unit of operation. However, depending on the model, the CPU timer may or may not be decremented when the time-of-day clock is in the error or not-operational state.

Depending on the model, the CPU timer may or may not be decremented when the CPU is in the check-stop state.

The CPU timer can be inspected by executing the instruction STORE CPU TIMER and can be set to a specific value by executing the SET CPU TIMER instruction.

The CPU timer is set to zero by initial program reset.

Programming Notes

1. The CPU timer in association with a program may be used both to measure CPU-execution time and to signal the end of a time interval on the CPU.
2. The time measured for the execution of a sequence of instructions may depend on the effects of such things as I/O interference, page faults, and instruction retry. Hence, repeated measurements of the same sequence on the same installation may differ.
3. The fact that a CPU-timer interruption does not remain pending when the CPU timer is set to a positive value eliminates the problem of an undesired interruption. This would occur if, between the time when the old value is stored and a new value is set, the CPU is disabled for CPU-timer interruptions and the CPU timer value goes from positive to negative.
4. The fact that CPU-timer interruptions are requested whenever the CPU timer is negative rather than just when the CPU timer goes from positive to negative eliminates the requirement for testing a value to ensure that it is positive before setting the CPU timer to that value.

   As an example, a program being timed by the CPU timer is interrupted for a cause other than the CPU timer, external interruptions are disallowed by the new PSW, and the CPU-timer value is then saved by STORE CPU TIMER. This value could be negative if the CPU timer went from positive to negative since the interruption. Subsequently, when the program being timed is to continue, the CPU timer may be set to the saved value by SET CPU TIMER. A CPU-timer interruption will occur immediately after external interruptions are again enabled if the saved value was negative.

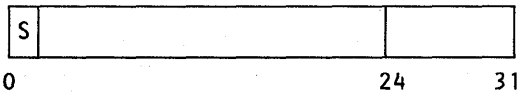
   The persistence of the CPU-timer-interruption request means, however, that after an external interruption for the CPU timer has occurred, either the value of the CPU timer has to be replaced or the CPU-timer submask has to be set to zero before the CPU is again

enabled for external interruptions. Otherwise, loops of external interruptions are formed.

5. The instruction STORE CPU TIMER may store a negative value even though the CPU is enabled for the interruption. This is because the CPU-timer value may be decremented one or more times between the instants when instruction execution is begun and when the CPU timer is accessed. In this situation, the interruption occurs when the execution of STORE CPU TIMER is completed.

Interval Timer

The interval timer is a binary counter that occupies a word at storage location 80 and has the following format:



The interval timer is treated as a 32-bit signed binary integer. In the basic form, the contents of the interval timer are reduced by one in bit position 23 every 1/300 of a second. Higher resolution of timing may be obtained in some models by counting with higher frequency in one of the positions 24 through 31. In each case, the frequency is adjusted to cause decrementing in bit position 23 at the rate of 300 times per second. The cycle of the interval timer is approximately 15.5 hours.
of 300 times per second. The cycle of the interval timer is approximately 15.5 hours.

The interval timer causes an external interruption, with bit 8 of the interruption code set to one and bits 0-7 set to zeros. Bits 9-15 of the interruption code are zeros unless set to ones for another condition that is concurrently indicated.

A request for an interval-timer interruption is generated whenever the interval-timer value is decremented from a positive or zero number to a negative number. The request is preserved and remains pending in the CPU until it is cleared by an interval-timer interruption or a program reset. The overflow occurring as the interval-timer value is decremented from a large negative number to a large positive number is ignored.

The interval timer is not necessarily synchronized with the time-of-day clock.

The interval-timer contents are updated at the appropriate frequency whenever other machine activity permits. The updating occurs only between instruction executions, except that the interval timer may be updated between units of operation of an interruptible instruction, such as MOVE LONG. An updated interval-timer value is normally available at the end of each instruction execution. When the execution of an instruction or other machine activity causes updating to be delayed by more than one period, the contents of the interval timer may be reduced by more than one unit in a single updating cycle. Interval-timer updating may be omitted when I/O data transmission approaches the limit of storage capability, or when a channel sharing CPU equipment and operating in burst mode causes CPU activity to be locked out. The program is not alerted when omission of updating causes the real-time count to be lost.

When the contents of the interval timer are fetched by a channel or are used as the source of an instruction, the result is unpredictable. Similarly, storing by the channel at location 80 causes the contents of the interval timer to be unpredictable.

The interval timer is not decremented when the manual interval-timer control is set to disable. The interval timer is also not decremented when the CPU is not in the operating state or when the manual rate control is set to instruction step.

Depending on the model, the interval timer may or may not be decremented when the time-of-day clock is in the error, stopped, or not-operational state.

Programming Notes

1. The value of the interval timer is accessible by fetching the word at location 80 as an operand, provided the location is not protected against fetching. It may be changed at any time by storing a word at location 80. When location 80 is protected, any attempt by the program to change the value of the interval timer causes a program interruption for protection exception.
2. The value of the interval timer may be changed without losing the real-time count by storing the new value at locations 84-87 and then shifting bytes 80-87 to locations 76-83 by means of the instruction MOVE (MVC). Thus, in a single operation, the new interval-timer value is placed at location 80, and the old value is made available at location 76.

   If any means other than the instruction MOVE (MVC) are used to interrogate and then replace the value of the interval timer,

including MOVE LONG or two separate instructions, the program may lose a time increment when an updating cycle occurs between fetching and storing.

3. When the value of the interval timer is to be recorded on an I/O device, the program should first store the interval-timer value in a temporary storage location to which the I/O operation subsequently refers. When the channel fetches the interval-timer value directly from location 80, the value obtained is unpredictable.

Externally Initiated Functions

Resets

Four reset functions are provided:

- Program reset
- Initial program reset
- Clear reset
- Power-on reset

Program reset provides a means of clearing equipment-check indications and any resultant unpredictability in the CPU and I/O state with the least amount of information destroyed. In particular, it is used to clear check conditions when the machine state is to be preserved for analysis or resumption of operation.

Initial program reset provides the functions of program reset together with initialization of the current PSW, CPU timer, clock comparator, and control registers.

Clear reset causes initial program reset to be performed and, additionally, clears or initializes all storage locations and registers, with the exception of the time-of-day clock. Such clearing is useful in debugging programs and in ensuring user privacy. Clearing does not affect external storage, such as direct-access storage devices used by the control program to hold the contents of unaddressable pages.

Power-on reset combines the functions of clear reset with initializing the time-of-day clock and selecting storage size.

Program reset and clear reset are initiated manually using the operator facilities (see Chapter 13, "Operator Facilities"). Initial program reset is part of the initial-program-loading function. Power-on reset is performed as part of turning power on. The reset actions are tabulated in the figure "Summary of Reset Actions."

Area Affected	Reset Function			
	Program Reset	Initial Program Reset	Clear Reset	Power-on Reset
CPU	S	S	S	S
Channels	R	R	R	R
PSW	U	C	C	C
CPU timer	U	C	C	C
Clock comparator	U	C	C	C
Time-of-day clock	U[1]	U[1]	U[1]	T
Control registers	U	I	I	I
General registers	U	U	C	C
Floating-point registers	U	U	C	C
Capacity counts	U	U	I	I
Page descriptions	U	U	C	C
Storage	U	U	P	P
Machine-save information	U[2]	U[2]	Y	Y

Explanation:

C The condition or contents are cleared. If the area affected is a field, the contents are cleared to zero with valid checking-block code.

I The state or contents are initialized. If the area affected is a field, the contents are set to their initial values with valid checking-block code.

P The first n storage pages are cleared and made addressable, where n is the lesser of the available-frame-capacity and page-capacity counts. Any remaining pages are left disconnected.

R I/O-system reset is performed in the channels, and pending I/O-interruption conditions are cleared. As part of this reset, system reset is signaled to the I/O control units and devices configured to the channels.

S The CPU is reset; current operations, if any, are terminated; interruption conditions in the CPU are cleared; and the CPU is placed in the stopped state.

T The time-of-day clock is initialized to zero and validated, and it enters the not-set state.

U The contents remain unchanged. However, the resulting value is unpredictable if an operation is in progress that changes the contents of the field at the time of reset.

Y The machine-save information is made invalid.

1 Access to the time-of-day clock by means of STORE CLOCK at the time a reset function is performed does not affect the value of the time-of-day clock.

2 If a machine-save function is in progress at the time of the reset, the machine-save function is halted, and any partially altered machine-save information is made invalid.

Summary of Reset Actions

Program Reset

Program reset causes the following actions:

1. The execution of the current instruction or other processing sequence, such as an interruption, is terminated, and all program-interruption and supervisor-call-interruption conditions are cleared.
2. Any pending external-interruption conditions are cleared.
3. Any pending machine-check-interruption conditions, error indications, and check-stop state are cleared.
4. Any buffers containing prefetched instructions, operands, or results due to be stored are cleared.
5. The CPU is placed in the stopped state after actions 1-4 have been completed.
6. I/O-system reset is performed in each channel.
7. Any ongoing machine-save function is halted, and any partially altered machine-save information is made invalid.

Register and storage contents remain unchanged by program reset. However, if a register or storage location is being accessed at the time the program-reset operation is performed, the subsequent contents of the register or location are unpredictable.

As part of the I/O-system reset performed (see the section "I/O-System Reset" in Chapter 12, "Input/Output Operations"), pending I/O-interruption conditions are cleared, and system reset is signaled to all control units and devices configured to the channel. The effect of system reset on I/O control units and devices and the resultant control-unit and device state are described in the appropriate publication on the control unit or device. A system reset, in general, resets only those functions in a shared control unit or device that are associated with the particular channel signaling the reset.

Program reset is performed when the system-reset-normal key is activated. It is also part of the initial-program-reset function.

Initial Program Reset

Initial program reset combines the program-reset functions with the following actions:

1. The contents of the current PSW, CPU timer, and clock comparator are set to zero.
2. All assigned control-register positions are set to their initial values.

These clearing and initializing functions include validation.

Setting the current PSW to zero causes the PSW to assume the BC-mode format. The instruction-length code and interruption code in the PSW are unpredictable, because these values are not retained when a new PSW is introduced.

Initial program reset is part of the clear-reset function. It is also part of the initial-program-loading function when the load-normal or load-clear key is activated.

Clear Reset

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

1. The general and floating-point registers are set to zero.
2. The storage key of every storage page is set to zero.
3. The page bits of every storage page are set to zeros.
4. All page frames that had been made temporarily unavailable by DECONFIGURE PAGE instructions are made available. (This excludes frames made permanently unavailable by maintenance intervention.)
5. The page-capacity, existing-frame-capacity, available-frame-capacity, and free-frame-capacity counts are initialized.
6. Let *n* be the lesser of AFCC, the current available-frame-capacity count, and PCC, the page-capacity count. Then each of *n* page frames is assigned to one of the first *n* storage pages, namely those with page addresses 0 to n minus one. These pages are cleared to zero bytes and have their page states set to addressable. Any remaining pages have their page states set to disconnected.
7. Any previously saved machine-save information is invalidated.

Validation is included in setting registers and capacity counts and in clearing storage and page descriptions.

Clear reset is performed when the system-reset-clear key is activated. Clear reset is also part of the power-on-reset function, and part of the initial-program-loading function when performed upon activating the load-clear key.

Programming Notes

1. For the program-reset operation not to affect the contents of fields that are to be left unchanged, the CPU must not be executing instructions and must be disabled for all interruptions at the time of the reset. Except for the operation of the time-of-day clock, interval timer, and CPU timer and for the possibility of taking a machine-check

interruption, all CPU activity can be quiesced by placing the CPU in the wait state and by disabling it for I/O and external interruptions. To avoid the possibility of causing a reset at the time the timing facilities are being updated or a machine-check interruption occurs, the CPU must be in the stopped state.

2. Program reset, initial program reset, and clear reset do not affect the value and state of the time-of-day clock.
3. The conditions under which the CPU enters the check-stop state are model-dependent and include malfunctions that preclude the completion of the current operation. Hence, if program reset or initial program reset is executed while the CPU is in the check-stop state, the contents of the PSW, registers, and storage locations, including the page descriptions and the storage location accessed at the time of the error, may still be in error after the check-stop state is cleared by these resets. In such a case, a clear reset is required to clear the error.
4. Clear reset causes all bit positions of the interval timer to be cleared to zeros.
5. Program reset and initial program reset leave machine-save information unchanged if no machine save is being performed at the time of the reset.

Power-On Reset

Power-on reset causes the following actions:

1. The clear-reset function is performed.
2. The value of the time-of-day clock is set to zero, and the clock enters the not-set state.

Power-on reset is part of the power-on sequence of the machine. The power-on sequence is not complete until the clear-reset function has been performed successfully and the time-of-day clock has entered the not-set state. The power-on sequences for control units and I/O devices are described in the appropriate System Library (SL) publications.

Initial Program Loading

Initial program loading (IPL) is provided to initiate processing when the contents of storage or of the PSW are not suitable for processing.

Initial program loading is initiated manually by designating an input device with the load-unit-address controls and subsequently activating the load-normal or load-clear key. The load-normal key causes an initial-program-reset operation to be performed, and the load-clear key causes a clear-reset operation to be performed. The CPU enters the load state. Subsequently, a read operation is initiated from the selected input device. The CPU does not necessarily enter the stopped state during the execution of the reset operation. The load indicator is on while the CPU is in the load state.

The read operation is performed as if a START I/O instruction were executed that specified the channel, subchannel, and I/O device designated by the load-unit-address controls. The operation uses an implied channel-address word (CAW) containing a subchannel key of zero, and a channel-command-word (CCW) address of 0, but the CAW location in storage, location 72, is not accessed. The load-unit-address controls provide the 12 rightmost bits of the I/O address; zeros are implied for the leftmost bits.

Although the location of the first CCW to be executed is specified by the CCW address as 0, the first CCW actually executed is an implied CCW, containing, in effect, a read command with the modifier bits set to zeros, a data address of 0, a byte count of 24, the chain-command flag set to one, the SLI flag set to one, the chain-data flag set to zero, the skip flag set to zero, and the PCI flag set to zero. The CCW fetched, as a result of command chaining, from storage location 8 or 16, as well as any subsequent CCW in the IPL sequence, is interpreted the same as a CCW in any I/O operation, except that any PCI flags that are specified in CCWs used for the IPL sequence are ignored.

When the I/O device provides channel-end status for the last operation of the IPL chain and no exceptional conditions are detected in the operation, a new PSW is obtained from storage locations 0-7. When this PSW specifies the EC mode, the I/O address that was used for the IPL operation is stored at locations 186-187, and zeros are stored at location 185; when the BC mode is specified, the I/O address is stored at locations 2-3. The CPU leaves the load state and enters the operating state, with CPU operation proceeding under the control of the new PSW, provided the rate control is set to process; if the rate control is set to instruction step, the CPU enters the stopped state after the new PSW has been obtained.

When channel-end status for the IPL operation is presented, either separate from or along with device-end status, no I/O-interruption condition is generated. Similarly, any PCI flags specified by the program in the CCWs used for the IPL sequence are ignored. If the device-end status for the IPL operation is provided separately after channel-end

status, it causes an I/O interruption condition to be generated.

If the IPL I/O operation or the PSW loading is not completed satisfactorily, the CPU remains in the load state, and the load indicator remains on. This occurs when the device designated by the load-unit-address controls is not operational, when the device or channel signals any condition other than channel end, device end, or status modifier during or at the completion of the IPL I/O operation, or when the PSW loaded from location 0 has a PSW-format error that is recognized during the loading procedure. The address of the I/O device used in the IPL operation is not stored. The contents of storage locations 0-7 are unpredictable. The contents of other storage locations remain unchanged, except possibly for those locations due to be changed by the read operations.

When fewer than eight bytes are read into locations 0-7, the PSW fetched from location 0 at the conclusion of the IPL operation is unpredictable.

Programming Notes

1. The information read and placed at locations 8-15 and 16-23 may be used as CCWs for reading additional information during the IPL sequence: the CCW at location 8 may specify reading additional CCWs elsewhere in storage, and the CCW at location 16 may specify the transfer-in-channel command, causing transfer to these CCWs.

   The status-modifier bit has its normal effect during the IPL operation, causing the channel to fetch and chain to the CCW whose address is 16 higher than that of the current CCW. This applies also to the initial chaining that occurs after completion of the read operation specified by the implicit CCW.

   The PSW that is loaded at the completion of the IPL procedure may be provided by the first eight bytes of the IPL I/O operation or may be placed at locations 0-7 by a subsequent CCW.
2. When the PSW in location 0 has bit 14 set to one, the CPU is placed in the wait state after the IPL procedure is completed; at that point, the load and manual indicators are off, and the wait indicator is on.
3. Activating the load-normal key permits an IPL program to be loaded with a minimum disturbance of storage contents. This function may be useful in debugging. When the power is turned on or the load-clear key is activated, the IPL program starts with a cleared machine in a known state, except that information on external storage remains unchanged.

Machine Save

The machine-save operation saves the current CPU status and the status and contents of storage page 0 for subsequent retrieval by programming. The operation is initiated manually by the machine-save key (see Chapter 13, "Operator Facilities"). The saved information may be retrieved by issuing a RETRIEVE STATUS AND PAGE instruction.

Machine save causes the following actions:

1. The current contents of all CPU registers and the status of page 0 are saved in internal storage. The format of the saved information is not defined. The figure "Machine Status, Retrieval Format" describes the machine-status information in the 256-byte format in which it is moved to addressable storage by a subsequent RETRIEVE STATUS AND PAGE instruction.
2. The current contents of page 0, that is, the 2,048 bytes at addresses 0-2047, are saved in internal storage.

The register contents and the status and contents of page 0 remain unchanged.

When a machine-save operation has been successfully completed, the save indicator is turned on.

A machine save replaces the information saved by the previous machine save.

When a clear-reset operation is performed, any previously saved information becomes invalid. Subsequent execution of the RETRIEVE STATUS AND PAGE instruction returns condition code 3 until another machine-save operation is successfully performed.

A reset operation occurring while a machine save is in progress halts the machine-save operation. If an incomplete machine save partially alters

Byte Offset	Bits	Contents
0-7		CPU timer[1]
8-15		Clock comparator[1]
16-23		Program-status word
24-31		Time-of-day clock[1]
32-63		Floating-point registers 0, 2, 4, 6
64-127		General registers 0-15
128-191		Control registers 0-15
192-199		CPU ID[1]
200-203		Page-capacity count[2]
204-207		Existing-frame-capacity count[2]
208-211		Available-frame-capacity count[2]
212-215		Free-frame-capacity count[2]
216	0	Zero
	1-3	Page bits of page 0
	4	Zero
	5	Reference bit of page 0
	6	Change bit of page 0
	7	Zero
217	0-3	Access-control bits of page 0
	4	Fetch-protection bit of page 0
	5-7	Zeros
218-219		Frame index[2] of page 0
220-255		Zeros

Explanation:

1 The formats of these fields are the same as those produced by STORE CPU TIMER, STORE CLOCK COMPARATOR, STORE CLOCK, and STORE CPU ID, respectively.

2 The capacity counts and the frame index are right-aligned with leftmost bits of zeros.

Machine Status, Retrieval Format

previously saved information, the saved information is indicated to be invalid, and subsequent execution of RETRIEVE STATUS AND PAGE returns condition code 3 until the next successful machine-save operation. Invalid machine saves cannot be retrieved.

The CPU must be in the stopped state before a machine-save operation can be initiated. If an error is encountered during the operation, the saved information becomes invalid, the CPU enters the check-stop state, and the save indicator is not turned on.

Programming Notes

1. Machine save may be used as part of a machine-dump procedure when the normal supervisor program is not functioning properly, such as after a hard wait (wait state with interruptions disabled). By preserving the complete machine status and page 0, machine save permits loading a dump program, which can preserve additional pages if necessary. The dump program can then merge the saved information with the undisturbed pages to create a complete image of the machine at the time of the machine save. The machine should not be cleared before loading the dump program.
2. When the supervisor program is still functioning, it is less disruptive to use the supervisor to invoke a dump program without a machine save. An intermediate option is the restart function.
3. The format of the byte at offset 216 corresponds to the byte inserted by the instruction INSERT PAGE BITS.
4. Unassigned bits in the retrieval format of the machine status are stored as zeros. The program should not depend on such zeros, however, to ensure that existing programs run if new facilities using these bits are defined.

Chapter 5. Program Execution

Contents

Normally, operation of the CPU is controlled by instructions in storage that are executed sequentially, one at a time, left to right in an ascending sequence of storage addresses. A change in the sequential operation may be caused by branching, LOAD PSW, interruptions, or manual intervention.

Instructions

Each instruction consists of two major parts:

- An operation code (op code), which specifies the operation to be performed, and
- The designation of the operands that participate

Operands

Operands can be grouped in three classes: operands located in registers, immediate operands, and operands in storage. Operands may be either explicitly or implicitly designated.

Register operands can be located in general, floating-point, or control registers, with the type of register identified by the op code. The register containing the operand is specified by identifying the register in a four-bit field, called the R field, in the instruction. For some instructions, an operand is located in an implicitly designated register, the register being implied by the op code.

Immediate operands are contained within the instruction, and the eight-bit field containing the immediate operand is called the I field.

Operands in storage may either have an implied length, be specified by a bit mask, or, in other cases, be specified by a four-bit or eight-bit length specification, called the L field, in the instruction. The addresses of operands in storage are specified by means of a format that uses the contents of a general register as part of the address. This makes it possible to:

1. Specify a complete address by using an abbreviated notation

2. Perform address manipulation using instructions which employ general registers for operands
3. Modify addresses by program means without alteration of the instruction stream
4. Operate independently of the location of data areas by directly using addresses received from other programs

The address used to refer to storage either is contained in a register designated by the R field in the instruction or is calculated from a base address, index, and displacement, designated by the B, X, and D fields, respectively, in the instruction.

For purposes of describing the execution of instructions, operands are designated as first and second operands and, in some cases, third operands.

In general, two operands participate in an instruction execution, and the result replaces the first operand. An exception is instructions with "store" in the instruction name, other than STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK, where the result replaces the second operand. Except when otherwise stated, the contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

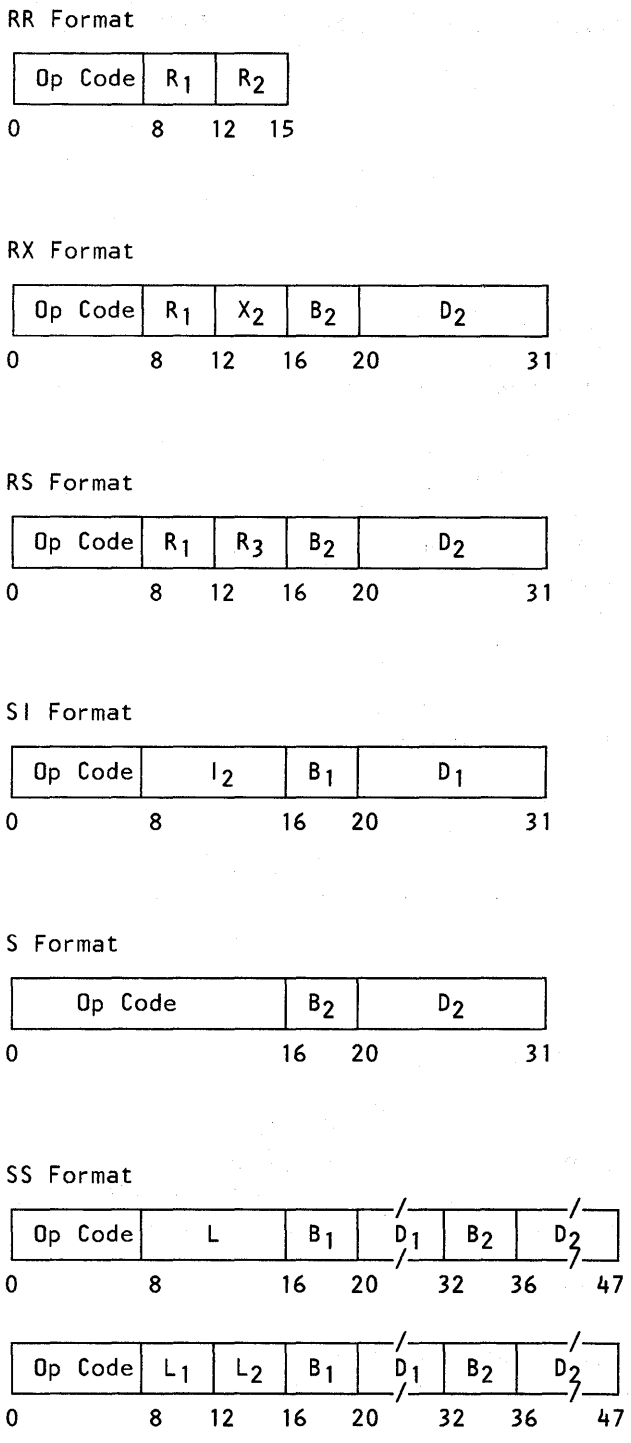
Instruction Format

An instruction is one, two, or three halfwords in length and must be located in storage on a halfword boundary. Each instruction is in one of six basic formats: RR, RX, RS, SI, S, and SS, with two variations of SS. (See the figure "Basic Instruction Formats.")

Some instructions contain fields that vary slightly from the basic format, and in some instructions the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

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

- RR denotes a register-and-register operation.
- RX denotes a register-and-indexed-storage operation.
- RS denotes a register-and-storage operation.
- SI denotes a storage-and-immediate operation.
- S denotes an operation using an implied operand and storage.
- SS denotes a storage-and-storage operation.



Basic Instruction Formats

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

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

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

In the format illustration for each individual instruction description, the op-code field shows the op code as hexadecimal digits within single quotes. The hexadecimal representation uses 0-9 for the codes 0000-1001 and A-F for the codes 1010-1111.

The remaining fields in the format illustration for each instruction are designated by code names, consisting of a letter and possibly a subscript number. The subscript number denotes the operand to which the field applies.

Register Operands
In the RR, RX, and RS formats, the contents of the register designated by the R_1 field are called the first operand. The register containing the first operand is sometimes referred to as the "first-operand location." In the RR format, the R_2 field designates the register containing the second operand, and the same register may be designated for the first and second operand. In the RS format, the use of the R_3 field depends on the instruction.

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

Unless otherwise indicated in the individual instruction description, the register operand is one register in length (32 bits for a general register or a control register and 64 bits for a floating-point register), and the second operand is the same length as the first.

Immediate Operands
In the SI format, the contents of the eight-bit immediate-data field, the I_2 field of the instruction, are used directly as the second operand. The B_1 and D_1 fields designate the first operand, which is one byte in length.

Storage Operands
In the SI and SS formats, the contents of the general register designated by the B_1 field are added to the contents of the D_1 field to form the first-operand address. In the S, RS, and SS formats, the contents of the general register designated by the B_2 field are added to the contents of the D_2 field to form the second-operand address. In the RX format, the contents of the general registers designated by the X_2 and B_2 fields are added to the contents of the D_2 field to form the second-operand address.

In the SS format, with two length fields given, L_1 specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-16, corresponding to a length code in L_1 of 0-15. Similarly, L_2 specifies the number of additional operand bytes to the right of the location designated by the second-operand address. Results replace the first operand, and are never stored outside the field specified by the address and length. If the first operand is longer than the second, the second operand is extended on the left with zeros up to the length of the first operand. This extension does not modify the second operand in storage.

In the SS format with a single, eight-bit length field, L specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-256, corresponding to a length code in L of 0-255. Storage results replace the first operand and are never stored outside the field specified by the address and length. In this format, the second operand has the same length as the first operand, except for the following instructions: EDIT, EDIT AND MARK, TRANSLATE, and TRANSLATE AND TEST. RETRIEVE STATUS AND PAGE does not use the L field, the operand lengths being fixed.

Address Generation

Execution of instructions by the CPU involves generation of the addresses of instructions and operands.

Sequential Instruction-Address Generation

When an instruction is fetched from the location designated by the current PSW, the instruction address is increased by the number of bytes in the instruction, and the instruction is executed. The same steps are then repeated using the new value

of the instruction address to fetch the next instruction in the sequence.

Instruction addresses wrap around, with the halfword at location $2^{24} - 2$ being followed by the halfword at location 0. Thus, any carry out of PSW bit position 40, as a result of updating the instruction address, is lost.

Operand-Address Generation

An operand address that refers to storage either is contained in a register designated by an R field in the instruction or is calculated from the sum of three binary numbers: base address, index, and displacement.

The base address (B) is a 24-bit number contained in a general register specified by the program in a four-bit field, called the B field, in the instruction. Base addresses can be used as a means of independently addressing each program and data area. In array-type calculations, it can specify the location of an array, and, in record-type processing, it can identify the record. The base address provides for addressing the entire storage. The base address may also be used for indexing.

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

The displacement (D) is a 12-bit number contained in a field, called the D field, in the instruction. The displacement provides for relative addressing of up to 4,095 bytes beyond the location designated by the base address. In array-type calculations, the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

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

A zero in any of the B_1, B_2, or X_2 fields indicates the absence of the corresponding address component. For the absent component, a zero is used in forming the address, regardless of the contents of general register 0. A displacement of zero has no special significance.

When an instruction description specifies that the contents of a general register designated by an R field are used to address an operand in storage, bit positions 8-31 of the register provide the operand address.

An instruction can designate the same general register both for address computation and as the location of an operand. Address computation is completed prior to the execution of the operation.

Unless otherwise indicated in an individual instruction definition, the generated operand address designates the leftmost byte of an operand in storage.

Programming Note

Negative values may be used in index and base-address registers. Bits 0-7 of these values are always ignored.

Branch-Address Generation

For branch instructions, the address of the next instruction to be executed when the branch is taken is called the branch address. Depending on the branch instruction, the instruction format may be RR, RS, or RX.

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

In the RR format, the contents of bit positions 8-31 of the general register designated by the R_2 field are used as the branch address. General register 0 cannot be designated as containing a branch address. A value of zero in the R_2 field causes the instruction to be executed without branching.

For several branch instructions, branching depends on satisfying a specified condition. When the condition is not satisfied, the branch is not taken, normal sequential instruction execution continues, and the branch address is not used. When a branch is taken, bits 8-31 of the generated branch address replace bits 40-63 of the current PSW. The branch address is not used to address storage as part of the branch operation.

A specification exception due to an odd branch address and access exceptions due to fetching of the instruction at the branch location are not recognized as part of the branch operation but

instead are recognized as exceptions associated with the execution of the instruction at the branch location.

A branch instruction, such as BRANCH AND LINK, can designate the same general register for branch-address computation and as the location of an operand. Branch-address computation is completed before the remainder of the operation is executed.

Instruction Execution and Sequencing

The program-status word (PSW), described in Chapter 4, "Control," contains information required for proper program execution. The PSW is used to control instruction sequencing and to hold and indicate the status of the machine in relation to the program currently being executed. The active or controlling PSW is called the current PSW.

Branch instructions perform the functions of decision-making, loop control, and subroutine linkage. A branch instruction affects instruction sequencing by introducing a new instruction address into the current PSW.

Facilities for decision making are provided by the BRANCH ON CONDITION instruction. This instruction inspects a condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. The condition code, which consists of two bits, provides for four possible condition-code settings: 0, 1, 2, and 3.

The specific meaning of any setting depends on the operation that sets the condition code. For example, the condition code reflects such conditions as zero, nonzero, first operand high, equal, overflow, and channel busy. Once set, the condition code remains unchanged until modified by an instruction that causes a different condition code to be set. See Appendix C, "Condition-Code Settings," for a summary of the instructions which set the condition code.

Loop control can be performed by the use of BRANCH ON CONDITION to test the outcome of address arithmetic and counting operations. For some particularly frequent combinations of arithmetic and tests, the instructions BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW are provided. These branches, being specialized, provide increased performance for these tasks.

Subroutine linkage is provided by the BRANCH AND LINK instructions, which permit not only the introduction of a new instruction address but also the preservation of the return address and associated information. Subroutine linkage between a program and the supervisor program is provided by means of the SUPERVISOR CALL instruction.

Interruptions

Interruptions permit the CPU to change state as a result of conditions external to the system, in input/output (I/O) devices, or in the CPU itself. Details are to be found in Chapter 6, "Interruptions."

Six classes of interruption conditions are possible: external, I/O, machine check, program, restart, and supervisor call. Each class has two related PSWs, called old and new, in permanently assigned storage locations. In all classes, an interruption involves storing information identifying the cause of the interruption, storing the current PSW at the old-PSW position, and fetching the PSW at the new-PSW position, which becomes the current PSW.

The old PSW contains CPU-status information necessary for resumption of the interrupted program. At the conclusion of the program invoked by the interruption, the instruction LOAD PSW may be used to restore the current PSW to the value of the old PSW.

Types of Instruction Ending

Instruction execution ends in one of five ways: completion, nullification, suppression, termination, and partial completion.

Completion of instruction execution provides results as called for in the definition of the instruction. When an interruption occurs after the completion of the execution of an instruction, the instruction address in the old PSW designates the next instruction to be executed.

Suppression of instruction execution causes the instruction to be executed as if it specified "no operation." The contents of any result fields, including the condition code, are not changed. The instruction address in the old PSW on an interruption after suppression designates the next sequential instruction.

Nullification of instruction execution has the same effect as suppression, except that when an interruption occurs after the execution of an instruction has been nullified, the instruction

address in the old PSW designates the instruction whose execution was nullified instead of the next sequential instruction.

Termination of instruction execution causes the contents of any fields due to be changed by the instruction to be unpredictable. The operation may have replaced all, part, or none of the contents of the designated result fields and may have changed the condition code if such change was called for by the instruction. Unless the interruption is caused by a machine-check condition, the validity of the instruction address in the PSW, the interruption code, and the ILC are not affected, and the state or the operation of the machine has not been affected in any other way. The instruction address in the old PSW on an interruption after termination designates the next sequential instruction.

Partial completion of instruction execution occurs only for interruptible instructions; it is described in the next section.

Interruptible Instructions

Point of Interruption

For most instructions, the entire execution of an instruction is one operation. An interruption is permitted between operations; that is, an interruption can occur after the performance of one operation and before the start of a subsequent operation.

For the following instructions, referred to as interruptible instructions, an interruption is permitted after partial completion of the instruction:

COMPARE LOGICAL LONG
MOVE LONG

The execution of an interruptible instruction is considered to consist of a number of units of operation, and an interruption is permitted between units of operation. The amount of data processed in a unit of operation depends on the particular instruction and may depend on the model and on the particular condition that causes the execution of the instruction to be interrupted.

Whenever points of interruption that include those occurring within the execution of an interruptible instruction are discussed, the term "unit of operation" is used. For a noninterruptible instruction, the entire execution consists, in effect, of one unit of operation.

Execution of Interruptible Instructions

The execution of an interruptible instruction is completed when all units of operation associated with that instruction are completed. When an interruption occurs after completion, nullification, or suppression of a unit of operation, all prior units of operation have been completed.

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

When a unit of operation is suppressed, the instruction address in the old PSW designates the next sequential instruction. The operand parameters, however, are adjusted so as to indicate the extent to which instruction execution has been completed. If the instruction is reexecuted after the conditions causing the suppression have been removed, the execution is resumed from the point of interruption. As in the case of completion and nullification, it depends on the instruction how the operand parameters are adjusted.

When an exception which causes termination occurs as part of a unit of operation of an interruptible instruction, the entire operation is terminated, and the contents, in general, of any fields due to be changed by the instruction are unpredictable. On such an interruption, the instruction address in the old PSW designates the next sequential instruction.

Programming Notes

1. Any interruption, other than supervisor call and some program interruptions, can occur after a partial execution of an interruptible instruction. In particular, interruptions for external, I/O, machine-check, restart, and program interruptions for access exceptions and PER events can occur between units of operation.
2. The amount of data processed in a unit of operation of an interruptible instruction depends on the model and may depend on the type of condition which causes the execution of the instruction to be interrupted or stopped. Thus, when an interruption occurs at the end of the current unit of operation, the length of the unit of operation may be different for different types of interruptions. Also, when the stop function is requested during the execution of an interruptible instruction, the CPU enters the

stopped state at the completion of the execution of the current unit of operation. Similarly, in the instruction-step mode, only a single unit of operation is performed, but the unit of operation for the various cases of stopping may be different.

Exceptions to Nullification and Suppression

In certain unusual situations, the result fields of an instruction having a store-type operand are changed in spite of the occurrence of an exception which would normally result in nullification or suppression. These situations are exceptions to the general rule that the operation is treated as a no-operation when an exception requiring nullification or suppression is recognized. Each of these situations may result in the turning on of the change bit associated with the store-type operand, even though the final result in storage may appear unchanged. Depending on the particular situation, additional effects may be observable, the extent of which is described for each of the situations.

All of these situations are limited to the extent that a store access does not occur and the change bit is not set when the store access is prohibited. For the CPU, a store access is prohibited whenever an access exception exists for that access, or whenever an exception exists which is of higher priority than the priority of an access exception for that access.

When, in these situations, an interruption for an exception requiring suppression occurs, the instruction address in the old PSW designates the next sequential instruction. When an interruption for an exception requiring nullification occurs, the instruction address in the old PSW designates the instruction causing the exception even though partial results may have been stored.

Storage Change and Restoration for Page-Access Exceptions

For page-access exceptions, on some systems, a channel may observe the effects on storage described in the following case.

When, for an instruction having a store-type operand, a page-access exception is recognized for any operand of the instruction, that portion, if any, of the store-type operand which would not cause an exception may change to an intermediate value and then back to the original value.

The accesses associated with storage change and restoration for page-access exceptions are only observable by a channel. Except for multiple-access operands, the intermediate value, if any, is always equal to what would have been the final value if the page-access exception had not occurred.

Programming Notes

1. Storage change and restoration for page-access exceptions occur in two main situations:
   a. The exception is recognized for a portion of a store-type operand which crosses a page boundary, and the other portion has no access exception.
   b. The exception is recognized for one operand of an instruction having two storage operands (for example, an SS-format instruction or MOVE LONG), and the other operand, which is a store-type operand, has no access exception.
2. To avoid letting the channel observe intermediate operand values due to storage change and restoration for page-access exceptions (especially when a CCW chain is modified), either one storage page should be operated on at a time or preliminary testing should be performed to ensure that all required pages are addressable.

Trial Execution for TRANSLATE and EDIT

For the instructions TRANSLATE (TR), EDIT (ED), and EDIT AND MARK (EDMK), the portions of the operands that are actually used in the operation may be established in a trial execution for operand accessibility that is performed before the execution of the instruction is started. This trial execution consists in an execution of the instruction in which results are not stored. If the first operand of TR or either operand of ED or EDMK is changed by an I/O operation after the initial trial execution but before completion of execution, the contents of any fields due to be changed by the instruction are unpredictable. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable.

Update for Suppression

When, for an instruction with a store-type operand, an exception is recognized whose priority is equal to or lower than an access exception for some portion of the store-type operand, an update which does not change the contents of the location may occur for that portion of the store-type operand.

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

Programming Note
Examples of when an update may occur to the destination-operand location in storage are:
- Decimal-divide exception for DIVIDE DECIMAL
- Specification exception for an odd register number for COMPARE DOUBLE AND SWAP
- Data exception for an invalid decimal sign for ADD DECIMAL

Sequence of Storage References

Conceptually, the CPU processes instructions one at a time, with the execution of one instruction preceding the execution of the following instruction. The execution of the instruction specified by a successful branch follows the execution of the branch. Similarly, an interruption takes place between instructions or, for interruptible instructions, between units of operation of such instructions.

The sequence of events implied by the processing just described is sometimes called the conceptual sequence.

Each operation appears to the program to be performed sequentially, with the current instruction being fetched after the preceding operation is completed and before the execution of the current operation is begun. This appearance is maintained, even though the storage-implementation characteristics and overlap of instruction execution with storage accessing may cause actual processing to be different. The results generated are those that would have been obtained had the operations been performed in the conceptual sequence. Thus, it is possible for an instruction to modify the next succeeding instruction in storage.

In simple models in which operations are not overlapped, the conceptual and actual sequences are essentially the same. However, in more complex machines, overlapped operation, buffering of operands and results, and execution times which are comparable to the propagation delays between units can cause the actual sequence to differ considerably from the conceptual sequence. In these machines, special circuitry is employed to detect dependencies between operations and ensure that the results obtained are those that would have been obtained if the operations had been performed in the conceptual sequence. However, channels may, unless otherwise constrained, observe a sequence that differs from the conceptual sequence.

It can normally be assumed that the execution of each instruction occurs as an indivisible event. However, in actual operation, the execution of an instruction consists of a series of discrete steps. Depending on the instruction, operands may be fetched and stored in a piecemeal fashion, and some delay may occur between fetching operands and storing results. As a consequence, a channel may be able to observe intermediate or partially completed results.

When the program on the CPU interacts with a program on a channel, the programs may have to take into consideration that a single operation may consist of a series of storage references, that a storage reference may in turn consist of a series of accesses, and that the conceptual and actual sequences of these accesses may differ. Storage references associated with instruction execution are of the following types: instruction fetches and storage-operand references. For the purposes of the following discussion, page-description accesses are also considered to be storage references.

Programming Note
The sequence of execution may differ from the simple conceptual definition in the following ways.
- As viewed by a program in a channel, the execution of an instruction may appear to be performed as a sequence of piecemeal steps. This is described for each type of storage reference in one of the following sections.
- As viewed by a program in a channel, the storage-operand accesses associated with one instruction are not necessarily performed in the conceptual sequence. (See the section "Relation Between Operand Accesses" in this chapter.)
- As viewed by a program in a channel, in certain unusual situations, the contents of storage may appear to change and then be restored to the original value. (See the section "Storage Change and Restoration for Page Access Exceptions" earlier in this chapter.)

Instruction Fetching

Instruction fetching consists in fetching the one, two, or three halfwords specified by the instruction address in the current PSW. The immediate field of an instruction is accessed as part of an instruction fetch. If, however, an instruction

specifies a storage operand at the location occupied by the instruction itself, the location is accessed both as an instruction and as a storage operand. The fetch of the target instruction of EXECUTE is considered to be an instruction fetch.

The bytes of an instruction may be fetched piecemeal and are not necessarily accessed in a left-to-right direction. The instruction may be fetched multiple times for a single execution; for example, it may be fetched for testing the addressability of operands or for inspection of PER events, and it may be refetched for actual execution.

Instructions are not necessarily fetched in the sequence in which they are conceptually executed and are not necessarily fetched for each time they are executed. In particular, the fetching of an instruction may precede the storage-operand references for an instruction that is conceptually earlier. The instruction fetch occurs prior to all storage-operand references for all instructions that are conceptually later.

There is no limit established as to the number of instructions which may be prefetched, and multiple copies of the contents of a single storage location may be fetched. As a result, the instruction executed is not necessarily the most recently fetched copy. Storing caused by channels does not necessarily change the copy of prefetched instructions. However, if a store that is conceptually earlier occurs on the CPU and modifies the location from which the instruction is subsequently fetched, the updated information is obtained.

All copies of prefetched instructions are discarded when:

- A serializing function is performed
- The CPU enters the operating state

Programming Note
When a channel modifies an instruction, it is possible for the CPU to recognize the changes to some but not all modified bit positions of the instruction.

Page-Description Accesses

References to the page description are handled as follows:

1. Whenever a reference to storage is made and key-controlled protection applies to the reference, the four access-control bits and the fetch-protection bit associated with the storage location are inspected concurrently with the reference to the storage location.
2. When storing is performed, the change bit is set in the associated storage key concurrently with the store operation.
3. The instruction SET STORAGE KEY causes all seven bits to be set concurrently in the storage key. The access to the storage key for SET STORAGE KEY follows the sequence rules for storage-operand store references and is a single-access reference.
4. The instruction INSERT STORAGE KEY provides a consistent image of the field, which consists of all seven bits of the storage key. The access to the storage key for INSERT STORAGE KEY follows the sequence rules for storage-operand fetch references and is a single-access reference.
5. The instruction RESET REFERENCE BIT modifies only the reference bit. All other bits of the storage key remain unchanged. The reference bit and change bit are examined concurrently to set the condition code. The access to the storage key for RESET REFERENCE BIT follows the sequence rules for storage-operand update references. The reference bit is the only bit which is updated.
6. The instruction SET PAGE BITS provides a consistent image of the change bit. The instruction modifies both the reference and change bits, and the three programmable page bits. The page bits are only accessible by the CPU. The access to the change bit follows the sequence rules for storage-operand update references, with the following exception: if the change bit is being set to zero, no storing in the associated storage page by a channel is permitted between the fetching of the change bit and the setting of the change bit to zero.
7. The instruction INSERT PAGE BITS inspects but does not modify the reference, change, and page bits. The page bits are only accessible by the CPU. The access to the reference, change, and page bits follows the sequence rules for storage-operand fetch references and is a single-access reference.
8. Whenever a reference to storage is made and page-state checking applies to the reference, the page state and frame index associated with the storage location must appear to be inspected concurrently with the reference to the storage location.
9. The instruction CONNECT PAGE causes the page state and frame index to be set concurrently in the page description, with the

access to the page state and frame index following the sequence rules for storage-operand store references.

10. During the execution of the instructions DECONFIGURE PAGE and DISCONNECT PAGE, the accesses to set the reference bit and the change bit to zeros occur concurrently with or after the access to set the page state to disconnected.
11. The instructions MAKE ADDRESSABLE and MAKE UNADDRESSABLE modify only the page state.
12. The instruction LOAD FRAME INDEX inspects but does not modify the page state and frame index. The page state and frame index may only be modified explicitly by other instructions.

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

In certain situations, the change bit may be set when no storing has actually taken place.

Storage-Operand References

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

During the execution of an instruction, all or some of the storage operands for that instruction may be fetched, intermediate results may be maintained for subsequent modification, and final results may be temporarily held prior to placing them in storage. Stores caused by channels do not necessarily affect these intermediate results. Storage-operand references are of three types: fetches, stores, and updates.

Storage-Operand Fetch References

When the bytes of a storage operand participate in the instruction execution only as a source, the operand is called a fetch-type operand, and the reference to the location is called a storage-operand fetch reference. A fetch-type operand is identified in individual instruction definitions by indicating that the access exception is for fetch.

All bits within a single byte of a fetch reference are accessed concurrently. When an operand consists of more than one byte, the bytes may be fetched from storage piecemeal, one byte at a time. Unless otherwise specified, the bytes are not necessarily fetched in any particular sequence.

Storage-Operand Store References

When the bytes of a storage operand participate in the instruction execution only as a destination, to the extent of being replaced by the result, the operand is called a store-type operand, and the reference to the location is called a storage-operand store reference. A store-type operand is identified in individual instruction definitions by indicating that the access exception is for store.

All bits within a single byte of a store reference are accessed concurrently. When an operand consists of more than one byte, the bytes may be placed in storage piecemeal, one byte at a time. Unless otherwise specified, the bytes are not necessarily stored in any particular sequence.

The CPU may delay storing results into storage. There is no defined limit on the length of time that results may remain pending before they are stored.

This delay does not affect the sequence in which results are placed in storage. The results of one instruction are placed in storage after the results of all preceding instructions have been placed in storage and before any results of the succeeding instructions are stored as observed by channels. The results of any one instruction are stored in the sequence specified for that instruction.

The CPU does not fetch operands from a storage location until all information destined for that location by the CPU has been stored. Prefetched instructions may appear to be updated before the information appears in storage.

The stores are necessarily completed only as a result of a serializing operation and before the CPU enters the stopped state.

Storage-Operand Update References

In some instructions, the storage-operand location participates both as a source and as a destination. In these cases, the reference to the location consists first of a fetch and subsequently of a store. Such an operand is called an update-type operand, and the combination of the two accesses is referred to as an update reference. Instructions such as MOVE ZONES, TRANSLATE, OR (OC, OI), and ADD DECIMAL cause an update to the first-operand location. No special interlock is provided between the fetch and store, and accesses by channels are permitted. An update-type operand is identified in the individual instruction definition by indicating that the access exception is for both fetch and store. The fetch and store accesses associated with an update reference do not necessarily occur one immediately after the other,

and it is possible for a channel to make one or more interleaved accesses to the same location. The interleaved accesses can be either fetches or stores.

Storage-Operand Consistency

Single-Access References

A fetch reference is said to be a single-access reference if the value is fetched in a single access to each byte of the data field. In the case of overlapping operands, the location may be accessed once for each operand. A store-type reference is said to be a single-access reference if a single store access occurs to each byte location within the data field. An update reference is said to be single-access if both the fetch and store accesses are each single-access.

Except for the accesses associated with multiple-access operands and the stores associated with storage change and restoration for page-access exceptions, storage-operand references are single-access references.

Multiple-Access Operands

For some instructions, multiple accesses may be made to all or some of the bytes of a storage operand. The following cases are those storage-operand references which may be multiple-access ones.

1. The storage references associated with the decimal operands of the following instructions are not necessarily single-access references: the decimal instructions and the instructions CONVERT TO BINARY, CONVERT TO DECIMAL, MOVE WITH OFFSET, PACK, and UNPACK.
2. The operands of MOVE INVERSE.
3. The stores into that portion of the first operand of MOVE LONG which is filled with padding bytes.

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

Programming Notes

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

Relation between Operand Accesses

Storage-operand fetches associated with one instruction execution must appear to precede all storage-operand references for conceptually subsequent instructions. A storage-operand store specified by one instruction must appear to precede all storage-operand stores specified by conceptually subsequent instructions, but it does not necessarily precede storage-operand fetches specified by conceptually subsequent instructions. However, a storage-operand store must precede a conceptually subsequent storage-operand fetch from the same main-storage location.

When an instruction has two storage operands both of which cause fetch references, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. When the two operands overlap, the common locations may be fetched independently for each operand.

When an instruction has two storage operands, the first of which causes a store and the second a fetch reference, it is unpredictable how much of the second operand is fetched before the results are stored. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

When an instruction has two storage operands, the first of which causes an update reference and the second a fetch reference, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. Similarly, it is unpredictable how much of the result is processed before it is returned to storage. In the case of destructively overlapping

operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

Programming Note
The independent fetching of a single location for each of two operands may affect the program execution in the following situation.

When the same storage location is designated by two operand addresses of an instruction, and a channel causes the contents of the location to change during execution of the instruction, the old and new values of the location may be used simultaneously. For example, comparison of a field to itself may yield a result other than equal, or EXCLUSIVE-ORing of a field to itself may yield a result other than zero.

Other Storage References

Store accesses for interruption codes not stored within the old PSW are not necessarily single-access stores. The external and SVC interruption-code stores occur between the conceptually previous and conceptually subsequent operations. The program interruption-code store accesses may precede the storage-operand references associated with the instruction which results in the program interruption.

The CSW and I/O-communications-area stores occur within the conceptual limits of the interruption or I/O instruction with which they are associated.

Updating of the interval timer occurs after storage-operand references for the conceptually previous instruction and before storage-operand references for the conceptually subsequent instruction. Interval-timer updates can also occur within an interruptible instruction between units of operation.

Serialization

The sequence of functions performed by a CPU is normally independent of the functions performed by channels. Similarly, the sequence of functions performed by a channel is normally independent of the functions performed by other channels and by the CPU. However, at certain points in its execution, serialization of the CPU occurs. Serialization also occurs at certain points for channels.

CPU Serialization

All interruptions and the execution of certain instructions cause serialization of CPU operation. A serialization operation consists in completing all conceptually previous storage accesses by the CPU, as observed by channels, before the conceptually subsequent storage accesses occur. Serialization affects the sequence of all CPU accesses to storage and to the page descriptions.

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

1. The general instructions BRANCH ON CONDITION (BCR) with the M_1 and R_2 field containing all ones and all zeros, respectively, and COMPARE AND SWAP, COMPARE DOUBLE AND SWAP, STORE CLOCK, SUPERVISOR CALL, and TEST AND SET.
2. LOAD PSW and SET STORAGE KEY.
3. All I/O instructions.

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

- All conceptually previous storage accesses by the CPU are completed, as observed by channels. This includes all conceptually previous stores and changes to page descriptions.
- The normal function associated with the serializing operation is performed. In the case of instruction execution, operands are fetched, and the storing of results is completed. The exceptions are LOAD PSW, in which the operand may be fetched before previous stores have been completed, and interruptions, in which the interruption code and associated fields may be stored prior to the serialization. The fetching of the serializing instruction occurs before the execution of the instruction and may precede the execution of previous instructions, but may not precede the completion of the previous serializing operation. In the case of an interruption, the old PSW, the interruption code, and other information, if any, are stored, and the new PSW is fetched, but not necessarily in that sequence.
- Finally, instruction fetch and operand accesses for conceptually subsequent operations may begin.

A serializing function affects the sequence of storage accesses that are under the control of the CPU. It does not affect the sequence of storage accesses under the control of a channel.

Programming Notes

1. When a serializing operation takes place, channels observe instruction and operand fetching and result storing to take place in the sequence established by the serializing operation.

   Storing by a channel into a location from which a serializing instruction is fetched does not necessarily affect the execution of the serializing instruction unless a serializing operation has been performed after the storing and before the execution of the serializing instruction.
2. For programs that are intended to run also on multiprocessing configurations of System/370, it should be noted that the serializing operations affect the sequence of CPU accesses to storage and to the storage key, as observed by other CPUs as well as by channels. Therefore, serializing instructions should be inserted wherever it is necessary to control the interaction of programs that may run concurrently on different CPUs.

Channel Serialization

Serialization of a channel occurs as follows:

1. For a single channel program, all storage accesses and page-description accesses by the channel follow the execution of START I/O or START I/O FAST RELEASE, as observed by the CPU and other channels. This includes all accesses for the CAW, CCWs, and data.
2. For the last CCW of a chain, all storage accesses and page-description accesses are completed, as observed by the CPU and other channels, before the interruption condition indicating channel end is presented to the CPU.
3. If a CCW in the chain contains a PCI bit which is one, all storage accesses and page-description accesses due to CCWs preceding it in the chain are completed, as observed by the CPU and other channels, before the PCI condition is presented to the CPU.

The serialization of a channel does not affect the sequence of storage accesses or page-description accesses caused by a program in the CPU or another channel. It also does not affect the sequence of storage accesses or page-description accesses caused by other channel programs on the same channel.

Chapter 6. Interruptions

Contents

The interruption facility permits the CPU to change its state as a result of conditions external to the system, within the system, or within the CPU itself. To permit fast response to conditions of high priority and immediate recognition of the type of condition, interruption conditions are grouped into six classes: external, input/output, machine check, program, restart, and supervisor call.

Interruption Action

An interruption consists in storing the current PSW as an old PSW, storing information identifying the cause of the interruption, and fetching a new PSW. Processing resumes as specified by the new PSW.

The old PSW stored on an interruption normally contains the address of the instruction that would have been executed next had the interruption not occurred, thus permitting resumption of the interrupted program. For program and supervisor-call interruptions, the information stored also contains a code that identifies the length of the last-executed instruction, thus permitting the program to respond to the cause of the interruption. In the case of some program conditions for which the normal response is reexecution of the instruction causing the interruption, the instruction address directly identifies the instruction last executed.

Except for restart, an interruption can take place only when the CPU is in the operating state. The restart interruption can occur with the CPU in either the stopped or operating state.

The details of source identification, location determination, and instruction execution are explained in later sections and are summarized in the figure "Interruption Action."

Source Identification	Interruption Code	PSW-Mask Bits EC	PSW-Mask Bits BC	Mask Bits in Ctrl Registers Reg, Bit	ILC Set	Execution of Instruction Identified by Old PSW
MACHINE CHECK (old PSW 48, new PSW 112)	Locations 232-239[1]					
Exigent condition		13	13		x	terminated
Repressible cond		13	13	14, 4-7	x	unaffected[2]
SUPERVISOR CALL (old PSW 32, new PSW 96)	Locations 138-139 in EC mode and 34-35 in BC mode					
Instruction bits	00000000 ssssssss				1,2	completed
PROGRAM (old PSW 40, new PSW 104)	Locations 142-143 in EC mode and 42-43 in BC mode					
Operation	00000000 p0000001				1,2,3	suppressed
Privileged oper	00000000 p0000010				1,2	suppressed
Execute	00000000 p0000011				2	suppressed
Protection	00000000 p0000100				1,2,3	suppressed or terminated
Addressing	00000000 p0000101				1,2,3	suppressed or terminated
Specification	00000000 p0000110				0,1,2,3	suppressed or completed
Data	00000000 p0000111				2,3	suppressed or terminated
Fixed-pt overflow	00000000 p0001000	20	36		1,2	completed
Fixed-point divide	00000000 p0001001				1,2	suppressed or completed
Decimal overflow	00000000 p0001010	21	37		2,3	completed
Decimal divide	00000000 p0001011				2,3	suppressed
Exponent overflow	00000000 p0001100				1,2	completed
Exponent underflow	00000000 p0001101	22	38		1,2	completed
Significance	00000000 p0001110	23	39		1,2	completed
Floating-pt divide	00000000 p0001111				1,2	suppressed
Special operation	00000000 p0010011			0, 1	2	suppressed
Page access	00000000 p0011000				1,2,3	nullified
Page state	00000000 p0011010				2	suppressed
Page transition	00000000 p0011011				2	suppressed
Monitor event	00000000 p1000000			8, 16+	2	completed
PER event	00000000 1n0nnnnn[3]	1	*	9, 0-3	0,1,2,3	completed[4]

Interruption Action (Part 1 of 2)

Source Identification	Interruption Code	PSW-Mask Bits EC	PSW-Mask Bits BC	Mask Bits in Ctrl Registers Reg, Bit	ILC Set	Execution of Instruction Identified by Old PSW
EXTERNAL (old PSW 24, new PSW 88)	Locations 134-135 in EC mode and 26-27 in BC mode					
Interval timer	00000000 1eeeeeee	7	7	0, 24	x	unaffected
Interrupt key	00000000 e1eeeeee	7	7	0, 25	x	unaffected
External signal 2	00000000 ee1eeeee	7	7	0, 26	x	unaffected
External signal 3	00000000 eee1eeee	7	7	0, 26	x	unaffected
External signal 4	00000000 eeee1eee	7	7	0, 26	x	unaffected
External signal 5	00000000 eeeee1ee	7	7	0, 26	x	unaffected
External signal 6	00000000 eeeeee1e	7	7	0, 26	x	unaffected
External signal 7	00000000 eeeeeee1	7	7	0, 26	x	unaffected
Clock comparator	00010000 00000100	7	7	0, 20	x	unaffected
CPU timer	00010000 00000101	7	7	0, 21	x	unaffected
INPUT/OUTPUT (old PSW 56, new PSW 120)	Locations 186-187 in EC mode and 58-59 in BC mode					
Channel 0	00000000 dddddddd	6	0	2, 0[5]	x	unaffected
Channel 1	00000001 dddddddd	6	1	2, 1[5]	x	unaffected
Channel 2	00000010 dddddddd	6	2	2, 2[5]	x	unaffected
Channel 3	00000011 dddddddd	6	3	2, 3[5]	x	unaffected
Channel 4	00000100 dddddddd	6	4	2, 4[5]	x	unaffected
Channel 5	00000101 dddddddd	6	5	2, 5[5]	x	unaffected
Channel 6 & up	cccccccc dddddddd	6	6	2, 6+	x	unaffected
RESTART (old PSW 8, new PSW 0)	Locations 2-3 in BC mode					
Restart key	00000000 00000000[6]				x	unaffected

Explanation:

1 A model-independent machine-check interruption code of 64 bits is stored at locations 232-239.
2 The effect of the machine-check condition is identified by the validity bits in the machine-check interruption code. The instruction is unaffected only if all the associated validity bits are ones.
3 When the interruption code indicates a PER event, an ILC of 0 may be stored only when bits 8-15 of the interruption code are 10000110 (PER, specification).
4 The unit of operation is completed, unless a program exception concurrently indicated causes the unit of operation to be nullified, suppressed, or terminated.
5 For channels 0-5, channel masks in control register 2 have no effect in the BC mode.
6 Bits 16-31 in the old PSW in the BC mode are set to zeros. No interruption code is provided in the EC mode.
+ Plus the following bits in the control register.
* In the BC mode, program-event recording is disabled.
c Channel-address bits.
d Device-address bits.
e If one, the bit indicates another concurrent external-interruption condition.
n A possible nonzero code, indicating another concurrent program-interruption condition.
p If one, the bit indicates a concurrent PER-event interruption condition.
s Bits of the I field of SUPERVISOR CALL.
x Unpredictable in the BC mode; not stored in the EC mode.

Interruption Action (Part 2 of 2)

Source Identification

The six classes of interruptions (external, I/O, machine check, program, restart, and supervisor call) are distinguished by the storage locations at which the old PSW is stored and from which the new PSW is fetched. For most classes, the causes are further identified by an interruption code and, for some classes, by additional information placed in permanently assigned storage locations during the interruption. (See also the section "Assigned Storage Locations" in Chapter 3, "Storage.") For external, I/O, program, and supervisor-call interruptions, the interruption code consists of 16 bits.

For external interruptions in the EC mode, the interruption code is stored at locations 134-135. In the BC mode, the interruption code is placed in the old PSW.

For I/O interruptions in the EC mode, the interruption code, which contains the I/O address, is stored at locations 186-187. In the BC mode, the interruption code is placed in the old PSW. Additional information is provided by the contents of the channel-status word (CSW) stored at location 64. Further information may be provided by the limited channel logout stored at location 176.

For machine-check interruptions, the interruption code consists of 64 bits and is stored at locations 232-239. Additional information for identifying the cause of the interruption and for recovering the state of the machine may be provided by the contents of the machine-check save areas. (See Chapter 11, "Machine-Check Handling.")

For program interruptions in the EC mode, the interruption code is stored at locations 142-143, and the instruction-length code is stored in bit positions 5 and 6 of location 141. In the BC mode, the interruption code and instruction-length code are placed in the old PSW. Further information may be provided in the form of the access-exception address, monitor-class number, monitor code, PER code, and PER address, which are stored at locations 144-159.

For restart interruptions in the EC mode, no interruption code is stored. In the BC mode, an interruption code of zero is placed in the old PSW.

For supervisor-call interruptions in the EC mode, the interruption code is stored at locations 138-139, and the instruction-length code is stored in bit positions 5 and 6 of location 137. In the BC mode, the interruption code and instruction-length code are placed in the old PSW.

Enabling and Disabling

By means of mask bits in the current PSW and in control registers, the CPU may be enabled or disabled for all external, I/O, and machine-check interruptions and for some program interruptions. When a mask bit is one, the CPU is enabled for the corresponding class of interruptions, and these interruptions can take place.

When a mask bit is zero, the CPU is disabled for the corresponding interruptions. The conditions that cause I/O or external interruptions remain pending. Machine-check-interruption conditions, depending on the type, are ignored, remain pending, or cause the CPU to enter the check-stop state. The disallowed program-interruption conditions are ignored, except that some causes are indicated also by the setting of the condition code.

Program interruptions for which mask bits are not provided, as well as the supervisor-call and restart interruptions, are always taken.

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

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

Programming Notes

1. Mask bits in the PSW provide a means of disallowing all maskable interruptions; thus, subsequent interruptions can be disallowed by the new PSW introduced by an interruption. Furthermore, the mask bits can be used to establish a hierarchy of interruption priorities, where a condition in one class can interrupt the program handling a condition in another class but not vice versa. To prevent an interruption-handling routine from being interrupted before the necessary housekeeping steps are performed, the new PSW must disable the CPU for further interruptions within the same class or within a class of lower priority.

2. Since the mask bits in control registers are not changed as part of the interruption procedure, these masks cannot be used to prevent an interruption immediately after a previous interruption in the same class. The mask bits in control registers provide a means for selectively enabling the CPU for some sources and disabling it for others within the same class.

Instruction-Length Code

The instruction-length code (ILC) occupies two bit positions and provides the length of the last instruction executed. It permits identifying the instruction causing the interruption when the instruction address in the old PSW designates the next sequential instruction. The ILC is provided also by the BRANCH AND LINK instructions.

When the old PSW specifies the EC mode, the ILC for program and supervisor-call interruptions is stored in bit positions 5 and 6 of the bytes at locations 137 and 141, respectively. For external, I/O, machine-check, and restart interruptions, the ILC is not stored since it cannot be related to the length of the last-executed instruction.

When the old PSW specifies the BC mode, the ILC is stored in bit positions 32 and 33 of that PSW. The ILC is meaningful, however, only after a supervisor-call or program interruption. For machine-check, external, I/O, and restart interruptions, the ILC does not indicate the length of the last-executed instruction and is unpredictable. Similarly, the ILC is unpredictable in the PSW stored during execution of the machine-save function and when the PSW is displayed.

For supervisor-call and program interruptions, a nonzero ILC identifies in halfwords the length of the instruction that was last executed. Whenever an instruction is executed by means of EXECUTE, instruction-length code 2 is set to indicate the length of EXECUTE and not that of the target instruction.

The value of a nonzero instruction-length code is related to the leftmost two bits of the instruction. The value is not contingent on whether the operation code is assigned or on whether the instruction is installed. The following table summarizes the meaning of the instruction-length code:

ILC		Instr Bits	Instruction Length
Decimal	Binary	0-1	
0	00		Not available
1	01	00	One halfword
2	10	01	Two halfwords
2	10	10	Two halfwords
3	11	11	Three halfwords

Zero ILC

Instruction-length code 0, after a program interruption, indicates that the location of the instruction causing the interruption is not made available to the program.

An ILC of 0 occurs when a specification exception is recognized that is due to a PSW-format error, other than one due to an odd instruction address, and the invalid PSW has been introduced by LOAD PSW or an interruption. (See the section "Exceptions Associated with the PSW" later in this chapter.) In the case of LOAD PSW, the address of the instruction has been replaced by the instruction address of the new PSW. When the invalid PSW is introduced by an interruption, the PSW-format error cannot be attributed to an instruction.

In the case of LOAD PSW and the supervisor-call interruption, a PER event may be indicated concurrently with a specification exception having an ILC of 0.

ILC on Instruction-Fetching Exceptions

When a program interruption occurs because of an exception that prohibits access to the instruction, the instruction-length code cannot be set on the basis of the first two bits of the instruction. As far as the significance of the ILC for this case is concerned, the following two situations are distinguished:

1. When an odd instruction address causes a specification exception to be recognized or when an addressing or protection exception is encountered on fetching an instruction, the ILC is set to 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the instruction address is incremented by 2, 4, or 6. By reducing the instruction address in the old

PSW by the number of halfword locations indicated in the ILC, the address originally appearing in the PSW may be obtained.

2. When a page-access exception is recognized while fetching an instruction, including the target instruction of EXECUTE, the ILC is arbitrarily set to 1, 2, or 3. In this case, the operation is nullified, and the instruction address is not incremented.

The ILC is not necessarily related to the first two bits of the instruction when the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword. The ILC may be arbitrarily set to 1, 2, or 3 in these cases. The instruction address is or is not updated, as described in situations 1 and 2 above.

When any exceptions other than page access are encountered on fetching the target instruction of EXECUTE, the ILC is 2.

Programming Notes

1. A nonzero instruction-length code for a program interruption indicates the number of halfword locations by which the instruction address in the old PSW must be reduced to obtain the address of the last instruction executed, unless one of the following situations exists:
   a. The interruption is caused by a page-access exception.
   b. An interruption for a PER event occurs before the execution of an interruptible instruction is ended.
   c. The interruption is caused by a PER event due to LOAD PSW or a branch or linkage instruction, including SUPERVISOR CALL.
   d. The interruption is caused by an access exception encountered in fetching an instruction, and the instruction address has been introduced into the PSW by a means other than sequential operation (by a branch instruction, LOAD PSW, or an interruption).
   e. The interruption is caused by a specification exception because of an odd instruction address.

   For situations a and b above, the instruction address in the PSW is not incremented, and the instruction designated by the instruction address is the same as the last one executed. These two are the only cases in which the instruction address in the old PSW identifies the instruction causing the exception.

   For situations c, d, and e, the instruction address has been replaced as part of the operation, and the address of the last instruction executed cannot be calculated using the one appearing in the old PSW.

2. When a PER event is indicated, bit 8 in the interruption code is one, the PER address in the word at location 152 identifies the location of the instruction causing the interruption, and the instruction-length code (ILC) is redundant. Similarly, the ILC is redundant when the operation is nullified, since in this case the instruction address in the PSW is not incremented. If the ILC value is required in this case, it can be derived from the operation code of the instruction identified by the old PSW.

Exceptions Associated with the PSW

Exceptions associated with erroneous information in the current PSW may be recognized when the information is introduced into the PSW or may be recognized as part of the execution of the next instruction. Errors in the PSW which are specification-exception conditions are called PSW-format errors.

Early Exception Recognition

A program interruption for a specification exception occurs immediately after the PSW becomes active if a one is introduced into an unassigned bit position of an EC-mode PSW (that is, bit positions 0, 2-5, 16, 17, or 24-39).

The interruption takes place regardless of whether the wait state is specified. If the invalid PSW causes the CPU to become enabled for a pending I/O, external, or machine-check interruption, the program interruption is taken instead, and the pending interruption is subject to the mask bits of the new PSW introduced by the program interruption.

When the execution of LOAD PSW or an interruption introduces a PSW with one of the above error conditions, the instruction-length code is set to 0, and the newly introduced PSW, except for the interruption code and the instruction-length code in the BC mode, is stored unmodified as the old PSW. When one of the above error conditions is introduced by execution of SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the instruction-length code is set to 2, and the

instruction address is updated by two halfword locations. The PSW containing the invalid value introduced into the system-mask field is stored as the old PSW.

When a PSW with one of the above error conditions is introduced during initial program loading, the loading sequence is not completed, and the load indicator remains on.

Late Exception Recognition

For the following conditions, the exception is recognized as part of the execution of the next instruction:

- A specification exception is recognized due to an odd instruction address in the PSW (PSW bit 63 is one).
- An access exception (addressing, page-access, or protection) is associated with the location designated by the instruction address or with the location of the second or third halfword of the instruction starting at the designated address.

The instruction-length code and instruction address stored in the program old PSW under these conditions are discussed in the section "ILC on Instruction-Fetching Exceptions" in this chapter.

If the invalid PSW causes the CPU to be enabled for a pending I/O, external, or machine-check interruption, the corresponding interruption occurs, and the PSW invalidity is not recognized. Similarly, the specification or access exception is not recognized in a PSW specifying the wait state.

Programming Notes

1. The execution of LOAD PSW, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK is suppressed on an addressing or protection exception, and hence the program old PSW provides information concerning the program causing the exception.
2. When the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword, the ILC is not necessarily related to the operation code.
3. If the new PSW introduced by an interruption contains a PSW-format error, a string of interruptions occurs. (See the section "Priority of Interruptions" in this chapter.)

External Interruption

The external interruption provides a means by which the CPU responds to various signals originating either from within or from without the system.

An external 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 the interruption code. When the old PSW specifies the EC mode, the interruption code is stored at locations 134-135, and zeros are stored at locations 132-133. When the old PSW specifies the BC mode, the interruption code is placed in bit positions 16-31 of the old PSW, and the instruction-length code is unpredictable.

External-interruption conditions are of two types: those for which an interruption request condition is held pending, and those for which the condition directly requests the interruption. Clock comparator and CPU timer are conditions which directly request external interruptions. If a condition which directly requests an external interruption is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and if the condition persists, more than one interruption may result from a single occurrence of the condition.

When several interruption requests for a single source are generated before the interruption is taken, and the interruption condition is of the type which is held pending, only one request for that source is preserved and remains pending.

An external interruption for a particular source can occur only when the CPU is enabled for interruption by that source. The external interruption occurs at the completion of a unit of operation. Whether the CPU is enabled for external interruption is controlled by the external mask, PSW bit 7, and external subclass mask bits in control register 0. Each source for an external interruption has a subclass mask bit assigned to it, and the source can cause an interruption only when the external-mask bit is one and the corresponding subclass-mask bit is one. The use of the subclass-mask bits does not depend on whether the CPU is in the EC or BC mode.

When the CPU becomes enabled for a pending external-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

More than one source may present a request for an external interruption at the same time. When the CPU becomes enabled for more than one concurrently pending request, the interruption

occurs for the pending condition or conditions having the highest priority.

The priorities for external-interruption requests in descending order are as follows:
Interval timer, interrupt key, external signals 2-7
Clock comparator
CPU timer

The interval timer, interrupt key, and external signals 2-7 are of equal priority; if more than one of these conditions is pending and allowed, the conditions are indicated concurrently. All other requests are honored one at a time.

Clock Comparator

An interruption request for the clock comparator exists whenever either of the following conditions is met:

1. The time-of-day clock is in the set or not-set state, and the value of the clock comparator is less than the value in the compared portion of the time-of-day clock, both compare values being considered unsigned binary integers.
2. The time-of-day clock is in the error or not-operational state.

If the condition responsible for the request is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from a single occurrence of the condition.

The clock-comparator condition is indicated by an external-interruption code of 1004 (hex).

The subclass-mask bit is in bit position 20 of control register 0. This bit is initialized to zero.

CPU Timer

An interruption request for the CPU timer exists whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). If the value is made positive before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may occur from a single occurrence of the condition.

The CPU-timer condition is indicated by an external-interruption code of 1005 (hex).

The subclass-mask bit is in bit position 21 of control register 0. This bit is initialized to zero.

External Signal

An interruption request for an external signal is generated when a signal is received on one or more of the signal-in lines. Up to six signal-in lines may be connected, providing for external signal 2 through external signal 7.The request is preserved and remains pending in the CPU until it is cleared. The pending request is cleared when it causes an interruption and by program reset.

Facilities are provided for holding a separate external-signal request pending for each of the six lines.

External signals 2-7 are indicated by setting to one interruption-code bits 10-15, respectively. Bits 0-7 are set to zeros, and any other bits in the rightmost byte are set to zeros unless set to ones for other conditions that are concurrently indicated.

All external signals are subject to control by the subclass-mask bit in bit position 26 of control register 0. This bit is initialized to one.

External signaling is independent of I/O operations and interruptions.

Programming Note

The pattern presented in bit positions 10-15 of the interruption code depends on the pattern received before the interruption is taken. Because of circuit skew, all simultaneously generated external signals do not necessarily arrive at the same time, and some may not be included in the external interruption resulting from the earliest signals. These late signals may cause another interruption to be taken.

Interrupt Key

An interruption request for the interrupt key is generated when the operator activates that key. The request is preserved and remains pending in the CPU until it is cleared. The pending request is cleared when it causes an interruption and by program reset.

When the interrupt key is activated while the CPU is in the load state, it depends on the model whether an interruption request is generated or the condition is lost.

The interrupt-key condition is indicated by setting bit 9 in the interruption code to one and by setting bits 0-7 to zeros. Bits 8 and 10-15 are zeros unless set to ones for other conditions that are concurrently indicated.

The subclass-mask bit is in bit position 25 of control register 0. This bit is initialized to one.

Interval Timer

An interruption request for the interval timer is generated when the value of the interval timer is

decremented from a positive number or zero to a negative number. The request is preserved and remains pending in the CPU until it is cleared. The pending request is cleared when it causes an interruption and by program reset.

The interval-timer condition is indicated by setting bit 8 in the interruption code to one and by setting bits 0-7 to zeros. Bits 9-15 are zeros unless set to ones for other conditions that are concurrently indicated.

The subclass-mask bit is in bit position 24 of control register 0. This bit is initialized to one.

Input/Output Interruption

The input/output (I/O) interruption provides a means by which the CPU responds to conditions in I/O devices and channels.

A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests are preserved and remain pending in channels or devices until accepted by the CPU. The I/O interruption occurs at the completion of a unit of operation. Priority is established among requests so that only one interruption request is processed at a time. For more details, see the section "Input/Output Interruptions" in Chapter 12, "Input/Output Operations."

When the CPU becomes enabled for I/O interruptions and a channel has established priority for a pending I/O-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

An I/O interruption causes the old PSW to be stored at location 56, a channel status word to be stored at location 64, and a new PSW to be fetched from location 120. Upon detection of equipment errors, additional information may be stored in the form of a limited channel logout at location 176.

When the old PSW specifies the EC mode, the I/O address identifying the channel and device causing the interruption is stored at locations 186-187, and zeros are stored at location 185. When the old PSW specifies the BC mode, the interruption code in PSW bit positions 16-31 contains the I/O address, and the instruction-length code in the PSW is unpredictable.

An I/O interruption can occur only while the CPU is enabled for interruption by the channel presenting the request. Mask bits in the PSW and channel masks in control register 2 determine whether the CPU is enabled for interruption by a channel; the method of control depends on whether the current PSW specifies the EC or BC mode.

The channel-mask bits in control register 2 start at bit position 0 and extend for as many contiguous bit positions as the number of channels provided. The assignment is such that a bit is assigned to the channel whose address is equal to the position of the bit in control register 2. Channel-mask bits for installed channels are initialized to one. The state of the channel-mask bits for unavailable channels is unpredictable.

When the current PSW specifies the EC mode, each channel is controlled by the I/O-mask bit, PSW bit 6, and by the corresponding channel-mask bit in control register 2; the channel can cause an interruption only when the I/O-mask bit is one and the corresponding channel-mask bit is one.

When the current PSW specifies the BC mode, interruptions from channels 6 and up are controlled by the I/O-mask bit, PSW bit 6, in conjunction with the corresponding channel-mask bit: the channel can cause an interruption only when the I/O-mask bit is one and the corresponding channel-mask bit is one. Interruptions from channels 0-5 are controlled by channel-mask bits 0-5 in the PSW: an interruption can occur only when the mask bit corresponding to the channel is one. In the BC mode, bits 0-5 in control register 2 do not participate in controlling I/O interruptions; they are, however, preserved in the control register if the corresponding channels are installed.

Machine-Check Interruption

The machine-check interruption is a means for reporting to the program the occurrence of equipment malfunctions. Information is provided to assist the program in determining the location of the fault and extent of the damage.

A machine-check interruption causes the old PSW to be stored at location 48 and a new PSW to be fetched from location 112. When the old PSW specifies the BC mode, the contents of the interruption-code and ILC fields in the old PSW are unpredictable.

The cause and severity of the malfunction are identified by a 64-bit machine-check-interruption code stored at locations 232-239. Further information identifying the cause of the interruption and the location of the fault may be stored at locations 216-511.

The interruption action and the storing of the associated information are under the control of

PSW bit 13 and bits in control register 14. See Chapter 11, "Machine-Check Handling," for more detailed information.

Program Interruption

Program interruptions are used to report exceptions and events which occur during execution of the program. Exceptions include the improper specification or use of instructions and data. Events are detected during monitoring (monitor events) and program-event recording (PER events).

A 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 interruption code. When the old PSW specifies the EC mode, the interruption code is placed at locations 142-143, the instruction-length code is placed in bit positions 5 and 6 of the byte at location 141 with the rest of the bits set to zeros, and zeros are stored at location 140. When the old PSW specifies the BC mode, the interruption code and the ILC are placed in the old PSW. For some causes, additional information identifying the reason for the interruption is stored at locations 144-159 in both the EC and BC modes.

Except for the PER-event condition, the condition causing the interruption is indicated by a coded value placed in the rightmost seven bit positions of the interruption code. Only one condition at a time can be indicated. Bits 0-7 of the interruption code are set to zeros.

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

A program interruption can occur only when the corresponding mask bit, if any, is one. The program mask in the PSW permits masking four of the exceptions, bit 1 in control register 0 controls whether SET SYSTEM MASK causes a special-operation exception, bits 16-31 in control register 8 control interruptions due to monitor events, and, in the EC mode, masks are provided for controlling interruptions due to PER events. When the mask bit is zero, the condition is ignored; the condition does not remain pending.

Programming Notes

1. When the new PSW for a program interruption has a PSW-format error or causes an exception to be recognized in the process of instruction fetching, a string of program interruptions takes place. See the section "Priority of Interruptions" in this chapter for a description of how such strings are terminated.
2. Some of the conditions indicated as program exceptions may be recognized also by an I/O operation, in which case the exception is indicated in the channel-status word.

Program-Interruption Conditions

The following is a detailed description of each program-interruption condition.

Addressing Exception

An addressing exception is recognized when the CPU causes a reference to a virtual-storage location that is not provided. A storage location is not provided when the page address, bits 8-20 of the storage address, equals or exceeds the page-capacity count. An address designating a storage location that is not provided is referred to as invalid.

The execution of the instruction is suppressed when the location of the instruction, including the location of the target instruction of EXECUTE, is not provided. Except for some specific instructions whose execution is suppressed, the operation is terminated when an operand location is not provided. For termination, changes may occur only to result fields, which include the condition code, registers, and any storage locations that are provided and that are designated to be changed by the instruction. Therefore, if an instruction is due to change only the contents of a field in storage, and every byte of the field is in a location that is not provided, the operation is suppressed.

The instructions whose execution is always suppressed are LOAD PSW, SET CLOCK COMPARATOR, SET CPU TIMER, SET SYSTEM MASK, STORE CLOCK COMPARATOR, STORE CPU ID, STORE CPU TIMER, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK.

When part of an operand location is provided and part is not, storing may be performed in the part that is provided.

When the address of any halfword of an instruction is invalid, the instruction-length code

(ILC) is 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the ILC is 1, 2, or 3.

In all cases of addressing exceptions not associated with instruction fetching, the ILC is 1, 2, or 3, designating the length of the instruction that caused the reference. When an addressing exception is associated with fetching the target of EXECUTE, the ILC is 2.

Data Exception

A data exception is recognized when:

1. The sign or digit codes of operands in the decimal instructions (described in Chapter 8, "Decimal Instructions") or in CONVERT TO BINARY are invalid.
2. The operand fields in ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, and SUBTRACT DECIMAL overlap in a way other than with coincident rightmost bytes; or operand fields in ZERO AND ADD overlap, and the rightmost byte of the second operand is to the right of the rightmost byte of the first operand.
3. The multiplicand in MULTIPLY DECIMAL has an insufficient number of high-order zeros.

For all instructions other than EDIT and EDIT AND MARK, the action taken for a data exception depends on whether a sign code is invalid. The operation is suppressed when a sign code is invalid, regardless of whether any other condition causing the exception exists; when no sign code is invalid, the operation is terminated. When the operation is terminated, the contents of the sign position in the rightmost byte of the result field either remain unchanged or are set to the preferred sign code; the contents of the remainder of the result field are unpredictable.

In the case of EDIT and EDIT AND MARK, an invalid sign code is not recognized; the operation is terminated on a data exception for an invalid digit code.

The instruction-length code is 2 or 3.

Programming Notes

1. The definition for data exception permits termination when digit codes are invalid but no sign code is invalid. On some models, valid digit codes may be placed in the result location even if the original contents were invalid. Thus it is possible, after getting a data exception, for all fields to appear valid.
2. When, on a program interruption for data exception, the program finds that a sign code is invalid, the operation has been suppressed if the following two conditions are met:
   a. The invalid sign of the source field is not located in the numeric portion of the result field.
   b. The sign code appears in a position specified by the instruction to be checked for valid sign. (This condition excludes the first operand of ZERO AND ADD and both operands of EDIT and EDIT AND MARK.)

   An invalid sign code for the rightmost byte of the result field is not generated when the operation is terminated. However, an invalid second-operand sign code is not necessarily preserved when it appears in the numeric portion of the result field.

Decimal-Divide Exception

A decimal-divide exception is recognized when in decimal division the divisor is zero or the quotient exceeds the specified data-field size.

The decimal-divide exception is indicated only if the sign codes of both the divisor and dividend are valid and only if the digit or digits used in establishing the exception are valid.

The operation is suppressed.

The instruction-length code is 2 or 3.

Decimal-Overflow Exception

A decimal-overflow exception is recognized when one or more significant high-order digits are lost because the destination field in a decimal operation is too short to contain the result.

The interruption may be disallowed by PSW bit 21 in the EC mode and by PSW bit 37 in the BC mode.

The operation is completed. The result is obtained by ignoring the overflow information, and condition code 3 is set.

The instruction-length code is 2 or 3.

Execute Exception

The execute exception is recognized when the target instruction of EXECUTE is another EXECUTE.

The operation is suppressed.

The instruction-length code is 2.

Exponent-Overflow Exception

An exponent-overflow exception is recognized when the result characteristic in floating-point

addition, subtraction, multiplication, or division exceeds 127 and the result fraction is not zero.

The operation is completed. The fraction is normalized, and the sign and fraction of the result remain correct. The result characteristic is made 128 smaller than the correct characteristic.

The instruction-length code is 1 or 2.

Exponent-Underflow Exception

An exponent-underflow exception is recognized when the result characteristic in floating-point addition, subtraction, multiplication, halving, or division is less than zero and the result fraction is not zero.

The operation is completed. The exponent-underflow mask also affects the result of the operation. When the mask bit is zero, the sign, characteristic, and fraction are set to zero, making the result a true zero. When the mask bit is one, the fraction is normalized, the characteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct.

The instruction-length code is 1 or 2.

Fixed-Point-Divide Exception

A fixed-point-divide exception is recognized when in fixed-point division the divisor is zero or the quotient exceeds the register size, or when the result of CONVERT TO BINARY exceeds 31 bits.

In the case of division, the operation is suppressed. The execution of CONVERT TO BINARY is completed by ignoring the high-order bits that cannot be placed in the register.

The instruction-length code is 1 or 2.

Fixed-Point-Overflow Exception

A fixed-point-overflow exception is recognized when an overflow occurs during signed binary arithmetic or left-shift operations.

The interruption may be disallowed by PSW bit 20 in the EC mode and by PSW bit 36 in the BC mode.

The operation is completed. The result is obtained by ignoring the overflow information, and condition code 3 is set.

The instruction-length code is 1 or 2.

Floating-Point-Divide Exception

A floating-point-divide exception is recognized when a floating-point division by a number with a zero fraction is attempted.

The operation is suppressed.

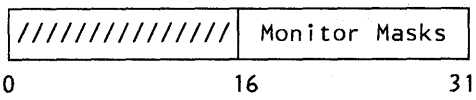
The instruction-length code is 1 or 2.

Monitor Event

A monitor event is recognized when MONITOR CALL is executed and the monitor-mask bit in control register 8 corresponding to the class specified by instruction bits 12-15 is one.

The monitor event can occur in both the EC and BC modes.

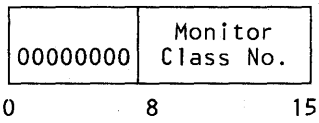
Control Register 8:



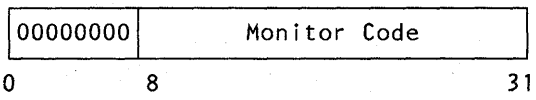
The monitor-mask bits, bits 16-31 of control register 8, correspond to monitor classes 0-15, respectively. Any number of monitor-mask bits may be on at a time; together they specify the classes of monitor events that are monitored at that time. The mask bits are initialized to zero.

When a MONITOR CALL instruction is interpreted for execution and the corresponding monitor-mask bit is one, a program interruption for monitoring occurs. The cause of the interruption is identified by setting bit 9 of the interruption code to one, and by the information stored at locations 148-149 and 156-159. The format of the information stored at these locations is the same in the EC and BC modes and is as follows:

Locations 148-149:



Locations 156-159:



The contents of bit positions 8-15 of MONITOR CALL are stored at location 149 and constitute the monitor-class number. The address specified by the B_1 and D_1 fields of the instruction forms the monitor code, which is stored at locations 157-159. Zeros are stored at locations 148 and 156.

The operation is completed, and the instruction-length code is 2.

Operation Exception

An operation exception is recognized when the CPU encounters an instruction with an invalid

operation code. The operation code may not be assigned, or the instruction with that operation code may not be available on the CPU.

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

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

The operation is suppressed.

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

Programming Notes

1. Some models may offer instructions not described in this publication, such as those provided for emulation or as part of special or custom features. Consequently, operation codes not described in this publication do not necessarily cause an operation exception to be recognized. Furthermore, these instructions may cause modes of operation to be set up or may otherwise alter the machine so as to affect the execution of subsequent instructions. To avoid causing such an operation, an instruction with an operation code not described in this publication should be issued only when the specific function associated with the operation code is desired.
2. The operation code 00, with a two-byte instruction format, currently is not assigned. It is improbable that this operation code will ever be assigned.
3. In the case of I/O instructions with the values 9C, 9D, and 9E in bit positions 0-7, the value of bit 15 is used to distinguish between two instructions. Bits 8-14, however, are not checked for zeros, and these operation codes never cause an operation exception to be recognized.

   To ensure that presently written programs run if and when the I/O operation codes (9C, 9D, 9E, and 9F) are extended further to provide for new functions, only zeros should be placed in the unused bit positions in the second op-code byte. In accordance with these recommendations, the operation codes for the I/O instructions are shown as 9C00, 9C01, 9D00, etc.

Page-Access Exception

A page-access exception is recognized when storage is addressed either explicitly or implicitly by the CPU and the addressed storage location is in a page that is in the connected or disconnected state.

The exception is recognized as part of the execution of the instruction when an attempt is made to access either the instruction or operand location. However, page-access exceptions are not recognized for the page operands of the instructions CLEAR PAGE, CONNECT PAGE, DECONFIGURE PAGE, DISCONNECT PAGE, MAKE ADDRESSABLE, and MAKE UNADDRESSABLE.

The unit of operation is nullified, except for the possible effects on storage described in the section "Nontransparent Nullification" in this chapter.

The address of the storage location causing the exception is stored at locations 145-147, and zeros are stored at location 144. The low-order 11 bits of the address stored are unpredictable.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception. When the exception occurs during fetching of an instruction, the ILC is 1, 2, or 3, the value being unpredictable.

Page-State Exception

A page-state exception is recognized when the target page of the CLEAR PAGE instruction is in the disconnected state.

The operation is suppressed.

The instruction-length code is 2.

Page-Transition Exception

A page-transition exception can only be recognized for instructions that cause a page-state transition. These instructions are CONNECT PAGE, DECONFIGURE PAGE, DISCONNECT PAGE, MAKE ADDRESSABLE, and MAKE UNADDRESSABLE.

The exception is recognized as part of the execution of the instruction when attempting to perform an invalid page-state transition. For the definition of an invalid page-state transition, see the section "Page States" in Chapter 3, "Storage."

The operation is suppressed.

The instruction-length code is 2.

PER Event

A PER event is recognized when program-event recording is specified by the contents of control registers 9-11 and one or more of these events occur.

The interruption may be disallowed by PSW bit 1 in the EC mode. Program-event recording is disallowed in the BC mode.

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

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

The instruction-length code is 0, 1, 2, or 3. Code 0 is set only if a specification exception is indicated concurrently.

Privileged-Operation Exception

A privileged-operation exception is recognized when the CPU encounters a privileged instruction in the problem state.

The operation is suppressed.

The instruction-length code is 1 or 2.

Protection Exception

A protection exception is recognized when the CPU attempts to access a storage location that is protected against the type of reference by the storage key.

The execution of an instruction is suppressed when the location of the instruction, including the location of the target instruction of EXECUTE, is protected against fetching. Except for some specific instructions whose execution is suppressed, the operation is terminated when a protection exception is encountered during a reference to an operand location. Changes may occur only to result fields. In this context, the term "result field" includes condition code, registers, and storage locations, if any, which are designated to be changed by the instruction. However, no change is made to a storage location when a reference to that location causes a protection exception. Therefore, if an instruction is due to change only the contents of a field in storage, and every byte of that field would cause a protection exception, the operation is suppressed.

The instructions whose execution is always suppressed are: LOAD PSW, SET CLOCK COMPARATOR, SET CPU TIMER, SET SYSTEM MASK, STORE CLOCK COMPARATOR, STORE CPU ID, STORE CPU TIMER, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK.

On fetching, the protected information is not loaded into an addressable register nor moved to another storage location. When a part of an operand is protected against storing and part is not, storing may be performed in the unprotected part. However, the contents of a protected location remain unchanged.

For a protected operand location, the instruction-length code (ILC) is 1, 2, or 3, designating the length of the instruction that caused the reference.

When the location of any part of an instruction is protected against fetching, the ILC is 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the ILC is 1, 2, or 3.

Significance Exception

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

The interruption may be disallowed by PSW bit 23 in the EC mode and by PSW bit 39 in the BC mode.

The operation is completed. The significance mask also affects the result of the operation. When the mask bit is zero, the operation is completed by replacing the result with a true zero. When the mask bit is one, the operation is completed without further change to the characteristic and sign of the result.

The instruction-length code is 1 or 2.

Special-Operation Exception

A special-operation exception is recognized when the instruction SET SYSTEM MASK is encountered in the supervisor state and the SSM-suppression-control bit, bit 1 of control register 0, is one.

The execution of SET SYSTEM MASK is suppressed.

The instruction-length code is 2.

Specification Exception

A specification exception is recognized for the following causes:

1. An odd instruction address is introduced into the PSW.
2. An operand address does not designate an integral boundary in an instruction requiring such integral-boundary designation.

3. The storage address in INSERT STORAGE KEY or SET STORAGE KEY does not have zeros in the four low-order bit positions.
4. An odd-numbered general register is designated by an R field of an instruction that requires an even-numbered register designation.
5. A floating-point register other than 0, 2, 4, or 6 is specified for a short or long operand, or a floating-point register other than 0 or 4 is specified for an extended operand.
6. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.
7. The length of the first-operand field is less than or equal to the length of the second-operand field in decimal multiplication or division.
8. Bit positions 8-11 of MONITOR CALL do not contain zeros.
9. A one is introduced into an unassigned bit position of an EC-mode PSW (bit positions 0, 2-5, 16, 17, and 24-39).
10. Page 0 is designated to become connected or disconnected.

The execution of the instruction identified by the old PSW is suppressed. However, for cause 9, the operation that introduces the new PSW is completed, but an interruption occurs immediately thereafter.

When the instruction address is odd (cause 1), the instruction-length code (ILC) is 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the ILC is 1, 2, or 3.

For causes 2-8 and 10, the ILC is 1, 2, or 3, designating the length of the instruction causing the exception.

When the exception is recognized because of cause 9, and the invalid bit value has been introduced by LOAD PSW or an interruption, the ILC is 0. When the exception due to cause 9 is introduced by SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the ILC is 2.

See the section "Exceptions Associated with the PSW" in this chapter for a discussion of when the exceptions associated with the PSW are recognized.

Recognition of Access Exceptions

The addressing, page-access, and protection exceptions are collectively referred to as access exceptions.

Any access exception is recognized as part of the execution of the instruction with which the exception is associated. An access exception is not recognized when the CPU has made an attempt to fetch from an inaccessible location or has detected some other access-exception condition, but a branch instruction or an interruption changes the instruction sequence such that the instruction is not executed.

Every instruction can cause an access exception to be recognized because of instruction fetch. Additionally, access exceptions associated with instruction execution may occur because of an access to an operand in storage.

An access exception due to fetching an instruction is indicated when the first instruction halfword cannot be fetched without encountering the exception. When the first halfword of the instruction has no access exceptions, access exceptions may be indicated for additional halfwords according to the instruction length specified by the first two bits of the instruction; however, when the operation can be performed without accessing the second or third halfwords of the instruction, it is unpredictable whether the access exception is indicated for the unused part. Since the indication of access exceptions for instruction fetch is common to all instructions, it is not covered in the individual instruction definitions.

Except where otherwise indicated in the individual instruction description, the following rules apply for exceptions associated with an access to an operand location. For a fetch-type operand, access exceptions are necessarily indicated only for that portion of the operand which is required for completing the operation. It is unpredictable whether access exceptions are indicated for those portions of a fetch-type operand which are not required for completing the operation. For a store-type operand, access exceptions are recognized for the entire operand even if the operation could be completed without the use of the inaccessible part of the operand. In situations where the value of a store-type operand is defined to be unpredictable, it is unpredictable whether an access exception is indicated.

Whenever an access to an operand location can cause an access exception to be recognized, the word "access" is included in the list of program exceptions in the description of the instruction. This entry also indicates which operand can cause the exception to be recognized and whether the exception is recognized on a fetch or store access to that operand location. Access exceptions are recognized only for the portion of the operand as defined by each particular instruction.

Multiple Program-Interruption Conditions

Except for PER events, only one program-interruption condition is indicated with a program interruption. The existence of one condition, however, does not preclude the existence of other conditions. When more than one program-interruption condition exists, only the condition having the highest priority is identified in the interruption code.

With two conditions of the same priority, it is unpredictable which is indicated. In particular, the priority of access exceptions associated with the two parts of an operand that crosses a page boundary is unpredictable and is not necessarily related to the sequence specified for the access of bytes within the operand.

The type of ending which occurs (nullification, suppression, or termination) is that which is defined for the type of exception that is indicated in the interruption code. However, if a condition is indicated which permits termination, and another condition also exists which would cause either nullification or suppression, then the unit of operation is suppressed.

The figure "Priority of Program-Interruption Conditions" lists the priorities of all program-interruption conditions other than PER events. All exceptions associated with references to storage for a particular instruction halfword or a particular operand byte are grouped as a single entry called "access." The priorities of access exceptions for a single access are, in descending order of priorities:

1. Addressing exception
2. Page-access exception
3. Protection exception

The relative priorities of any two conditions can be found by comparing the priority numbers within a table from left to right until a mismatch is found. If the first inequality is between numeric characters, either the two conditions are mutually exclusive or, if both can occur, the condition with the smaller number is indicated. If the first inequality is between alphabetic characters, then the two conditions are not exclusive, and it is unpredictable which is indicated when both occur.

1.	Specification exception due to a one in an unassigned bit position of an EC-mode PSW.[1]
2.	Specification exception due to an odd instruction address in the PSW.
3.	Access exceptions for first halfword of EXECUTE.[2]
4.	Access exceptions for second halfword of EXECUTE.[2]
5.	Specification exception due to target instruction of EXECUTE not being specified on halfword boundary.[2]
6.	Access exceptions for first instruction halfword.
7.A	Access exception for second instruction halfword.[3]
7.B	Access exception for third instruction halfword.[3]
7.C.1	Operation exception.
7.C.2	Privileged-operation exception.
7.C.3	Execute exception.
7.C.4	Special-operation exception.
8.A	Specification exception due to conditions other than those included in 1, 2 and 5 above.
8.B	Access exceptions for any particular access to an operand in storage.[4]
8.C	Data exception.[5]
8.D	Decimal-divide exception.[6]
8.E	Page-state exception.
9.	Page-transition exception.
10.	Fixed-point divide, floating-point divide, and conditions, other than PER events, which result in completion. These conditions are mutually exclusive, or their priority is specified in the corresponding definitions.

Explanation:

Numbers indicate priority, with priority decreasing in ascending order of numbers; letters indicate no priority.

[1] A one may be introduced in an unassigned bit position of an EC-mode PSW by a new PSW loaded as a result of an interruption or by the instructions LOAD PSW, SET SYSTEM MASK, and STORE THEN OR SYSTEM MASK. The priority shown in the chart is that for a PSW error introduced by an interruption and may also be considered as the priority for a PSW error introduced by the previous instruction. The error is introduced only if the instruction encounters no other exceptions. The resulting interruption has a higher priority than any interruption caused by the instruction which would have been executed next; it has a lower priority, however, than any interruption caused by the instruction which introduced the erroneous PSW.

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

Explanation (Continued):

2 Priorities 3, 4, and 5 apply only to an EXECUTE instruction. Priorities 6-10 apply to instructions other than EXECUTE, including the target instruction of EXECUTE.

3 Separate accesses may occur for each halfword of an instruction. The second instruction halfword is accessed if bits 0-1 of the instruction are not both zeros. The third instruction halfword is accessed only if bits 0-1 of the instruction are both ones. Access exceptions for one of these halfwords are not necessarily recognized if the instruction can be completed without use of the contents of the halfword or if an exception of priority 8, 9, or 10 can be determined without the use of the halfword.

4 As in instruction fetching, separate accesses may occur for each portion of an operand. Each of the accesses is of equal priority. Addressing exceptions for INSERT STORAGE KEY, RESET REFERENCE BIT, and SET STORAGE KEY are also included in 8.B. For MOVE LONG and COMPARE LOGICAL LONG, an access exception for a particular operand can be indicated only if the R field for that operand designates an even-numbered register.

5 The exception can be indicated only if the sign, digit, or digits responsible for the exception were fetched without encountering an access exception.

6 The exception can be indicated only if the digits used in establishing the exception, and also the signs, were fetched without encountering an access exception, and only if the digits used in establishing the exception are valid.

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

Restart Interruption

The restart interruption provides a means for the operator to invoke the execution of a specified program. The CPU cannot be disabled for this interruption.

A restart interruption causes the old PSW to be stored at location 8 and a new PSW, specifying the start of the program to be executed, to be fetched from location 0. The instruction-length code and interruption code are not stored in the EC mode. In the BC mode, the instruction-length code in the PSW is unpredictable, and zeros are stored in the interruption-code field.

If the CPU is in the operating state, the exchange of the PSWs occurs at the completion of the current unit of operation and after all pending interruption conditions for which the CPU is enabled have been taken. If the CPU is in the stopped state, the CPU enters the operating state and exchanges the PSWs without first taking any pending interruptions.

The restart interruption is initiated by activating the restart key.

When the rate control is set to instruction step, it is unpredictable whether restart causes a unit of operation or additional interruptions to be performed after the PSWs have been exchanged.

Programming Note

In order to perform restart when the CPU is in the check-stop state, the CPU has to be reset. This can be accomplished by means of the system-reset-normal key, which does not clear the contents of program-addressable registers, including the control registers, but causes the channels to be reset.

Supervisor-Call Interruption

The supervisor-call interruption occurs when the instruction SUPERVISOR CALL is executed. The CPU cannot be disabled for the interruption, and the interruption occurs immediately upon the execution of the instruction.

The supervisor-call interruption causes the old PSW to be stored at location 32 and a new PSW to be fetched from location 96.

The contents of bit positions 8-15 of SUPERVISOR CALL are placed in the rightmost byte of the interruption code. The leftmost byte of the interruption code is set to zero. The instruction-length code is 1, unless the instruction was executed by means of EXECUTE, in which case the code is 2.

When the old PSW specifies the EC mode, the interruption code is placed in locations 138-139, the instruction-length code is placed in bit positions 5 and 6 of the byte at location 137, with the other bits set to zeros, and zeros are stored at location 136. When the old PSW specifies the BC mode, the interruption code and instruction-length code appear in the old PSW.

Priority of Interruptions

During the execution of an instruction, several interruption-causing events may occur simultaneously. The instruction may give rise to a program interruption, a request for an external interruption may be received, equipment malfunctioning may be detected, an I/O-interruption request may be made, and the restart key may be activated. Instead of the program interruption, a supervisor-call interruption might occur; or both can occur if program-event-recording is active. Simultaneous interruption requests are honored in a predetermined order.

An exigent machine-check condition has the highest priority. When it occurs, the current operation is terminated. Program and supervisor-call interruptions that would have occurred as a result of the current operation may be eliminated. Any pending repressible machine-check conditions may be indicated with the exigent machine-check interruption. Every reasonable attempt is made to limit the side effects of an exigent machine-check condition, and requests for I/O and external interruptions normally remain unaffected.

In the absence of an exigent machine-check condition, interruption requests existing concurrently at the end of a unit of operation are honored, in descending order of priority, as follows:

Supervisor call
Program
Repressible machine check
External
Input/output
Restart

The processing of multiple simultaneous interruption requests consists in storing the old PSW and fetching the new PSW belonging to the interruption first taken. This new PSW is subsequently stored without the execution of any instructions, and the new PSW associated with the next interruption is fetched. Storing and fetching of PSWs continues until no more interruptions are to be serviced. The priority is reevaluated after each new PSW is loaded. Each evaluation is performed taking into consideration any additional interruptions which may have become pending. Additionally, external and I/O interruptions, as well as machine-check interruptions due to repressible conditions, are taken only if the current PSW at the instant of evaluation indicates that the CPU is interruptible for the cause.

Instruction execution is resumed using the last-fetched PSW. The order of executing interruption subroutines is, therefore, the reverse of the order in which the PSWs are fetched.

If the new PSW for a program interruption has an odd instruction address or causes an access exception to be recognized, another program interruption occurs. Since this second interruption introduces the same unacceptable PSW, a string of interruptions is established. These program exceptions are recognized as part of the execution of the following instruction, and the string may be broken by an external, I/O, machine-check, or restart interruption or by the stop function.

If the new PSW for a program interruption contains a one in an unassigned bit position of an EC-mode PSW, another program interruption occurs. This condition is of higher priority than restart, I/O, external, or repressible machine-check conditions, or the stop function, and program reset has to be used to break the string of interruptions.

A string of interruptions for other interruption classes can also exist if the new PSW is enabled for the interruption just taken. These include machine-check interruptions, external interruptions, and I/O interruptions due to PCI conditions generated because of CCWs which form a loop. Furthermore, a string of interruptions involving more than one interruption class can exist. For example, assume that the CPU timer is negative and the CPU-timer subclass mask is one. If the external new PSW has a one in an unassigned bit position in the EC mode, and the program new PSW is enabled for external interruptions, then a string of interruptions occurs, alternating between external and program. Even more complex strings

of interruptions are possible. As long as more interruptions must be serviced, the string of interruptions cannot be broken by employing the stop function; program reset is required.

Interruptions for all requests for which the CPU is enabled are taken before the CPU is placed in the stopped state. When the CPU is in the stopped state, restart has the highest priority.

Programming Note

The order in which concurrent interruption requests are honored can be changed to some extent by masking.

Chapter 7. General Instructions

Contents

This chapter includes all the unprivileged instructions described in this publication, other than the decimal and floating-point instructions.

Data Format

The general instructions treat data as being of four types: signed binary integers, unsigned binary integers, unstructured logical data, and decimal data. Data is treated as decimal by the conversion, packing, and unpacking instructions. Decimal data is described in Chapter 8, "Decimal Instructions."

Data resides in general registers or in storage or is introduced from the instruction stream.

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

Binary-Integer Representation

Binary integers are treated as signed or unsigned.

In an unsigned binary integer, all bits are used to express the absolute value of the number. When two unsigned binary integers of different lengths are added, the shorter number is considered to be extended on the left with zeros.

For signed binary integers, the leftmost bit represents the sign, which is followed by the numeric field. Positive numbers are represented in true binary notation with the sign bit set to zero. Negative numbers are represented in two's-complement binary notation with a one in the sign-bit position.

Specifically, a negative number is represented by the two's complement of the positive number of the same absolute value. The two's complement of a number is obtained by inverting each bit of the number, including the sign, 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 are zeros. When the number is negative, all these bits are ones. Therefore, when a signed operand must be extended with high-order bits, the extension is achieved by setting these bits equal to the sign bit of the operand.

The notation for signed binary integers does not include a negative zero. It has a number range in which the set of negative numbers is one larger than the set of positive numbers. The maximum positive number consists of a sign bit of zero followed by all ones, whereas the maximum negative number (the negative number with the greatest absolute value) consists of a sign bit of one followed by all zeros. The number zero consists of all-zero bits.

A signed binary integer of either sign, except for zero and for the maximum negative number, is changed to the number with opposite sign by forming its two's complement. This operation of complementing a number is equivalent to subtracting the number from zero. The complement of zero is zero.

The complement of the maximum negative number cannot be represented in the same number of bits. When an operation, such as a subtraction of the maximum negative number from zero, attempts to produce the complement of the maximum negative number, the result is the maximum negative number, and a fixed-point-overflow exception is recognized. An overflow does not result, however, when the maximum negative number is complemented as an intermediate result but the final result is within the representable range. An example of this case is a subtraction of the maximum negative number from minus one. The product of two maximum negative numbers is representable as a double-length positive number.

In discussions of signed binary integers in this publication, a signed binary integer includes the sign bit. Thus, the expression "32-bit signed binary integer" denotes an integer with 31 numeric bits and a sign bit, and the expression "64-bit signed binary integer" denotes an integer with 63 numeric bits and a sign bit.

In some operations, the result is achieved by the use of the one's complement of the number. The one's complement of a number is obtained by inverting each bit of the number.

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

Programming Notes

1. An alternate way of forming the two's complement of a signed binary integer is to invert all bits to the left of the rightmost one bit, leaving the rightmost one bit and all zero bits to the right of it unchanged.
2. The numeric bits of a signed binary integer may be considered to represent a positive value, with the sign representing a value of either zero or the maximum negative number.

Signed and Unsigned Binary Arithmetic

Addition of signed binary integers is performed by adding all bits of each operand, including the sign bits. When one of the operands is shorter, the shorter operand is extended on the left to the length of the longer operand by propagating the sign-bit value. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position disagree, an overflow occurs. The sign bit is not changed after the overflow.

Subtraction is performed by adding the one's complement of the second operand and a low-order one to the first operand.

Signed addition and subtraction produce an overflow when the result is outside the range of representation for signed binary integers. Specifically, for ADD and SUBTRACT, which operate on 32-bit signed binary integers, there is an overflow when the proper result would be greater than or equal to $+2^{31}$ or less than -2^{31}. The actual result placed in the general register after an overflow differs from the proper result by 2^{32}. An overflow causes a program interruption for fixed-point overflow if it is allowed.

Addition of unsigned binary integers is performed by adding all bits of each operand. When one of the operands is shorter, the shorter operand is extended on the left with zeros. Unsigned binary arithmetic is used in address arithmetic for adding the X, B, and D fields. It is also used to obtain the addresses of the function bytes in the instructions TRANSLATE and TRANSLATE AND TEST. Furthermore, unsigned binary arithmetic is used on 32-bit unsigned binary integers by the instructions ADD LOGICAL and SUBTRACT LOGICAL. Given the same two operands, ADD and ADD LOGICAL produce the same 32-bit result. The instructions differ only in the interpretation of this result. ADD interprets the result as a signed binary integer and inspects it for sign, magnitude, and overflow to set the condition code accordingly. ADD LOGICAL interprets the result as an unsigned binary integer and sets the condition code according to whether the result is zero and whether there was a carry out of the high-order bit position. Such a carry is not necessarily considered an overflow, and no program interruption can occur for ADD LOGICAL.

SUBTRACT LOGICAL differs from ADD LOGICAL in that the one's complement of the second operand and a low-order one are added to the first operand.

Programming Notes

1. Logical addition and subtraction may be used to program multiple-precision arithmetic. Thus, for multiple-precision binary-integer addition, ADD LOGICAL is used to add the corresponding lower-order parts of the operands. If the condition code indicates a carry, a one is added to the first operand of the next higher pair of integers before adding the second operand. If the integers are signed, the ADD instruction is used on the highest-order parts after propagating any carry. The condition code then indicates any overflow or the proper sign and magnitude of the entire result; an overflow is also indicated by a fixed-point-overflow interruption if it is allowed. If the integers are unsigned, ADD LOGICAL is used throughout.
2. Another use for ADD LOGICAL is to increment values representing binary counters, which are allowed to wrap around from all ones to all zeros without necessarily indicating overflow.

Signed and Logical Comparison

Comparison operations determine whether two operands are equal or not and, for most operations, which of two unequal operands is the greater (high). Signed-binary comparison operations are provided which treat the operands as signed binary integers, and logical comparison operations are provided which treat the operands as unsigned binary integers or as unstructured data.

The instructions COMPARE and COMPARE HALFWORD are signed-binary comparison operations. These instructions are equivalent to SUBTRACT and SUBTRACT HALFWORD without replacing either operand, the resulting difference being used only to set the condition code. The operations permit comparison of numbers of opposite sign which differ by 2^{32} or

more. Thus, unlike SUBTRACT, COMPARE can cause no overflow.

Logical comparison of two operands is performed byte by byte, in a left-to-right sequence. The operands are equal when all their bytes are equal. When the operands are unequal, the comparison result is determined by a left-to-right comparison of corresponding bit positions in the first unequal pair of bytes: the zero bit in the first unequal pair of bits indicates the low operand, and the one bit the high operand. Since the remaining bit and byte positions do not change the comparison, it is not necessary to continue comparing unequal operands beyond the first unequal bit pair.

Instructions

The general instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of General Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

A detailed definition of instruction formats, operand designation and length, and address generation is contained in the section "Instructions" in Chapter 5, "Program Execution." Exceptions to the general rules stated in that section are explicitly identified in the individual instruction descriptions.

Several instruction descriptions in this chapter contain references to other CPUs, even though the ECPS:VSE mode makes no provision for multiprocessing, so as to permit the writing of problem-state programs that are compatible with multiprocessing configurations of System/370 (see the section "Problem-State Compatibility" in Chapter 1, "Introduction").

Note: *In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designations for the assembler language are shown with each instruction. For LOAD AND TEST, for example, LTR is the mnemonic and* R_1, R_2 *the operand designation.*

Name	Mne-monic	Characteristics								Op Code
ADD	AR	RR	C			IF		R		1A
ADD	A	RX	C		A	IF		R		5A
ADD HALFWORD	AH	RX	C		A	IF		R		4A
ADD LOGICAL	ALR	RR	C					R		1E
ADD LOGICAL	AL	RX	C		A			R		5E
AND	NR	RR	C					R		14
AND	N	RX	C		A			R		54
AND (character)	NC	SS	C		A				ST	D4
AND (immediate)	NI	SI	C		A				ST	94
BRANCH AND LINK	BALR	RR						B R		05
BRANCH AND LINK	BAL	RX						B R		45
BRANCH ON CONDITION	BCR	RR					$[1]	B		07
BRANCH ON CONDITION	BC	RX						B		47
BRANCH ON COUNT	BCTR	RR						B R		06
BRANCH ON COUNT	BCT	RX						B R		46
BRANCH ON INDEX HIGH	BXH	RS						B R		86
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS						B R		87
COMPARE	CR	RR	C							19
COMPARE	C	RX	C		A					59
COMPARE AND SWAP	CS	RS	C		A	SP	$		R ST	BA
COMPARE DOUBLE AND SWAP	CDS	RS	C		A	SP	$		R ST	BB
COMPARE HALFWORD	CH	RX	C		A					49
COMPARE LOGICAL	CLR	RR	C							15
COMPARE LOGICAL	CL	RX	C		A					55
COMPARE LOGICAL (character)	CLC	SS	C		A					D5
COMPARE LOGICAL (immediate)	CLI	SI	C		A					95
COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS	C		A					BD
COMPARE LOGICAL LONG	CLCL	RR	C		A	SP	II	R		OF
CONVERT TO BINARY	CVB	RX			A		D IK	R		4F
CONVERT TO DECIMAL	CVD	RX			A				ST	4E
DIVIDE	DR	RR				SP	IK	R		1D
DIVIDE	D	RX			A	SP	IK	R		5D
EXCLUSIVE OR	XR	RR	C					R		17
EXCLUSIVE OR	X	RX	C		A			R		57
EXCLUSIVE OR (character)	XC	SS	C		A				ST	D7
EXCLUSIVE OR (immediate)	XI	SI	C		A				ST	97
EXECUTE	EX	RX			A	SP	EX			44
INSERT CHARACTER	IC	RX			A			R		43
INSERT CHARACTERS UNDER MASK	ICM	RS	C		A			R		BF
LOAD	LR	RR						R		18
LOAD	L	RX			A			R		58
LOAD ADDRESS	LA	RX						R		41
LOAD AND TEST	LTR	RR	C					R		12
LOAD COMPLEMENT	LCR	RR	C				IF	R		13
LOAD HALFWORD	LH	RX			A			R		48
LOAD MULTIPLE	LM	RS			A			R		98
LOAD NEGATIVE	LNR	RR	C					R		11
LOAD POSITIVE	LPR	RR	C				IF	R		10
MONITOR CALL	MC	SI				SP	MO			AF
MOVE (character)	MVC	SS			A				ST	D2

Summary of General Instructions (Part 1 of 2)

Name	Mnemonic	Characteristics								Op Code
MOVE (immediate)	MVI	SI			A				ST	92
MOVE INVERSE	MVCIN	SS			A				ST	E8
MOVE LONG	MVCL	RR	C		A	SP		II	R ST	0E
MOVE NUMERICS	MVN	SS			A				ST	D1
MOVE WITH OFFSET	MVO	SS			A				ST	F1
MOVE ZONES	MVZ	SS			A				ST	D3
MULTIPLY	MR	RR				SP			R	1C
MULTIPLY	M	RX			A	SP			R	5C
MULTIPLY HALFWORD	MH	RX			A				R	4C
OR	OR	RR	C						R	16
OR	O	RX	C		A				R	56
OR (character)	OC	SS	C		A				ST	D6
OR (immediate)	OI	SI	C		A				ST	96
PACK	PACK	SS			A				ST	F2
SET PROGRAM MASK	SPM	RR	L							04
SHIFT LEFT DOUBLE	SLDA	RS	C			SP	IF		R	8F
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS				SP			R	8D
SHIFT LEFT SINGLE	SLA	RS	C				IF		R	8B
SHIFT LEFT SINGLE LOGICAL	SLL	RS							R	89
SHIFT RIGHT DOUBLE	SRDA	RS	C			SP			R	8E
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS				SP			R	8C
SHIFT RIGHT SINGLE	SRA	RS	C						R	8A
SHIFT RIGHT SINGLE LOGICAL	SRL	RS							R	88
STORE	ST	RX			A				ST	50
STORE CHARACTER	STC	RX			A				ST	42
STORE CHARACTERS UNDER MASK	STCM	RS			A				ST	BE
STORE CLOCK	STCK	S	C		A		$		ST	B205
STORE HALFWORD	STH	RX			A				ST	40
STORE MULTIPLE	STM	RS			A				ST	90
SUBTRACT	SR	RR	C				IF		R	1B
SUBTRACT	S	RX	C		A		IF		R	5B
SUBTRACT HALFWORD	SH	RX	C		A		IF		R	4B
SUBTRACT LOGICAL	SLR	RR	C						R	1F
SUBTRACT LOGICAL	SL	RX	C		A				R	5F
SUPERVISOR CALL	SVC	RR					$			0A
TEST AND SET	TS	S	C		A		$		ST	93
TEST UNDER MASK	TM	SI	C		A					91
TRANSLATE	TR	SS			A				ST	DC
TRANSLATE AND TEST	TRT	SS	C		A				R	DD
UNPACK	UNPK	SS			A				ST	F3

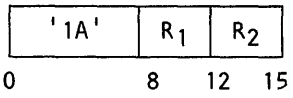
Explanation:

A	Access exceptions	RR	RR instruction format
B	PER branch event	RS	RS instruction format
C	Condition code is set	RX	RX instruction format
D	Data exception	S	S instruction format
EX	Execute exception	SI	SI instruction format
IF	Fixed-point-overflow exception	SP	Specification exception
II	Interruptible instruction	SS	SS instruction format
IK	Fixed-point-divide exception	ST	PER storage-alteration event
L	New condition code loaded	$	Causes serialization
MO	Monitor event	$1	Causes serialization when the M_1 and R_2 fields contain all ones and all zeros, respectively
R	PER general-register-alteration event		

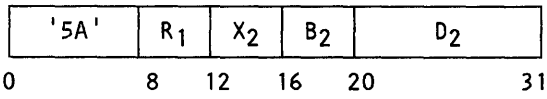
Summary of General Instructions (Part 2 of 2)

ADD

AR R_1,R_2 [RR]



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



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

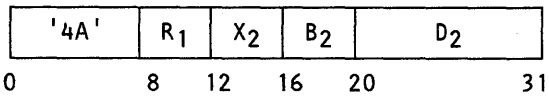
An overflow causes a program interruption when the fixedpointoverflow mask bit is one.

Resulting Condition Code:
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Exceptions:
Access (fetch, operand 2 of A only)
Fixed-Point Overflow

ADD HALFWORD

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



The second operand is added to the first operand, and the sum is placed in the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand and the sum are treated as 32-bit signed binary integers.

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

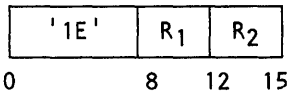
Resulting Condition Code:
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Exceptions:
Access (fetch, operand 2)
Fixed-Point Overflow

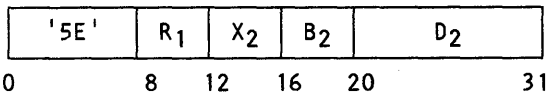
Programming Note
An example of the use of ADD HALFWORD is given in Appendix A.

ADD LOGICAL

ALR R_1,R_2 [RR]



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



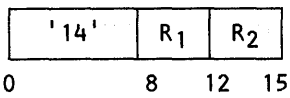
The second operand is added to the first operand, and the sum is placed in the first-operand location. The operands and the sum are treated as 32-bit unsigned binary integers.

Resulting Condition Code:
0 Sum is zero, with no carry
1 Sum is not zero, with no carry
2 Sum is zero, with carry
3 Sum is not zero, with carry

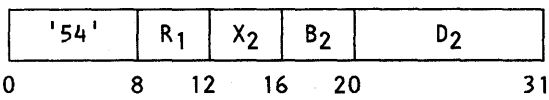
Program Exceptions:
Access (fetch, operand 2 of AL only)

AND

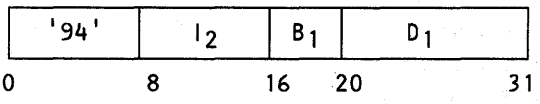
NR R_1,R_2 [RR]



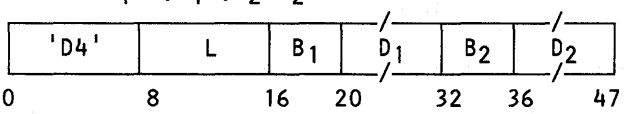
N $R_1,D_2(X_2,B_2)$ [RX]



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



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



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

The connective AND is applied to the operands bit by bit. A bit position in the result is set to one if the corresponding bit positions in both operands contain ones; otherwise, the result bit is set to zero.

For NC, each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

For NI, the first operand is one byte in length, and only one byte is stored.

Resulting Condition Code:

0 Result is zero
1 Result is not zero
2 –
3 –

Program Exceptions:

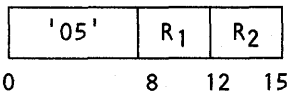
Access (fetch, operand 2, N and NC; fetch and store, operand 1, NI and NC)

Programming Notes

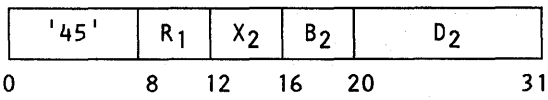
1. An example of the use of the AND instruction is given in Appendix A.
2. The instruction AND may be used to set a bit to zero.
3. Accesses to the first operand of NI and NC consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, the instruction AND cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for the instruction OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

BRANCH AND LINK

BALR R_1,R_2 [RR]



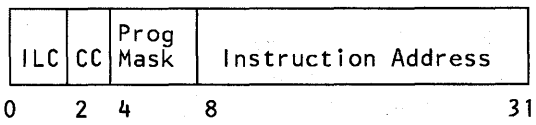
BAL $R_1,D_2(X_2,B_2)$ [RX]



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

In the RX format, the second-operand address is used as the branch address. In the RR format, bits 8-31 of the general register designated by R_2 are used as the branch address; however, when the R_2 field contains zeros, the operation is performed without branching. The branch address is computed before the link information is loaded.

The link information consists of the instruction-length code (ILC), the condition code (CC), the program mask bits, and the updated instruction address, arranged in the following format:



The instruction-length code is 1 or 2.

Condition Code: The code remains unchanged.

Program Exceptions: None.

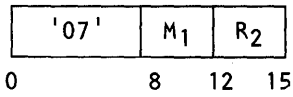
Programming Notes

1. An example of the use of BRANCH AND LINK is given in Appendix A.
2. When the R_2 field in the RR format contains all zeros, the link information is loaded without branching.
3. When BRANCH AND LINK is the target instruction of EXECUTE, the instruction-length code is 2.
4. The format and the contents of the link information do not depend on whether the

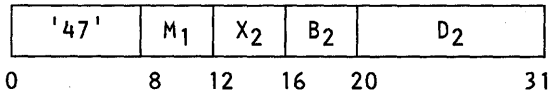
PSW specifies the EC or BC mode. In both modes, the link information is in the format of the rightmost 32 bit positions of the BC-mode PSW.

BRANCH ON CONDITION

BCR M_1,R_2 [RR]



BC $M_1,D_2(X_2,B_2)$ [RX]



The instruction address in the current PSW is replaced by the branch address if the condition code has one of the values specified by M_1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

In the RX format, the second-operand address is used as the branch address. In the RR format, bits 8-31 of the general register specified by R_2 are used as the branch address; however, when the R_2 field contains zeros, the operation is performed without branching.

The M_1 field is used as a four-bit mask. The four condition codes (0, 1, 2, and 3) correspond, left to right, with the four bits of the mask, as follows:

Condition Code	Instruction Bit	Mask Position Value
0	8	8
1	9	4
2	10	2
3	11	1

The current condition code is used to select the corresponding mask bit. If the mask bit selected by the condition code is one, the branch is successful. If the mask bit selected is zero, normal instruction sequencing proceeds with the next sequential instruction.

When the M_1 and R_2 fields of BCR are all ones and all zeros, respectively, a serialization function is performed. CPU operation is delayed until all previous accesses by this CPU to storage have been completed, as observed by channels and other CPUs. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed.

Condition Code: The code remains unchanged.

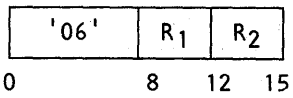
Program Exceptions: None.

Programming Notes

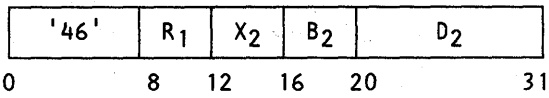
1. An example of the use of BRANCH ON CONDITION is given in Appendix A.
2. When a branch is to depend on more than one condition, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made when the condition code is 0 or 1.
3. When all four mask bits are zero or when the R_2 field in the RR format contains zero, the branch instruction is equivalent to a no-operation. When all four mask bits are ones, that is, the mask value is 15, the branch is unconditional unless the R_2 field in the RR format is zero.
4. Execution of BCR 15,0 (that is, an instruction with a value of 07F0 hex) may result in significant performance degradation. To ensure optimum performance, the program should avoid use of BCR 15,0 except in cases when the serialization function is actually required.
5. Note that the relation between the RR and RX formats in branch-address specification is not the same as in operand-address specification. For branch instructions in the RX format, the branch address is the address specified by X_2, B_2, and D_2; in the RR format, the branch address is contained in the register specified by R_2. For operands, the address specified by X_2, B_2, and D_2 is the operand address, but the register specified by R_2 contains the operand itself.

BRANCH ON COUNT

BCTR R_1,R_2 [RR]



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



A one is subtracted from the first operand, and the result is placed in the first-operand location. The first operand and result are treated as 32-bit binary integers, with overflow ignored. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address in the current PSW is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of bit positions 8-31 of the general register specified by R_2 are used as the branch address; however, when the R_2 field contains zeros, the operation is performed without branching.

The branch address is computed before the counting operation.

Condition Code: The code remains unchanged.

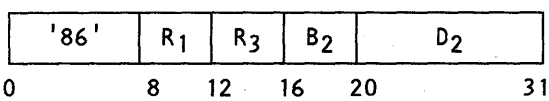
Program Exceptions: None.

Programming Notes

1. An example of the use of BRANCH ON COUNT is given in Appendix A.
2. The first operand and result can be considered as either signed or unsigned binary integers since the result of a binary subtraction is the same in both cases.
3. An initial count of one results in zero, and no branching takes place; an initial count of zero results in −1 and causes branching to be executed; an initial count of −1 results in −2 and causes branching to be executed; and so on. In a loop, branching takes place each time the instruction is executed until the result is again zero. Note that, because of the number range, an initial count of -2^{31} results in a positive value of $2^{31} - 1$.
4. Counting is performed without branching when the R_2 field in the RR format contains zero.

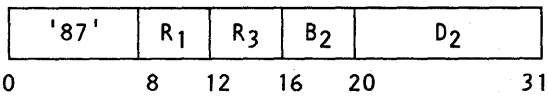
BRANCH ON INDEX HIGH

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



BRANCH ON INDEX LOW OR EQUAL

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



An increment is added to the first operand, and the sum is compared with a compare value. The result of the comparison determines whether branching occurs. Subsequently, the sum is placed in the first-operand location. The second-operand address is used as a branch address. The R_3 field designates registers containing the increment and the compare value.

For BXH, when the sum is high, the instruction address in the current PSW is replaced by the branch address. When the sum is low or equal, normal instruction sequencing proceeds with the updated instruction address.

For BXLE, when the sum is low or equal, the instruction address in the current PSW is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address.

When the R_3 field is even, it designates a pair of registers; the contents of the even and odd registers of the pair are used as the increment and the compare value, respectively. When the R_3 field is odd, it designates a single register, the contents of which are used as both the increment and the compare value.

For purposes of the addition and comparison, all operands and results are treated as 32-bit signed binary integers. Overflow caused by the addition is ignored.

The original contents of the compare-value register are used as the compare value even when that register is also specified to be the first-operand location. The branch address is computed before the addition and comparison.

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

Condition Code: The code remains unchanged.

Program Exceptions: None.

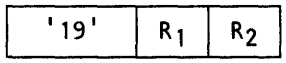
Programming Notes

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

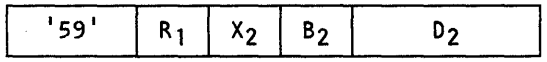
2. The word "index" in the names of these instructions indicates that one of the major purposes is the incrementing and testing of an index value. The increment, being a signed binary integer, may be used to increase or decrease the value in register R_1 by an arbitrary amount.

COMPARE

CR R_1,R_2 [RR]



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



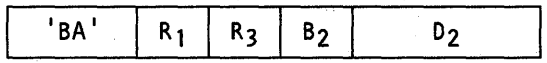
The first operand is compared with the second operand, and the result is indicated in the condition code. The operands are treated as 32-bit signed binary integers.

Resulting Condition Code:
0 Operands are equal
1 First operand is low
2 First operand is high
3 –

Program Exceptions:
Access (fetch, operand 2 of C only)

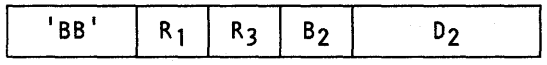
COMPARE AND SWAP

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



COMPARE DOUBLE AND SWAP

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



The first and second operands are compared. If they are equal, the third operand is stored at the second-operand location. If they are unequal, the second operand is loaded into the first-operand location. The result of the comparison is indicated in the condition code.

For CS, the first and third operands are 32 bits in length, with each operand occupying a general register. The second operand is a word in storage.

For CDS, the first and third operands are 64 bits in length, with each operand occupying an even-odd pair of general registers. The second operand is a doubleword in storage.

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

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

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

The second operand of CS must be designated on a word boundary. The R_1 and R_3 fields for CDS must each designate an even register, and the second operand for CDS must be designated on a doubleword boundary. Otherwise, a specification exception is recognized.

Resulting Condition Code:
0 First and second operands equal, second operand replaced by third operand
1 First and second operands unequal, first operand replaced by second operand
2 –
3 –

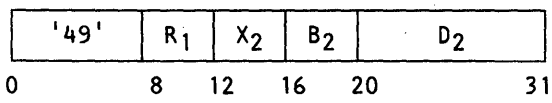
Program Exceptions:
Access (fetch and store, operand 2)
Specification

Programming Notes

1. Several examples of the use of the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions are given in Appendix A.
2. The instruction CS can be used by programs sharing common storage areas in either a multiprogramming or multiprocessing environment. Two examples are:
   a. By performing the following procedure, a program can modify the contents of a storage location even though the possibility exists that the program may be interrupted by another program that will update the location or even though the possibility exists that another CPU may simultaneously update the location. First, the entire word containing the byte or bytes to be updated is loaded into a general register. Next, the updated value is computed and placed in another general register. Then the instruction CS is executed with the R_1 field designating the register that contains the original value and the R_3 field designating the register that contains the updated value. If condition code 0 is set, the update has been successful. If condition code 1 is set, the storage location no longer contains the original value, the update has not been successful, and the general register designated by the R_1 field of the CS instruction contains the new current value of the storage location. When condition code 1 is set, the program can repeat the procedure using the new current value.
   b. The instruction CS can be used for controlled sharing of a common storage area in a manner similar to that described in the programming note under TEST AND SET, but it provides the added capability of leaving a message when the common area is in use. To accomplish this, a word in storage can be used as a control word, with a zero value in the word indicating that the common area is not in use, a negative value indicating that the area is in use, and a nonzero positive value indicating that the common area is in use and that the value is the address of the most recent message added to the list. Thus, any number of programs desiring to seize the area can use CS to update the control word to indicate that the area is in use or to add messages to the list. The single program which has seized the area can also safely use CS to remove messages from the list.
3. The instruction CDS can be used in a manner similar to that described for CS. In addition, it has another use. Consider a chained list, with a control word used to address the first message in the list, as described in programming note 2b above. If multiple programs are permitted to add and delete messages by using CS, there is a possibility the list will be incorrectly updated. This would occur if, after one program has fetched the address of the most recent message in order to remove the message, another program removes the first two messages and then adds the first message back into the chain. The first program, on continuing, cannot easily detect that the list is changed. By increasing the size of the control word to a doubleword containing both the first message address and a word with a change number that is incremented for each modification of the list, and by using CDS to update both fields together, the possibility of the list being incorrectly updated is reduced to a negligible level. That is, an incorrect update can occur only if the first program is delayed while changes exactly equal in number to a multiple of 2^{32} take place *and* only if the last change places the original message address in the control word.
4. The instructions CS and CDS do not interlock against storage accesses by channels. Therefore, the instructions should not be used to update a location which is in an I/O input area, since the input data may be lost.
5. For the case of a condition-code setting of 1, the instructions CS and CDS may or may not, depending on the model, cause any of the following to occur for the second-operand location: a PER storage-alteration event may be recognized; a protection exception for storing may be recognized; and, provided no access exception exists, the change bit may be turned on.

COMPARE HALFWORD

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



The first operand is compared with the second operand, and the result is indicated in the condition code. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand is treated as a 32-bit signed binary integer.

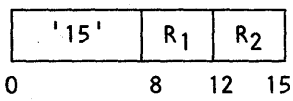
Resulting Condition Code:
0 Operands are equal
1 First operand is low
2 First operand is high
3 –

Program Exceptions:
Access (fetch, operand 2)

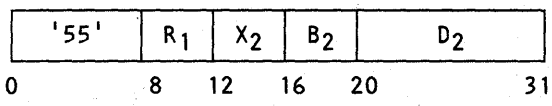
Programming Note
An example of the use of COMPARE HALFWORD is given in Appendix A.

COMPARE LOGICAL

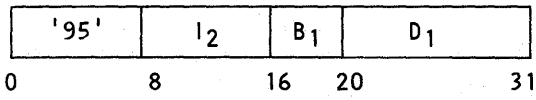
CLR R_1,R_2 [RR]



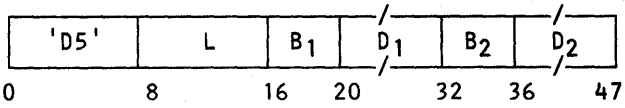
CL $R_1,D_2(X_2,B_2)$ [RX]



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



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



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

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the fields is reached. For CL and CLC, access exceptions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte.

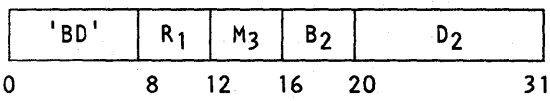
Resulting Condition Code:
0 Operands are equal
1 First operand is low
2 First operand is high
3 –

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

Programming Notes
1. Examples of the use of the COMPARE LOGICAL instructions are given in Appendix A.
2. The COMPARE LOGICAL instructions treat all bits of each operand alike as part of a field of unstructured logical data. For CLC, the comparison may extend to field lengths of 256 bytes.

COMPARE LOGICAL CHARACTERS UNDER MASK

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



The first operand is compared with the second operand under control of a mask, and the result is indicated in the condition code.

The contents of the M_3 field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of the general register designated by the R_1 field. The byte positions corresponding to ones in the mask are

considered as a contiguous field and are compared with the second operand. The second operand is a contiguous field in storage, starting at the second-operand address and equal in length to the number of ones in the mask. The bytes in the general register corresponding to zeros in the mask do not participate in the operation.

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the fields is reached.

When the mask is not zero, exceptions associated with storage-operand access are recognized for no more than the number of bytes specified by the mask. Access exceptions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte. When the mask is zero, access exceptions are recognized for one byte.

Resulting Condition Code:

0 Selected bytes are equal, or mask is zero
1 Selected field of first operand is low
2 Selected field of first operand is high
3 –

Program Exceptions:

Access (fetch, operand 2)

Programming Note

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

COMPARE LOGICAL LONG

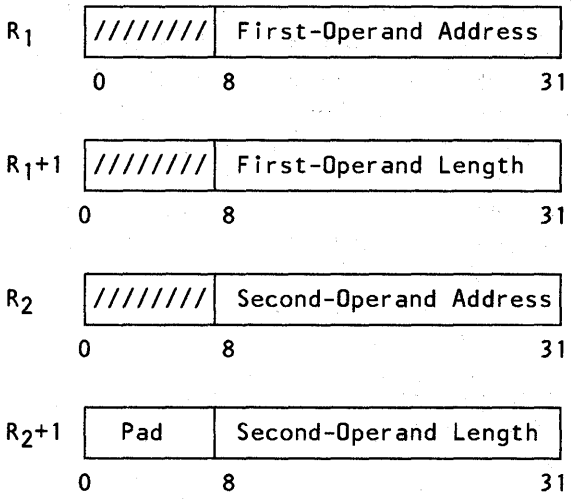
CLCL R_1,R_2 [RR]

The first operand is compared with the second operand, and the result is indicated in the condition code. The shorter operand is considered to be extended on the right with padding bytes.

The R_1 and R_2 fields each specify an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by bits 8-31 of the general registers specified by the R_1 and R_2 fields, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 8-31 of general registers R_1+1 and R_2+1, respectively. Bit positions 0-7 of register R_2+1 contain the padding byte. The contents of bit positions 0-7 of registers R_1, R_2, and R_1+1 are ignored.

Graphically, the contents of the registers just described are as follows:



The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the longer operand is reached. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of padding bytes.

If both operands are of zero length, the operands are considered to be equal.

The execution of the instruction is interruptible. When an interruption occurs, other than one that causes termination, the contents of registers R_1+1 and R_2+1 are decremented by the number of bytes compared, and the contents of registers R_1 and R_2 are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. The high-order bits which are not part of the address in registers R_1 and R_2 are set to zeros; the contents of the high-order byte of registers R_1+1 and R_2+1 remain unchanged; and the condition code is unpredictable. If the operation is interrupted after the shorter operand has been exhausted, the length field pertaining to the shorter operand is zero, and its address is updated accordingly.

If the operation ends because of an inequality, the address fields in registers R_1 and R_2 at completion identify the first unequal byte in each operand. The lengths in bit positions 8-31 of

registers R_1+1 and R_2+1 are decremented by the number of bytes that were equal, unless the inequality occurred with the padding byte, in which case the length field for the shorter operand is set to zero. The addresses in registers R_1 and R_2 are incremented by the amounts by which the corresponding length fields were reduced.

If the two operands, including the padding byte, if necessary, are equal, both length fields are made zero at completion, and the addresses are incremented by the corresponding operand-length values. The bits which are not part of the address in registers R_1 and R_2 are set to zeros, including the case when one or both of the initial length values are zero. The contents of bit positions 0-7 of registers R_1+1 and R_2+1 remain unchanged.

Access exceptions for the portion of a storage operand to the right of the first unequal byte may or may not be recognized. For operands longer than 2,048 bytes, access exceptions are not recognized more than 2,048 bytes beyond the byte being processed. Access exceptions are not indicated for locations more than 2,048 bytes beyond the first unequal byte.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

0 Operands are equal, or both have zero length
1 First operand is low
2 First operand is high
3 –

Program Exceptions:

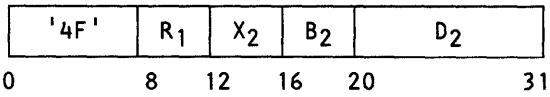
Access (fetch, operands 1 and 2)
Specification

Programming Notes

1. An example of the use of COMPARE LOGICAL LONG is given in Appendix A.
2. When the R_1 and R_2 fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, and, in the absence of dynamic modification of the operand area by another CPU or a channel, condition code 0 is set. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed.
3. Other programming notes concerning interruptible instructions are included in the section "Interruptible Instructions" in Chapter 5, "Program Execution."
4. Special precautions should be taken when COMPARE LOGICAL LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.

CONVERT TO BINARY

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



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

The second operand occupies eight bytes in storage and is treated as packed decimal data, as described in Chapter 8, "Decimal Instructions." It is checked for valid sign and digit codes, and a data exception is recognized when an invalid code is detected.

The result of the conversion is a 32-bit signed binary integer, which is placed in the general register specified by R_1. The maximum positive number that can be converted and still be contained in a 32-bit register is 2,147,483,647; the maximum negative number (the negative number with the greatest absolute value) that can be converted is –2,147,483,648. For any decimal number outside this range, the operation is completed by placing the 32 low-order bits of the binary result in the register, and a fixed-point-divide exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

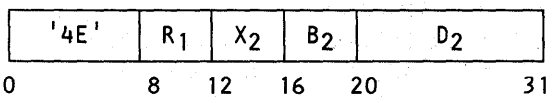
Access (fetch, operand 2)
Data
Fixed-Point Divide

Programming Notes

1. An example of the use of CONVERT TO BINARY is given in Appendix A.
2. When the second operand is negative, the result is in two's-complement notation.

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 at the second-operand location. The first operand is treated as a 32-bit signed binary integer.

The result occupies eight bytes in storage and is in the format for packed decimal data, as described in Chapter 8, "Decimal Instructions." The low-order four bits of the result represent the sign. A positive sign is encoded as 1100; a negative sign is encoded as 1101.

Condition Code: The code remains unchanged.

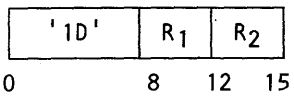
Program Exceptions:
Access (store, operand 2)

Programming Notes

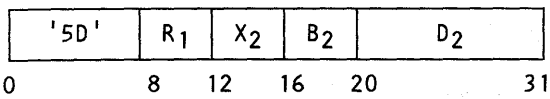
1. An example of the use of CONVERT TO DECIMAL is given in Appendix A.
2. The number to be converted is a 32-bit signed binary integer obtained from a general register. Since 15 decimal digits are available for the result, and the decimal equivalent of 31 bits requires at most 10 decimal digits, an overflow cannot occur.

DIVIDE

DR R_1,R_2 [RR]



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



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

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

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

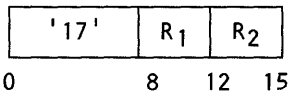
The sign of the quotient is determined by the rules of algebra. The remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive. When the magnitudes of the dividend and divisor are such that the quotient cannot be expressed by a 32-bit signed binary integer, a fixed-point-divide exception is recognized, and the operation is suppressed.

Condition Code: The code remains unchanged.

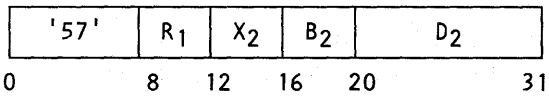
Program Exceptions:
Access (fetch, operand 2 of D only)
Fixed-Point Divide
Specification

EXCLUSIVE OR

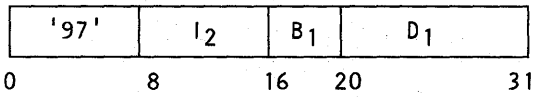
XR R_1,R_2 [RR]



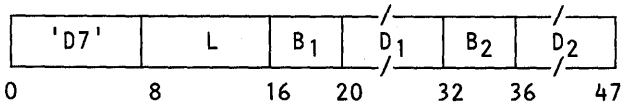
X $R_1,D_2(X_2,B_2)$ [RX]



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



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



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

The connective EXCLUSIVE OR is applied to the operands bit by bit. A bit position in the result is set to one if the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

For XC, each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

For XI, the first operand is one byte in length, and only one byte is stored.

Resulting Condition Code:

0 Result is zero
1 Result is not zero
2 –
3 –

Program Exceptions:

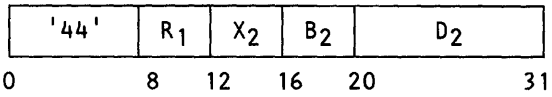
Access (fetch, operand 2, X and XC; fetch and store, operand 1, XI and XC)

Programming Notes

1. An example of the use of EXCLUSIVE OR is given in Appendix A.
2. The instruction EXCLUSIVE OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.
3. A field EXCLUSIVE-ORed with itself becomes all zeros.
4. For XR, the sequence A EXCLUSIVE-OR B, B EXCLUSIVE-OR A, A EXCLUSIVE-OR B results in the exchange of the contents of A and B without the use of an additional general register.
5. Accesses to the first operand of XI and XC consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, the instruction EXCLUSIVE OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for the instruction OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

EXECUTE

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



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

When the R_1 field is not zero, bits 8-15 of the instruction designated by the second-operand address are ORed with bits 24-31 of the register specified by R_1. The ORing does not change either the contents of the register specified by R_1 or the instruction in storage, and it is effective only for the interpretation of the instruction to be executed. When the R_1 field is zero, no ORing takes place.

The target instruction may be two, four, or six bytes in length. The execution and exception handling of the target instruction are exactly as if the target instruction were obtained in normal sequential operation, except for the instruction address and the instruction-length code.

The instruction address of the current PSW is increased by the length of EXECUTE. This updated address and the instruction-length code of EXECUTE are used, for example, as part of the link information when the target instruction is BRANCH AND LINK. When the target instruction is a successful branching instruction, the instruction address of the current PSW is replaced by the branch address specified by the target instruction.

When the target instruction is in turn an EXECUTE, an execute exception is recognized.

The effective address of EXECUTE must be even; otherwise, a specification exception is recognized. When the target instruction is two or three halfwords in length but can be executed without fetching its second or third halfword, it is unpredictable whether access exceptions are recognized for the unused halfwords. Access exceptions are not recognized for the second-operand address when the address is odd.

Condition Code: The code may be set by the target instruction.

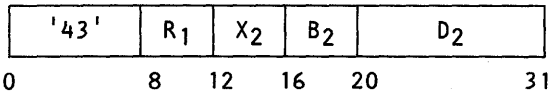
Program Exceptions:
Access (fetch, target instruction)
Execute
Specification

Programming Notes

1. An example of the use of EXECUTE is given in Appendix A.
2. The ORing of eight bits from the general register with the designated instruction permits indirect length, index, mask, immediate-data, and register specification.
3. The fetching of the target instruction is considered to be an instruction fetch for purposes of program-event recording and for purposes of reporting access exceptions.
4. An access or specification exception may be caused by EXECUTE or by the target instruction.
5. When an interruptible instruction is made the target of EXECUTE, the program normally should not designate any register updated by the interruptible instruction as the R_1, X_2, or B_2 register for EXECUTE, since on resumption of execution after an interruption, or if the instruction is refetched without an interruption, the updated values of these registers will be used in the execution of EXECUTE. Similarly, the program should normally not let the destination field of an interruptible instruction include the location of the EXECUTE, since the new contents of the location may be interpreted when resuming execution.

INSERT CHARACTER

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



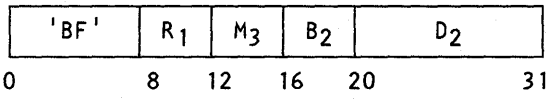
The byte at the second-operand location is inserted into bit positions 24-31 of the general register designated by the R_1 field. The remaining bits in the register remain unchanged.

Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

INSERT CHARACTERS UNDER MASK

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



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

The contents of the M_3 field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of the general register designated by the R_1 field. The byte positions corresponding to ones in the mask are filled, left to right, with bytes from successive storage locations beginning at the second-operand address. When the mask is not zero, the length of the second operand is equal to the number of ones in the mask. The bytes in the general register corresponding to zeros in the mask remain unchanged.

The resulting condition code is based on the mask and on the value of the bits inserted. When the mask is zero or when all inserted bits are zeros, the condition code is set to 0. When the inserted bits are not all zeros, the code is set according to the leftmost bit of the storage operand: if this bit is one, the code is set to 1; if this bit is zero, the code is set to 2.

When the mask is not zero, exceptions associated with storage-operand access are recognized only for the number of bytes specified by the mask. When the mask is zero, access exceptions are recognized for one byte.

Resulting Condition Code:

0 All inserted bits are zeros, or mask is zero
1 Leftmost bit of the inserted field is one
2 Leftmost bit of the inserted field is zero, and not all inserted bits are zeros
3 –

Program Exceptions:
Access (fetch, operand 2)

Programming Notes

1. Examples of the use of INSERT CHARACTERS UNDER MASK are given in Appendix A.
2. The condition code for INSERT CHARACTERS UNDER MASK (ICM) is

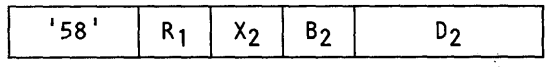
defined such that, when the mask is 1111, the instruction causes the same condition code to be set as for LOAD AND TEST. Thus, the instruction may be used as a storage-to-register load-and-test operation.

3. An ICM instruction with a mask of 1111 or 0001 performs a function similar to that of a LOAD (L) or INSERT CHARACTER (IC), respectively, with the exception of the condition-code setting. However, the performance of ICM may be slower.

LOAD

LR R_1,R_2 [RR]

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



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

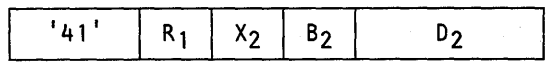
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2 of L only)

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

LOAD ADDRESS

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



The address specified by the X_2, B_2, and D_2 fields is placed in bit positions 8-31 of the general register specified by the R_1 field. Bits 0-7 of the register are set to zeros. The address computation follows the rules for address arithmetic.

No storage references for operands take place, and the address is not inspected for access exceptions.

Condition Code: The code remains unchanged.

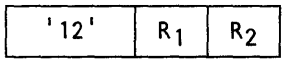
Program Exceptions: None.

Programming Notes

1. An example of the use of the LOAD ADDRESS instruction is given in Appendix A.
2. The same general register may be specified by the R_1, X_2, and B_2 fields, 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 register 0, by the contents of the D_2 field of the instruction. The register to be incremented should be specified by R_1 and by either X_2 (with B_2 set to zero) or B_2 (with X_2 set to zero).

LOAD AND TEST

LTR R_1,R_2 [RR]



The second operand is placed unchanged in the first-operand location, and the sign and magnitude of the second operand, treated as a 32-bit signed binary integer, are indicated in the condition code.

Resulting Condition Code:

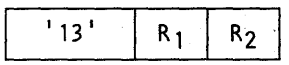
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 –

Program Exceptions: None.

Programming Note
When the R_1 and R_2 fields designate the same register, the operation is equivalent to a test without data movement.

LOAD COMPLEMENT

LCR R_1,R_2 [RR]



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

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

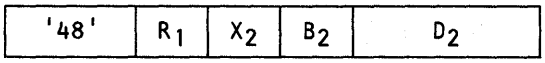
Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:
Fixed-Point Overflow

Programming Note
The operation complements all numbers. Zero and the maximum negative number remain unchanged. An overflow condition occurs when the maximum negative number is complemented.

LOAD HALFWORD

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



The second operand is extended to a 32-bit signed binary integer and placed in the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed binary integer. The second operand is extended by propagating the sign-bit value through the 16 high-order bit positions.

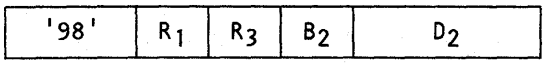
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

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

LOAD MULTIPLE

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



The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is loaded from storage beginning at the location designated by the second-operand address and continuing through as many locations as needed.

The general registers are loaded in the ascending order of their register numbers, starting with the register specified by R_1 and continuing up to and including the register specified by R_3, with register 0 following register 15.

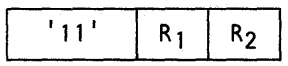
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

Programming Note
All combinations of register numbers specified by R_1 and R_3 are valid. When the register numbers are equal, only four bytes are transmitted. When the number specified by R_3 is less than the number specified by R_1, the register numbers wrap around from 15 to 0.

LOAD NEGATIVE

LNR R_1,R_2 [RR]



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

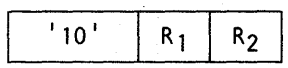
Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 –
3 –

Program Exceptions: None.

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

LOAD POSITIVE

LPR R_1,R_2 [RR]



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

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

Resulting Condition Code:

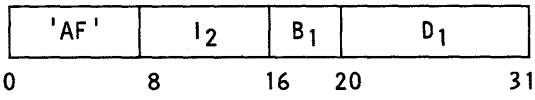
0 Result is zero
1 –
2 Result is greater than zero
3 Overflow

Program Exceptions:
Fixed-Point Overflow

Programming Note
The operation complements negative numbers; positive numbers and zero remain unchanged. An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged.

MONITOR CALL

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



A program interruption is caused if the appropriate monitor-mask bit in control register 8 is one.

The monitor-mask bits are in bit positions 16-31 of control register 8, which correspond to monitor classes 0-15, respectively.

Bit positions 12-15 in the I_2 field contain a binary number specifying one of 16 monitoring classes. When the monitor-mask bit corresponding to the class specified by the I_2 field is one, a monitor-event program interruption occurs. The contents of the I_2 field are stored at location 149, with zeros stored at location 148. Bit 9 of the program-interruption code is set to one.

The first-operand address is not used to address data; instead, the address specified by the B_1 and D_1 fields forms the monitor code, which is placed in the word at location 156. Address computation follows the rules of address arithmetic; bits 0-7 are set to zeros.

When the monitor-mask bit corresponding to the class specified by bits 12-15 of the instruction is zero, no interruption occurs, and the instruction is executed as a no-operation.

Bit positions 8-11 of the instruction must contain zeros; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

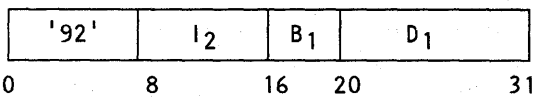
Program Exceptions:
Monitor Event
Specification

Programming Notes

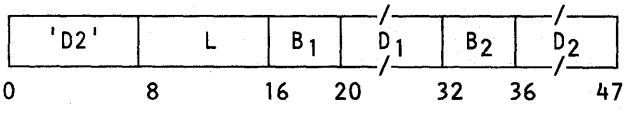
1. The MONITOR CALL instruction provides the capability for passing control to a monitoring program when selected points are reached in the monitored program. This is accomplished by implanting MONITOR CALL instructions at the desired points in the monitored program. This function may be useful in performing various measurement functions; specifically, tracing information can be generated indicating which programs were executed, counting information can be generated indicating how often particular programs were used, and timing information can be generated indicating how long a particular program required for execution.
2. The monitor masks provide a means of disallowing all interruptions due to MONITOR CALL or allowing monitoring for all or selected classes.
3. The monitor code provides a means of associating descriptive information, in addition to the class number, with each MONITOR CALL instruction. Without the use of a base register, up to 4,096 distinct monitor codes can be associated with a monitoring interruption. With the base register designated by a nonzero value in the B_1 field, each monitoring interruption can be identified by a 24-bit code.

MOVE

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



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



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

For MVC, each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

For MVI, the first operand is one byte in length, and only one byte is stored.

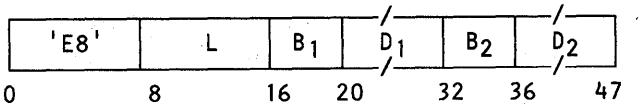
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2 of MVC; store, operand 1, MVI and MVC)

Programming Notes
1. Examples of the use of the MOVE instructions are given in Appendix A.
2. It is possible to propagate one byte through an entire field by having the first operand start one byte to the right of the second operand.

MOVE INVERSE

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



The second operand is placed in the first-operand location with the left-to-right sequence of the bytes inverted.

The first-operand address designates the leftmost byte of the first operand. The second-operand address designates the rightmost byte of the second operand. Both operands have the same length.

The result is obtained as if the second operand were processed from right to left and the first operand from left to right. The second operand may wrap around from location 0 to location 16,777,215. The first operand may wrap around from location 16,777,215 to location 0.

When the operands overlap by more than one byte, the contents of the overlapped portion of the result field are unpredictable.

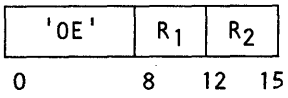
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2; store, operand 1)

Programming Notes
1. The contents of each byte moved remain unchanged.
2. MOVE INVERSE is the only SS-format instruction for which the second-operand address designates the rightmost, instead of the leftmost, byte of the second operand.

MOVE LONG

MVCL R_1,R_2 [RR]

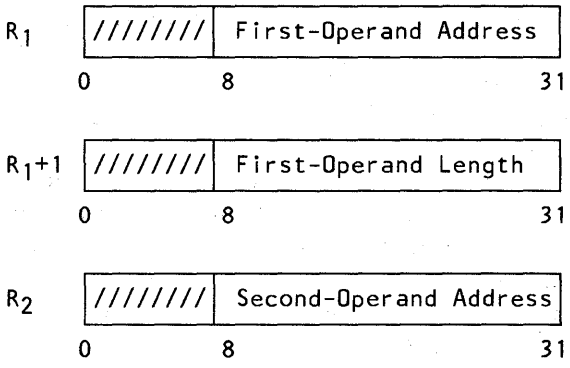


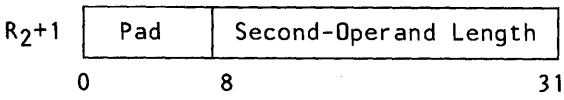
The second operand is placed in the first-operand location, provided overlapping of operand locations does not affect the final contents of the first-operand location. The remaining rightmost byte positions, if any, of the first-operand location are filled with padding bytes.

The R_1 and R_2 fields each specify an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by bits 8-31 of the general registers specified by the R_1 and R_2 fields, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 8-31 of general registers R_1+1 and R_2+1, respectively. Bit positions 0-7 of register R_2+1 contain the padding byte. The contents of bit positions 0-7 of registers R_1, R_2, and R_1+1 are ignored.

Graphically, the contents of the registers just described are as follows:





The movement starts at the left end of both fields and proceeds to the right. The operation is ended when the number of bytes specified by bit positions 8-31 of register R_1+1 have been moved into the first-operand location. If the second operand is shorter than the first operand, the remaining rightmost bytes of the first-operand location are filled with the padding byte.

As part of the execution of the instruction, the values of the two length fields are compared for the setting of the condition code, and a check is made for destructive overlap of the operands. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it. When the operands overlap destructively, no movement takes place, and condition code 3 is set.

Operands do not overlap destructively, and movement is performed, if the leftmost byte of the first operand does not coincide with any of the second-operand bytes participating in the operation other than the leftmost byte of the second operand. When an operand wraps around from location 16,777,215 to location 0, operand bytes in locations up to and including 16,777,215 are considered to be to the left of bytes in locations from 0 up.

When the length specified by bit positions 8-31 of register R_1+1 is zero, no movement takes place, and condition code 0 or 1 is set to indicate the relative values of the lengths.

The execution of the instruction is interruptible. When an interruption occurs other than one that causes termination, the contents of registers R_1+1 and R_2+1 are decremented by the number of bytes moved, and the contents of register R_1 and R_2 are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. The high-order bits which are not part of the address in registers R_1 and R_2 are set to zeros; the contents of the high-order byte of registers R_1+1 and R_2+1 remain unchanged; and the condition code is unpredictable. If the operation is interrupted during padding, the length field in register R_2+1 is 0, the address in register R_2 is incremented by the original contents of register R_2+1 , and registers R_1 and R_1+1 reflect the extent of the padding operation.

When the first-operand location includes the location of the instruction, the instruction may be refetched from storage and reinterpreted even in the absence of an interruption during execution. The exact point in the execution at which such a refetch occurs is unpredictable.

As viewed by channels and other CPUs, that portion of the first operand which is filled with the padding byte is not necessarily stored into in a left-to-right direction and may appear to be stored more than once.

At the completion of the operation, the length in register R_1+1 is decremented by the number of bytes stored at the first-operand location, and the address in register R_1 is incremented by the same amount. The length in register R_2+1 is decremented by the number of bytes moved out of the second-operand location, and the address in register R_2 is incremented by the same amount. The bits which are not part of the address in registers R_1 and R_2 are set to zeros, including the case when one or both of the original length values are zeros or when condition code 3 is set. The contents of bit positions 0-7 of registers R_1+1 and R_2+1 remain unchanged.

When condition code 3 is set, no exceptions associated with operand access are recognized. When the length of an operand is zero, no access exceptions for that operand are recognized. Similarly, when the second operand is longer than the first operand, access exceptions are not recognized for the part of the second-operand field that is in excess of the first-operand field. For operands longer than 2,048 bytes, access exceptions are not recognized for locations more than 2,048 bytes beyond the current location being processed. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Also, when the R_1 field is odd, PER storage alteration is not recognized, and no change bits are set.

Resulting Condition Code:

0 First-operand and second-operand lengths are equal
1 First-operand length is low
2 First-operand length is high
3 No movement performed because of destructive overlap

Program Exceptions:

Access (fetch, operand 2; store, operand 1)
Specification

Programming Notes

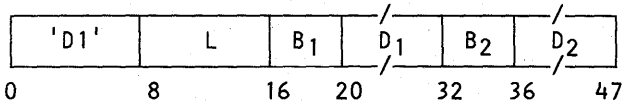
1. The instruction MOVE LONG may be used for clearing storage by setting the padding byte to zero and the second-operand length to zero. However, the stores associated with this clearing may be multiple-access stores and should not be used to clear an area if the possibility exists that a channel or another CPU will attempt to access and use the area as soon as it appears to be zero.
2. The program should avoid specification of a length for either operand which would result in an addressing exception. Addressing (and also protection) exceptions may result in termination of the entire operation, not just the current unit of operation. The termination may be such that the contents of all result fields are unpredictable; in the case of MVCL, this includes the condition code and the two even-odd general-register pairs, as well as the first-operand location in main storage. The following are situations that have actually occurred on one or more models.
   a. When a protection exception occurs on a 2,048-byte block of a first operand which is several blocks in length, stores to the protected block are suppressed. However, the move continues into the subsequent blocks of the first operand, which are not protected. Similarly, in the case of reconfigurable storage, an addressing exception on a block does not necessarily suppress processing of subsequent blocks which are addressable.
   b. The model may update the general registers only when an I/O interruption occurs or when a program interruption occurs which is required to nullify or suppress. Thus, if after a move into several blocks of the first operand, an addressing or protection exception occurs, the registers remain unchanged.
3. When the first-operand length is zero, the operation consists in setting the condition code and setting the high-order bytes of registers R_1 and R_2 to zero.
4. When the contents of the R_1 and R_2 fields are the same, the operation proceeds the same way as when two distinct pairs of registers having the same contents are specified. Condition code 0 is set.
5. The following is a detailed description of those cases in which movement takes place, that is, where destructive overlap does not exist. Depending on whether the second operand wraps around from location 16,777,215 to location 0, movement takes place in the following cases:
   a. When the second operand does not wrap around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, *or* if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.
   b. When the second operand wraps around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, *and* if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.

   The rightmost second-operand byte is determined by using the smaller of the first-operand and second-operand lengths.

   When the second-operand length is one or zero, destructive overlap cannot exist.
6. Special precautions must be taken if MOVE LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.
7. Since the execution of MOVE LONG is interruptible, the instruction cannot be used for situations where the program must rely on uninterrupted execution of the instruction or on the interval timer not being updated during the execution of the instruction. Similarly, the program should normally not let the first operand of MOVE LONG include the location of the instruction since the new contents of the location may be interpreted for a resumption after an interruption, or the instruction may be refetched without an interruption.
8. Further programming notes concerning interruptible instructions are included in the section "Interruptible Instructions" in Chapter 5, "Program Execution."

MOVE NUMERICS

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



The rightmost four bits of each byte in the second operand are placed in the rightmost bit positions of the corresponding bytes in the first operand. The leftmost four bits of each byte in the first operand remain unchanged.

Each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

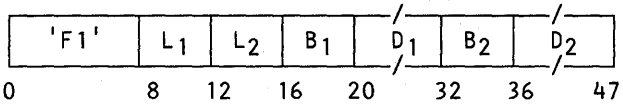
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes
1. An example of the use of MOVE NUMERICS is given in Appendix A.
2. MVN moves the numeric portion of a decimal-data field that is in the zoned format. The zoned-decimal format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.
3. Accesses to the first operand of MVN consist in fetching the rightmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for the instruction OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

MOVE WITH OFFSET

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



The second operand is placed to the left of and adjacent to the rightmost four bits of the first operand.

The rightmost four bits of the first operand are attached as the rightmost bits to the second operand, the second operand bits are offset by four bit positions, and the result is placed in the first-operand location.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand is too short to contain all of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand bytes are fetched. The left digit of each second-operand byte remains available for the next result byte and is not refetched.

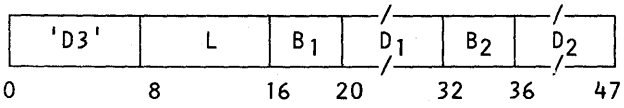
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes
1. An example of the use of MOVE WITH OFFSET is given in Appendix A.
2. Access to the rightmost byte of the first operand of MVO consists in fetching the rightmost four bits and subsequently storing the updated value of this byte. These fetch and store accesses to the rightmost byte of the first operand do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for the instruction OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.
3. MVO may be used to shift packed decimal data by an odd number of digit positions. The packed-decimal format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes. In many cases however, the instruction SHIFT AND ROUND DECIMAL may be more convenient to use.

MOVE ZONES

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



The leftmost four bits of each byte in the second operand are placed in the leftmost four bit positions of the corresponding bytes in the first operand. The rightmost four bits of each byte in the first operand remain unchanged.

Each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

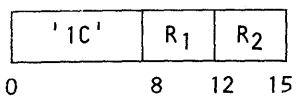
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2; fetch and store, operand 1)

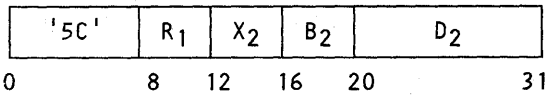
Programming Notes
1. An example of the use of MOVE ZONES is given in Appendix A.
2. MVZ moves the zoned portion of a decimal field in the zoned format. The zoned format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.
3. Accesses to the first operand of MVZ consist in fetching the leftmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for the instruction OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

MULTIPLY

MR R_1,R_2 [RR]



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



The second word of the first operand (multiplicand) is multiplied by the second operand (multiplier), and the doubleword product is placed at the first-operand location.

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

Both the multiplicand and multiplier are treated as 32-bit signed binary integers. The multiplicand is taken from the odd-numbered register of the pair specified by the R_1 field. The contents of the even-numbered register are ignored. The product is a 64-bit signed binary integer, which replaces the contents of the even-odd pair of general registers specified by the R_1 field. An overflow cannot occur.

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

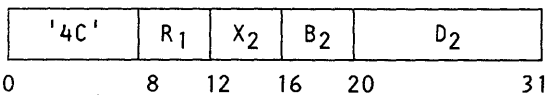
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2 of M only)
Specification

Programming Notes
1. An example of the use of MULTIPLY is given in Appendix A.
2. The significant part of the product usually occupies 62 bits or fewer. Only when two maximum negative numbers are multiplied are 63 significant product bits formed.

MULTIPLY HALFWORD

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



The first operand (multiplicand) is multiplied by the second operand (multiplier), and the product is placed at the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed binary integer.

The multiplicand is treated as a 32-bit signed binary integer and is replaced by the low-order 32 bits of the signed-binary-integer 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.

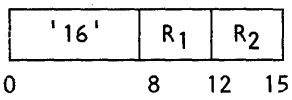
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

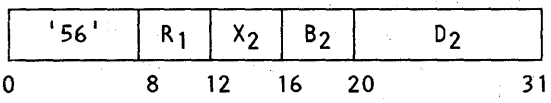
Programming Notes
1. An example of the use of MULTIPLY HALFWORD is given in Appendix A.
2. The significant part of the product usually occupies 46 bits or fewer. Only when two maximum negative numbers are multiplied are 47 significant product bits formed. Since the low-order 32 bits of the product are 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. For a negative product, the 32 bits placed in register R_1 are the low-order part of the product in two's-complement notation.

OR

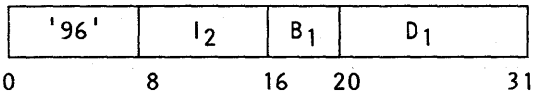
OR R_1,R_2 [RR]



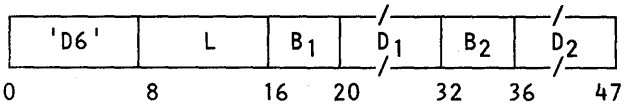
O $R_1,D_2(X_2,B_2)$ [RX]



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



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



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

The connective OR is applied to the operands bit by bit. A bit position in the result is set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero.

For OC, each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

For OI, the first operand is only one byte in length, and only one byte is stored.

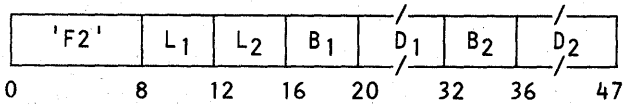
Resulting Condition Code:
0 Result is zero
1 Result is not zero
2 –
3 –

Program Exceptions:
Access (fetch, operand 2, O and OC; fetch and store, operand 1, OI and OC)

Programming Notes
1. Examples of the use of the OR instructions are given in Appendix A.
2. The instruction OR may be used to set a bit to one.
3. Accesses to the first operand of OI and OC consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, the instruction OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

PACK

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



The format of the second operand is changed from zoned to packed, and the result is placed in the first-operand location. The zoned and packed formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the zoned format. The numerics are treated as digits. All zones are ignored, except the zone in the rightmost byte, which is treated as a sign.

The sign and digits are moved unchanged to the first operand and are not checked for valid codes. The sign is placed in the rightmost four bit positions of the rightmost byte of the result field, and the digits are placed adjacent to the sign and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if each result byte were stored immediately after the necessary operand bytes are fetched. Two second-operand bytes are needed for each result byte, except for the rightmost byte of the result field, which requires only the rightmost second-operand byte.

Condition Code: The code remains unchanged.

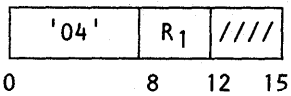
Program Exceptions:
Access (fetch, operand 2; store, operand 1)

Programming Notes

1. An example of the use of PACK is given in Appendix A.
2. The PACK instruction may be used to interchange the two hexadecimal digits in one byte by specifying a zero in the L_1 and L_2 fields and the same address for both operands.
3. To remove the zones of all bytes of a field, including the rightmost byte, both operands must be extended on the right with a dummy byte, which subsequently is ignored in the result field.

SET PROGRAM MASK

SPM R_1 [RR]



The contents of the general register specified by the R_1 field are used to set the condition code and the program mask of the current PSW. Bits 12-15 of the instruction are ignored.

Bits 2 and 3 of the register specified by the R_1 field replace the condition code, and bits 4-7 replace the program mask. Bits 0, 1, and 8-31 of the register specified by the R_1 field are ignored.

Resulting Condition Code:

0 Bit 2 is zero, and bit 3 is zero
1 Bit 2 is zero, and bit 3 is one
2 Bit 2 is one, and bit 3 is zero
3 Bit 2 is one, and bit 3 is one

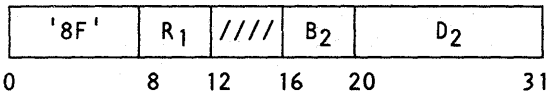
Program Exceptions: None.

Programming Notes

1. Bits 2-7 of the general register may have been loaded from the PSW by BRANCH AND LINK.
2. The instruction permits setting of the condition code and the mask bits in either the problem or supervisor state.
3. The program should take into consideration that the setting of the program mask can have a significant effect on subsequent execution of the program. Not only do the four mask bits control whether the corresponding interruptions occur, but the exponent-underflow and significance masks also determine the result which is obtained.

SHIFT LEFT DOUBLE

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



The double-length numeric part of the first operand is shifted left the number of bits specified by the

second-operand address. Bits 12-15 of the instruction are ignored.

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

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

The first operand is treated as a 64-bit signed binary integer. The sign position of the even register remains unchanged. The leftmost position of the odd register contains a numeric bit, which participates in the shift in the same manner as the other numeric bits. Zeros are supplied to the vacated register positions on the right.

If one or more bits unlike the sign bit are 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.

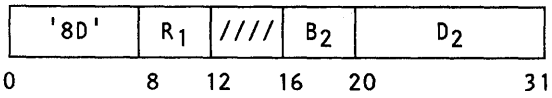
Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:
Fixed-Point Overflow
Specification

Programming Notes
1. An example of the use of SHIFT LEFT DOUBLE is given in Appendix A.
2. The eight shift instructions provide the following three pairs of alternatives: left or right, single or double, and signed or logical. The signed shifts differ from the logical shifts in that, in the signed shifts, overflow is recognized, the condition code is set, and the leftmost bit participates as a sign.
3. A zero shift amount in the two signed double-shift operations provides a double-length sign and magnitude test.
4. The base register participating in the generation of the second-operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

SHIFT LEFT DOUBLE LOGICAL

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



The double-length first operand is shifted left the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

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

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

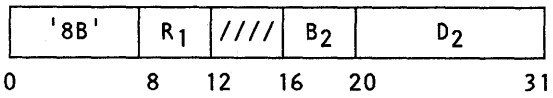
All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 0 of the even-numbered register are not inspected and are lost. Zeros are supplied to the vacated register positions on the right.

Condition Code: The code remains unchanged.

Program Exceptions:
Specification

SHIFT LEFT SINGLE

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



The numeric part of the first operand is shifted left the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

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

The first operand is treated as a 32-bit signed binary integer. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the left shift. Zeros are supplied to the vacated register positions on the right.

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

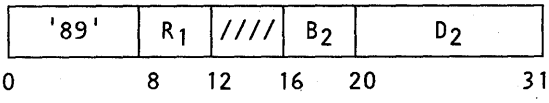
Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:
Fixed-Point Overflow

Programming Notes
1. An example of the use of SHIFT LEFT SINGLE is given in Appendix A.
2. For numbers with an absolute value of less than 2^{30}, a left shift of one bit position is equivalent to multiplying the number by two.
3. Shift amounts from 31 to 63 cause the entire numeric part to be shifted out of the register, leaving a result of the maximum negative number or zero, depending on whether or not the initial contents were negative.

SHIFT LEFT SINGLE LOGICAL

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



The first operand is shifted left the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

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

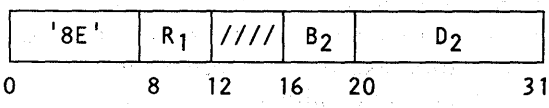
All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 0 are not inspected and are lost. Zeros are supplied to the vacated register positions on the right.

Condition Code: The code remains unchanged.

Program Exceptions: None.

SHIFT RIGHT DOUBLE

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



The double-length numeric part of the first operand is shifted right the number of places specified by the second-operand address. Bits 12-15 of the instruction are ignored.

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

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

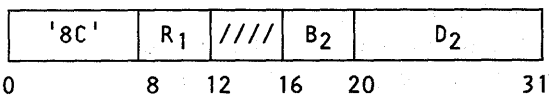
The first operand is treated as a 64-bit signed binary integer. The sign position of the even register remains unchanged. The leftmost position of the odd register contains a numeric bit, which participates in the shift in the same manner as the other numeric bits. Bits shifted out of bit position 31 of the odd-numbered register are not inspected and are lost. Bits equal to the sign are supplied to the vacated register positions on the left.

Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 –

Program Exceptions:
Specification

SHIFT RIGHT DOUBLE LOGICAL

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



The double-length first operand is shifted right the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The R_1 field of the instruction specifies an even-odd pair of general registers and must

designate an even-numbered register. When R_1 is odd, a specification exception is recognized.

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

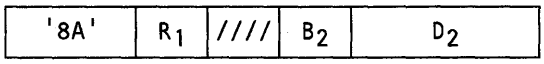
All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 31 of the odd-numbered register are not inspected and are lost. Zeros are supplied to the vacated register positions on the left.

Condition Code: The code remains unchanged.

Program Exceptions:
Specification

SHIFT RIGHT SINGLE

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



The numeric part of the first operand is shifted right the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

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

The first operand is treated as a 32-bit signed binary integer. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the right shift. Bits shifted out of bit position 31 are not inspected and are lost. Bits equal to the sign are supplied to the vacated register positions on the left.

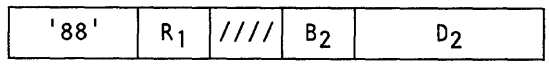
Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 –

Program Exceptions: None.

Programming Notes
1. A right shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the result is equivalent to dividing the number by 2. When an odd number is shifted right one position, the result is equivalent to dividing the *next lower* number by 2. For example, +5 shifted right by one bit position yields +2, whereas –5 yields –3.
2. Shift amounts from 31 to 63 cause the entire numeric part to be shifted out of the register, leaving a result of –1 or zero, depending on whether or not the initial contents were negative.

SHIFT RIGHT SINGLE LOGICAL

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



The first operand is shifted right the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

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

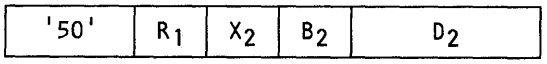
All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 31 are not inspected and are lost. Zeros are supplied to the vacated register positions on the left.

Condition Code: The code remains unchanged.

Program Exceptions: None.

STORE

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



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

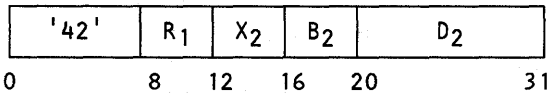
The 32 bits in the general register are placed unchanged at the second-operand location.

Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

STORE CHARACTER

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



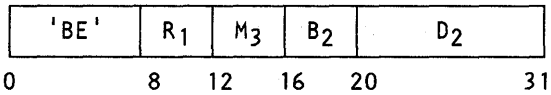
Bits 24-31 of the general register designated by the R_1 field are placed unchanged at the second-operand location. The second operand is one byte in length.

Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

STORE CHARACTERS UNDER MASK

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



Bytes selected from the first operand under control of a mask are placed in contiguous byte locations beginning at the second-operand address.

The contents of the M_3 field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of the general register designated by the R_1 field. The bytes corresponding to ones in the mask are placed in the same order in successive and contiguous storage locations beginning at the second-operand address. When the mask is not zero, the length of the second operand is equal to the number of ones in the mask. The contents of the general register remain unchanged.

When the mask is not zero, exceptions associated with storage-operand accesses are recognized only for the number of bytes specified by the mask.

When the mask is zero, the single byte designated by the second-operand address remains unchanged; however, on some models, the value may be fetched and subsequently stored back at the same storage location. No access by another CPU is permitted to the location designated by the second-operand address between the moment that the value is fetched and the value is stored.

Condition Code: The code remains unchanged.

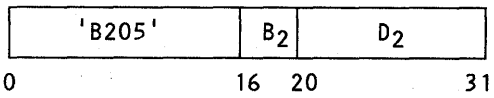
Program Exceptions:
Access (store, operand 2)

Programming Notes

1. An example of the use of STORE CHARACTERS UNDER MASK is given in Appendix A.
2. STCM with a mask of 0111 may be used to store a three-byte address, for example, in modifying the address in a CCW.
3. STCM with a mask of 1111, 0011, or 0001 performs the same function as STORE (ST), STORE HALFWORD (STH), or STORE CHARACTER (STC), respectively. However, on most models, the performance of STCM will be slower.
4. Using STCM with a zero mask should be avoided since this instruction, depending on the model, may perform a fetch and store of the single byte specified by the second-operand address. This access is not interlocked against accesses by channels. In addition, it may cause any of the following to occur for the byte specified by the second-operand address: a PER storage-alteration event may be recognized; access exceptions may be recognized; and, provided no access exceptions exist, the change bit may be turned on.

STORE CLOCK

STCK $D_2(B_2)$ [S]



The current value of the time-of-day clock is stored at the eight-byte field designated by the second-operand address, provided the clock is in the set or not-set state.

Zeros are stored for the rightmost bit positions that are not provided by the clock.

Zeros are stored at the operand location when the clock is in the error state or in the not-operational state.

The quality of the clock value stored by the instruction is indicated by the resultant condition-code setting.

A serialization function is performed before the value of the clock is fetched and again after the value is placed in storage. CPU operation is delayed until all previous accesses by this CPU to

storage have been completed, as observed by channels and other CPUs, and then the value of the clock is fetched. No subsequent instructions or their operands are fetched by this CPU until the clock value has been placed in storage, as observed by channels and CPUs.

Resulting Condition Code:

0 Clock in set state
1 Clock in not-set state
2 Clock in error state
3 Clock in not-operational state

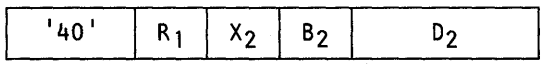
Program Exceptions:
Access (store, operand 2)

Programming Notes

1. Bit position 31 of the clock is incremented every 1.048576 seconds; hence, for timing applications involving human responses, the high-order clock word may provide sufficient resolution.
2. Condition code 0 normally indicates that the clock has been set by the control program. Accordingly, the value may be used in elapsed-time measurements and as a valid time-of-day and calendar indication. Condition code 1 indicates that the clock value is the elapsed time since the power for the clock was turned on. In this case the value may be used in elapsed-time measurements but is not a valid time-of-day indication. Condition codes 2 and 3 mean that the value provided by STORE CLOCK cannot be used for time measurement or indication.
3. If a problem program written for the ECPS:VSE mode is to be run also on a model of System/370, then the program should take into account the fact that, on a model of System/370, the value stored when the condition code is 2 or 3 is not necessarily zero.

STORE HALFWORD

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



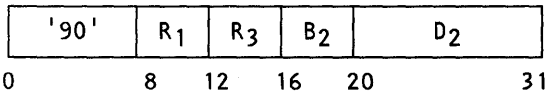
Bits 16-31 of the general register designated by the R_1 field are placed unchanged at the second-operand location. The second operand is two bytes in length.

Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

STORE MULTIPLE

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



The contents of the set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 are placed in the storage area beginning at the location designated by the second-operand address and continuing through as many locations as needed.

The general registers are stored in the ascending order of register numbers, starting with the register specified by R_1 and continuing up to and including the register specified by R_3, with register 0 following register 15.

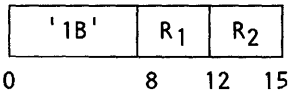
Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

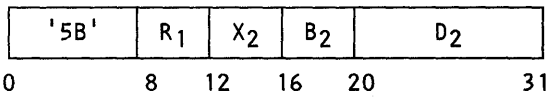
Programming Note
An example of the use of STORE MULTIPLE is given in Appendix A.

SUBTRACT

SR R_1,R_2 [RR]



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



The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The operands and the difference are treated as 32-bit signed binary integers.

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

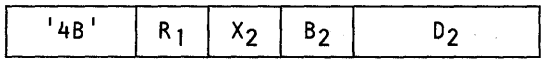
Resulting Condition Code:
0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Exceptions:
Access (fetch, operand 2 of S only)
Fixed-Point Overflow

Programming Notes
1. When, in the RR format, the R_1 and R_2 fields designate the same register, subtracting is equivalent to clearing the register.
2. Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

SUBTRACT HALFWORD

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



The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand and the difference are treated as 32-bit signed binary integers.

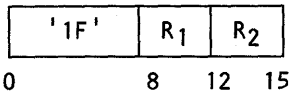
An overflow causes a program interruption when the fixed-point-overflow mask bit is one.

Resulting Condition Code:
0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

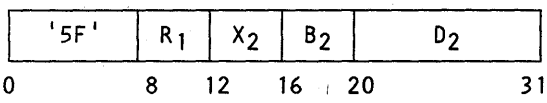
Program Exceptions:
Access (fetch, operand 2)
Fixed-Point Overflow

SUBTRACT LOGICAL

SLR R_1,R_2 [RR]



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



The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The operands and the difference are treated as 32-bit unsigned binary integers.

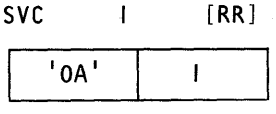
Resulting Condition Code:
0 –
1 Difference is not zero, with no carry
2 Difference is zero, with carry
3 Difference is not zero, with carry

Program Exceptions:
Access (fetch, operand 2 of SL only)

Programming Notes
1. Logical subtraction is performed by adding the one's complement of the second operand and a low-order one to the first operand. The use of the one's complement and the low-order one instead of the two's complement of the second operand results in a carry when subtracting zero.
2. SUBTRACT LOGICAL differs from SUBTRACT only in the meaning of the condition code and in the absence of the interruption for overflow.
3. A zero difference is always accompanied by a carry out of the high-order bit position.
4. The condition-code setting for SUBTRACT LOGICAL can also be interpreted as indicating the presence and absence of a borrow, as follows:
   1 Difference is not zero, with borrow
   2 Difference is zero, with no borrow
   3 Difference is not zero, with no borrow

SUPERVISOR CALL

SVC I [RR]



The instruction causes a supervisor-call interruption, with the I field of the instruction providing the interruption code.

Bits 8-15 of the instruction, with eight high-order zeros appended, are placed in the supervisor-call interruption code that is stored in the course of the interruption. See "Supervisor-Call Interruption" in Chapter 6, "Interruptions."

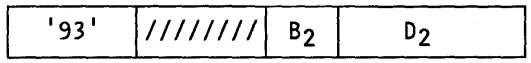
A serialization function is performed. CPU operation is delayed until all previous storage accesses by this CPU to storage have been completed, as observed by channels and and other CPUs. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed.

Condition Code: The code remains unchanged and is saved as part of the old PSW. A new condition code is loaded as part of the supervisor-call interruption.

Program Exceptions: None.

TEST AND SET

TS $D_2(B_2)$ [S]



The leftmost bit (bit position 0) of the byte located at the second-operand address is used to set the condition code, and then the byte is set to all ones. Bits 8-15 of the instruction are ignored.

The byte in storage is set to all ones as it is fetched for the testing of bit position 0. No access by another CPU to this location is permitted between the moment of fetching and the moment of storing all ones.

A serialization function is performed before the byte is fetched and again after the storing of all ones. CPU operation is delayed until all previous accesses by this CPU to storage have been completed, as observed by channels and other CPUs, and then the byte is fetched. No subsequent instructions or their operands are accessed by this CPU until the all-ones value has been placed in storage, as observed by channels and other CPUs.

Resulting Condition Code:

0 Leftmost bit of byte specified was zero
1 Leftmost bit of byte specified was one
2 –
3 –

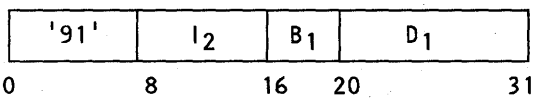
Program Exceptions:
Access (fetch and store, operand 2)

Programming Notes

1. TEST AND SET may 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 condition code 0, the area is available for use; if it sets condition code 1, the area cannot be used. Because TEST AND SET permits no other CPU 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 testing the byte before the first program is able to set it.
2. It should be noted that TEST AND SET does not interlock against storage accesses by channels.

TEST UNDER MASK

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



A mask is used to select bits of the first operand, and the result is indicated in the condition code.

The byte of immediate data, I_2, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the byte in storage designated by the first-operand address.

A mask bit of one indicates that the storage bit is to be tested. When the mask bit is zero, the storage bit is ignored. When all storage bits thus selected are zero, condition code 0 is set. Condition code 0 is also set when the mask is all zeros. When the selected bits are all ones, condition code 3 is set; otherwise, the code is set to 1.

Access exceptions associated with the storage operand are recognized for one byte even when the mask is all zeros.

Resulting Condition Code:

0 Selected bits all zeros; or the mask is all zeros
1 Selected bits mixed zeros and ones
2 –
3 Selected bits all ones

Program Exceptions:

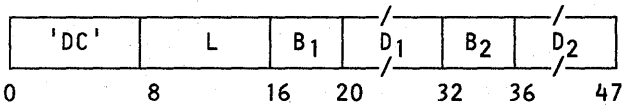
Access (fetch, operand 1)

Programming Note

An example of the use of TEST UNDER MASK is given in Appendix A.

TRANSLATE

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



The bytes of the first operand are used as eight-bit arguments to reference a list designated by the second-operand address. Each function byte selected from the list replaces the corresponding argument in the first operand.

The L field designates the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding left to right. Each argument byte is added to the initial second-operand address. The addition is performed following the rules for address arithmetic, with the argument byte treated as an eight-bit unsigned binary integer and extended with high-order zeros. The sum is used as the address of the function byte, which then replaces the original argument byte.

The operation proceeds until the first-operand field is exhausted. The list is not altered unless an overlap occurs.

When the operands overlap, the result is obtained as if each result byte were stored immediately after the corresponding function byte is fetched.

Access exceptions are recognized only for those bytes in the second operand which are actually required.

Condition Code: The code remains unchanged.

Program Exceptions:

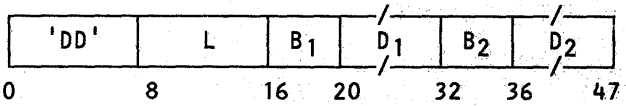
Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes

1. An example of the use of TRANSLATE is given in Appendix A.
2. The instruction TRANSLATE may be used to convert data from one code to another code.
3. The instruction may also be used to rearrange data. This may be accomplished by placing a pattern in the destination area, by designating the pattern as the first operand of TRANSLATE, and by designating the data that is to be rearranged as the second operand. Each byte of the pattern contains an eight-bit number specifying the byte destined for this position. Thus, when the instruction is executed, the pattern selects the bytes of the second operand in the desired order.
4. The fetch and subsequent store accesses to a particular byte in the first-operand field do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for the instruction OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.
5. Because each eight-bit argument byte is added to the initial second-operand address to obtain the address of a function byte, the list may contain 256 bytes. In cases where it is known that not all eight-bit argument values will occur, it is possible to reduce the size of the list.
6. Significant performance degradation is possible when the second-operand address of TRANSLATE designates a location that is less than 256 bytes to the left of a 2,048-byte boundary. This is because the machine may perform a trial execution of the instruction to determine if the second operand actually crosses the boundary.

TRANSLATE AND TEST

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



The bytes of the first operand are used as eight-bit arguments to select function bytes from a list designated by the second-operand address. The

first nonzero function byte is inserted in general register 2, and the related argument address in general register 1.

The L field designates the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding from left to right. The first operand remains unchanged in storage. Fetching of the function byte from the list is performed as in TRANSLATE. The function byte retrieved from the list is inspected for a value of zero.

When the function byte is zero, the operation proceeds with the next byte of the first operand. When the first-operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting condition code 0. The contents of general registers 1 and 2 remain unchanged.

When the function byte is nonzero, the operation is completed by inserting the function byte in general register 2 and the related argument address in general register 1. This address points to the argument byte last translated. The function byte replaces bits 24-31 of general register 2. The address replaces bits 8-31 of general register 1. Bits 0-7 of general register 1 and bits 0-23 of general register 2 remain unchanged.

When the function byte is nonzero, either condition code 1 or 2 is set, depending on whether the argument byte is the rightmost byte of the first operand. Condition code 1 is set if one or more argument bytes remain to be translated. Condition code 2 is set if no more argument bytes remain.

Access exceptions are recognized only for those bytes in the second operand which are actually required. Access exceptions are not recognized for those bytes in the first operand which are to the right of the first byte for which a nonzero function byte is obtained.

Resulting Condition Code:

0 All function bytes zero
1 Nonzero function byte; first-operand field not exhausted
2 Nonzero function byte; first-operand field exhausted
3 –

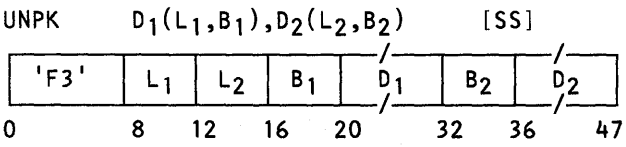
Program Exceptions:

Access (fetch, operands 1 and 2)

Programming Notes

1. An example of the use of TRANSLATE AND TEST is given in Appendix A.
2. The instruction TRANSLATE AND TEST may be used to scan the first operand for characters with special meaning. The second operand, or list, is set up with all-zero function bytes for those characters to be skipped over and with nonzero function bytes for the characters to be detected.

UNPACK



The format of the second operand is changed from packed to zoned, and the result is placed in the first-operand location. The packed and zoned formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the packed format. Its digits and sign are placed unchanged in the first-operand location, using the zoned format. Zones with coding of 1111 are supplied for all bytes except the low-order byte, which receives the sign of the second operand. The sign and digits are not checked for valid codes.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first-operand field is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched. The entire rightmost second-operand byte is used in forming the first result byte. For the remainder of the field, information for two result bytes is obtained from a single second-operand byte, and the leftmost four bits of the byte remain available and are not refetched. Thus, two result bytes are stored immediately after fetching a single operand byte.

Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2; store, operand 1)

Programming Notes
1. An example of the use of UNPACK is given in Appendix A.
2. A field that is to be unpacked can be destroyed by improper overlapping. To save storage space for unpacking by overlapping the operands, the rightmost position of the first operand must be to the right of the rightmost position of the second operand by the number of bytes in the second operand minus 2. If only one or two bytes are to be unpacked, the low-order positions of the two operands may coincide.

Chapter 8. Decimal Instructions

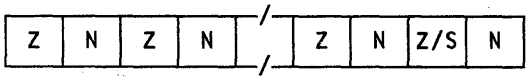
Contents

The decimal instructions of this chapter perform arithmetic and editing operations on decimal data. Additional operations on decimal data are provided by several of the instructions in Chapter 7, "General Instructions." Decimal operands always reside in storage, and all instructions operating on decimal data use the SS instruction format.

Decimal-Number Formats

Decimal numbers may be in either the zoned or packed format. Both decimal-number formats have from one to 16 bytes, each byte consisting of a pair of four-bit codes. The four-bit codes include decimal-digit codes, sign codes, and a zone code. Decimal operands occupy storage fields that start on a byte boundary.

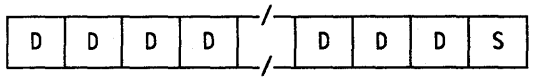
Zoned Format



In the zoned format, the rightmost four bits of a byte are called the numeric bits (N) and normally comprise a code representing a decimal digit. The leftmost four bits of a byte are called the zone bits (Z), except for the rightmost byte of a decimal operand, where these bits may be treated either as a zone or as a sign (S).

Decimal digits in the zoned format may be part of a larger character set, which includes also alphabetic and special characters. The zoned format is, therefore, suitable for input, editing, and output of numeric data in human-readable form. There are no decimal-arithmetic instructions which operate directly on decimal numbers in the zoned format; such numbers must first be converted to the packed format.

Packed Format



In the packed format, each byte contains two decimal digits (D), except for the rightmost byte, which contains a sign to the right of a decimal digit. Decimal arithmetic is performed with operands in the packed format and generates results in the packed format.

For all decimal instructions in this chapter other than EDIT and EDIT AND MARK, both operands are in the packed format.

Decimal Codes

The decimal digits 0-9 have the binary encoding 0000-1001.

The preferred sign codes are 1100 for plus and 1101 for minus. These are the sign codes

generated for the results of the decimal-arithmetic instructions and the CONVERT TO DECIMAL instruction.

Alternate sign codes are also recognized as valid when appearing in the sign position: 1010, 1110, and 1111 are alternate codes for plus, and 1011 is an alternate code for minus. Alternate sign codes are accepted for any decimal operand but are never generated or propagated in the signed result of a decimal-arithmetic instruction or CONVERT TO DECIMAL, even when an operand remains otherwise unchanged, such as when adding zero to a number. An alternate sign code is, however, left unchanged by the instructions MOVE NUMERICS, MOVE WITH OFFSET, MOVE ZONES, PACK, and UNPACK.

When an invalid code is detected, a data exception is recognized. For the decimal-arithmetic instructions, the action taken for a data exception depends on whether a sign code is invalid. When a sign code is invalid, the operation is suppressed regardless of whether any other condition causing an exception exists. When no sign code is invalid, the operation is terminated.

For the editing instructions EDIT and EDIT AND MARK, an invalid sign code is not recognized. The operation is terminated for a data exception due to an invalid digit code. No validity checking is performed by the instructions MOVE NUMERICS, MOVE WITH OFFSET, MOVE ZONES, PACK, and UNPACK.

The zone code 1111 appears in the left four bit positions of each byte representing a decimal digit in zoned-format results. Zoned-format results are produced by the instructions EDIT, EDIT AND MARK, and UNPACK, except that the left four bit positions of the rightmost byte produced by UNPACK contain whatever code exists in the sign position of the packed operand. The right four bit positions of each byte in the zoned format contain a decimal-digit code.

The meaning of the decimal codes is summarized in the figure "Summary of Digit and Sign Codes."

Programming Notes

1. Since 1111 is both the zone code and an alternate code for plus, unsigned (positive) decimal numbers may be represented in the zoned format with 1111 codes in all byte positions. The result of the PACK instruction converting such a number to the packed format may be used directly as an operand for decimal instructions.
2. The use of the alternate minus code 1011 is not recommended.

	Recognized As	
Code	Digit	Sign
0000	0	Invalid
0001	1	Invalid
0010	2	Invalid
0011	3	Invalid
0100	4	Invalid
0101	5	Invalid
0110	6	Invalid
0111	7	Invalid
1000	8	Invalid
1001	9	Invalid
1010	Invalid	Plus
1011	Invalid	Minus
1100	Invalid	Plus (preferred)
1101	Invalid	Minus (preferred)
1110	Invalid	Plus
1111	Invalid	Plus (zone)

Summary of Digit and Sign Codes

Decimal Operations

The decimal instructions in this chapter consist of two classes, the decimal-arithmetic instructions and the editing instructions.

Decimal-Arithmetic Instructions

The decimal-arithmetic instructions, which comprise all of the instructions in this chapter except the two editing instructions, perform addition, subtraction, multiplication, division, comparison, and shifting.

Operands of the decimal-arithmetic instructions are in the packed format and are treated as signed decimal integers. A decimal integer is represented in true form as an absolute value with a separate plus or minus sign. It contains an odd number of decimal digits, from one to 31, and the sign; this corresponds to an operand length of one to 16 bytes.

A decimal zero normally has a plus sign, but multiplication, division, and overflow may produce a zero value with a minus sign. Such a negative zero is a valid operand and is treated as equal to a positive zero by the COMPARE DECIMAL instruction.

The lengths of the two operands specified in the instruction need not be the same. If necessary, the shorter operand is considered to be extended with zeros to the left of the high-order digit. Results, however, cannot exceed the first-operand length as specified in the instruction.

When a carry or some high-order nonzero digits of the result are lost because the first-operand field is too short, the result is obtained by ignoring the overflow information, condition code 3 is set, and, if the decimal-overflow mask bit is one, a program interruption for decimal overflow occurs. The operand lengths alone are not an indication of overflow; significant digits must have been lost during the operation.

The operands of decimal-arithmetic instructions should not overlap at all or should have coincident rightmost bytes. In ZERO AND ADD, the operands may also overlap in such a manner that the rightmost byte of the first operand (which becomes the result) is to the right of the rightmost byte of the second operand. For these cases of proper overlap, the result is obtained as if operands were processed right to left. Because the codes for digits and signs are verified during the performance of the arithmetic, improperly overlapping operands are recognized as data exceptions.

Programming Note
The same decimal field in storage may be specified for both operands of the instructions ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, and SUBTRACT DECIMAL. Thus, a decimal number may be added to itself, compared to itself, etc. SUBTRACT DECIMAL may be used to set a decimal field in storage to zero.

Editing Instructions

The editing instructions are EDIT and EDIT AND MARK. For these instructions, only one operand (the pattern) has an explicitly specified length. The other operand (the source) is considered to have as many digits as necessary for the completion of the operation.

Overlapping operands for the editing instructions yield unpredictable results.

Execution of Decimal Instructions

During the execution of a decimal instruction, all bytes of the operands are not necessarily accessed concurrently, and the fetch and store accesses to a single location do not necessarily occur one immediately after the other. Furthermore, for decimal instructions, intermediate values may be placed in the result field that may differ from the original operand and final result values. Thus, in a multiprocessing system, an instruction such as ADD DECIMAL cannot be safely used to update a shared storage location when the possibility exists that another CPU may also be updating that location.

Other Instructions for Decimal Operands

In addition to the decimal instructions in this chapter, the instructions MOVE NUMERICS and MOVE ZONES are provided for operating on data in the zoned format. Two instructions are provided for converting data between the zoned and packed formats: the PACK instruction transforms zoned data into packed data, and UNPACK performs the reverse transformation. The MOVE WITH OFFSET instruction operates on packed data. Two instructions are provided for conversion between the packed-decimal and binary formats. The CONVERT TO BINARY instruction converts packed decimal to binary, and CONVERT TO DECIMAL converts binary to packed decimal. These seven instructions are not considered to be decimal instructions and are described in Chapter 7, "General Instructions." The editing instructions in this chapter may also be used to change data from the packed to the zoned format.

Instructions

The decimal instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of Decimal Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

Note: *In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For ADD DECIMAL, for example, AP is the mnemonic and* $D_1(L_1,B_1)$, $D_2(L_2,B_2)$ *the operand designation.*

Name	Mnemonic	Characteristics								Op Code
ADD DECIMAL	AP	SS	C		A		D	DF	ST	FA
COMPARE DECIMAL	CP	SS	C		A		D			F9
DIVIDE DECIMAL	DP	SS			A	SP	D	DK	ST	FD
EDIT	ED	SS	C		A		D		ST	DE
EDIT AND MARK	EDMK	SS	C		A		D		R ST	DF
MULTIPLY DECIMAL	MP	SS			A	SP	D		ST	FC
SHIFT AND ROUND DECIMAL	SRP	SS	C		A		D	DF	ST	F0
SUBTRACT DECIMAL	SP	SS	C		A		D	DF	ST	FB
ZERO AND ADD	ZAP	SS	C		A		D	DF	ST	F8

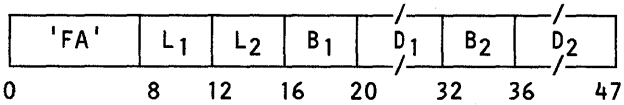
Explanation:

A Access exceptions
C Condition code is set
D Data exception
DF Decimal-overflow exception
DK Decimal-divide exception
R PER general-register-alteration event
SP Specification exception
SS SS instruction format
ST PER storage-alteration event

Summary of Decimal Instructions

ADD DECIMAL

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



The second operand is added to the first operand, and the resulting sum is placed in the first-operand location. The operands and result are in the packed format.

Addition is algebraic, taking into account the signs and all digits of both operands. All sign and digit codes are checked for validity.

If the first operand is too short to contain all significant digits of the sum, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow information, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow takes place.

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. When high-order digits are lost because of an overflow, a zero result 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 Exceptions:

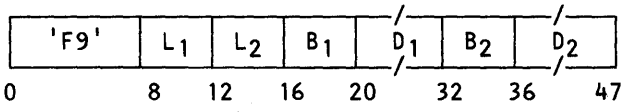
Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal Overflow

Programming Note

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

COMPARE DECIMAL

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



The first operand is compared with the second operand, and the result is indicated in the condition code. The operands are in the packed format.

Comparison is algebraic and follows the procedure for decimal subtraction, except that both operands remain unchanged. When the difference is zero, the operands are equal. When a nonzero difference is positive or negative, the first operand is high or low, respectively.

Overflow cannot occur because the difference is discarded.

All sign and digit codes are checked for validity.

Resulting Condition Code:

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

Program Exceptions:

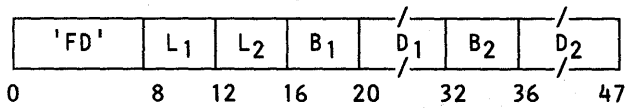
Access (fetch, operands 1 and 2)
Data

Programming Notes

1. An example of the use of COMPARE DECIMAL is given in Appendix A.
2. The comparison operation does not distinguish between valid sign codes. A valid plus or minus sign is equivalent to any other valid plus or minus sign, respectively.

DIVIDE DECIMAL

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



The first operand (the dividend) is divided by the second operand (the divisor). The resulting quotient and remainder are placed in the first-operand location. The operands and result are in the packed format.

The quotient is placed leftmost in the first-operand location. The number of bytes in the quotient is equal to the difference between the dividend and divisor lengths (L_1 - L_2). The remainder is placed rightmost in the first-operand location and has a length equal to the divisor length. Together, the quotient and remainder occupy the entire first operand; therefore, the address of the quotient is the address of the first operand.

The divisor length cannot exceed 15 digits and sign (L_2 not greater than seven) and must be less than the dividend length (L_2 less than L_1); otherwise, a specification exception is recognized. The operation is suppressed, and a program interruption occurs.

The dividend, divisor, quotient, and remainder are all signed decimal integers, right-aligned in their fields. All sign and digit codes of the dividend and divisor are checked for validity.

The sign of the quotient is determined by the rules of algebra from the dividend and divisor signs. The sign of the remainder has the same value as the dividend sign. These rules hold even when the quotient or remainder is zero.

Overflow cannot occur. If the divisor is zero or the quotient is too large to be represented by the number of digits allowed, a decimal-divide exception is recognized. The operation is suppressed, and a program interruption occurs. The operands remain unchanged in storage. The decimal-divide exception is indicated only if the sign codes of both the dividend and divisor are valid, and only if the digit or digits used in establishing the exception are valid.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal Divide
Specification

Programming Notes

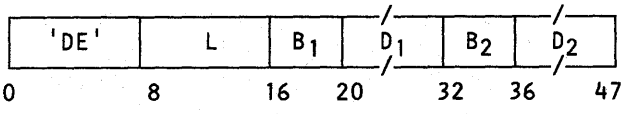
1. An example of the use of DIVIDE DECIMAL is given in Appendix A.
2. The dividend cannot exceed 31 digits and sign. Since the remainder cannot be shorter than one digit and sign, the quotient cannot exceed 29 digits and sign.
3. The condition for a decimal-divide exception can be determined by a trial subtraction. The leftmost digit of the divisor is aligned one digit to the right of the leftmost dividend digit.

When the divisor, so aligned, is less than or equal to the dividend, a divide exception is indicated.

4. A decimal-divide exception always occurs when the leftmost dividend digit is not zero.

EDIT

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



The second operand (the source), which normally contains one or more decimal numbers in the packed format, is changed to the zoned format and modified under the control of the first operand (the pattern). The edited result replaces the first operand.

The length field specifies the length of the first operand, which may contain bytes of any value.

The length of the source is determined by the operation according to the contents of the pattern. The source has the packed format. The leftmost four bits of each source byte must specify a decimal digit code (0000-1001); a sign code (1010-1111) is recognized as a data exception. The rightmost four bits may specify either a sign or a decimal digit. Access and data exceptions are recognized only for those bytes in the second operand which are actually required.

The result is obtained as if both operands were processed left to right one byte at a time. Overlapping pattern and source fields give unpredictable results.

During the editing process, each byte of the pattern is affected in one of three ways:

1. It is left unchanged.
2. It is replaced by a source digit expanded to the zoned format.
3. It is replaced by the first byte in the pattern, called the fill byte.

Which of the three actions takes place is determined by one or more of the following: the type of the pattern byte, the state of the significance indicator, and whether the source digit examined is zero.

Pattern Bytes: There are four types of pattern bytes: digit selector, significance starter, field separator, and message byte. Their coding is as follows:

Name	Code
Digit selector	0010 0000
Significance starter	0010 0001
Field separator	0010 0010
Message byte	Any other

The detection of either a digit selector or a significance starter in the pattern causes an examination to be made of the significance indicator and of a source digit. As a result, either the expanded source digit or the fill byte, as appropriate, is selected to replace the pattern byte. Additionally, encountering a digit selector or a significance starter may cause the significance indicator to be changed.

The field separator identifies individual fields in a multiple-field editing operation. It is always replaced in the result by the fill byte, and the significance indicator is always off after the field separator is encountered.

Message bytes in the pattern are either replaced by the fill byte or remain unchanged in the result, depending on the state of the significance indicator. They may thus be used for padding, punctuation, or text in the significant portion of a field or for the insertion of sign-dependent symbols.

Fill Byte: The first byte of the pattern is used as the fill byte. The fill byte can have any code and may concurrently specify a control function. If this byte is a digit selector or significance starter, the indicated editing action is taken after the code has been assigned to the fill byte.

Source Digits: Each time a digit selector or significance starter is encountered in the pattern, a new source digit is examined for placement in the pattern field. Either the source digit is disregarded, or it is expanded to the zoned format, by appending the zone code 1111 on the left, and stored in place of the pattern byte.

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 byte that calls for a digit examination. When the leftmost four bits contain an invalid digit code, the operation is terminated.

At the time the left digit of a source byte is examined, the rightmost four bits are checked for the existence of a sign code. When a sign code is encountered in the rightmost four bit positions, these bits are not treated as a decimal-digit code, and a new source byte is fetched from storage when the next pattern byte calls for a source-digit examination.

When the pattern contains no digit selector or significance starter, no source bytes are fetched and examined.

Significance Indicator: The significance indicator is turned on or off to indicate the significance or nonsignificance, respectively, of subsequent source digits or message bytes. Significant source digits replace their corresponding digit selectors or significance starters in the result. Significant message bytes remain unchanged in the result.

The significance indicator, by its on or off state, indicates also the negative or positive value, respectively, of a completed source field and is used as one factor in the setting of the condition code.

The indicator is set to off at the start of the editing operation, after a field separator is encountered, or after a source byte is examined that has a plus code in the rightmost four bit positions.

The indicator is set to on when a significance starter is encountered whose source digit is a valid decimal digit, or when a digit selector is encountered whose source digit is a nonzero decimal digit, provided that in both instances the source byte does not have a plus code in the rightmost four bit positions.

In all other situations, the indicator is not changed. A minus sign code has no effect on the significance indicator.

Result Bytes: The result of an editing operation replaces and is equal in length to the pattern. It is composed of pattern bytes, fill bytes, and zoned source digits.

If the pattern byte is a message byte and the significance indicator is on, the message byte remains unchanged in the result. If the pattern byte is a field separator or if the significance indicator is off when a message byte is encountered in the pattern, the fill byte replaces the pattern byte in the result.

If the digit selector or significance starter is encountered in the pattern with the significance indicator off and the source digit zero, the source digit is considered nonsignificant, and the fill byte replaces the pattern byte. If the digit selector or significance starter is encountered with either the significance indicator on or with a nonzero decimal source digit, the source digit is considered significant, is changed to the zoned format, and replaces the pattern byte in the result.

Condition Code: The sign and magnitude of the last field edited are used to set the condition code. The term "last field" refers to those source bytes in the second operand selected by digit selectors or significance starters after the last field separator. When the pattern contains no field separator, there is only one field, which is considered to be the last field. The last field is considered to be of zero length if no digit selectors or significance starters appear in the pattern, if none appear after the last field separator, or if the last byte in the pattern is a field separator.

Condition code 0 is set when the last field is zero or of zero length.

Condition code 1 is set when the last field edited is nonzero and the significance indicator is on, indicating a result less than zero.

Condition code 2 is set when the last field edited is nonzero and the significance indicator is off, indicating a result greater than zero.

The figure "Summary of EDIT Functions" 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 byte placed in the result field and the new setting of the significance indicator.

Resulting Condition Code:

0 Last field is zero or of zero length
1 Last field is less than zero
2 Last field is greater than zero
3 –

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Notes

1. Examples of the use of EDIT are given in Appendix A.
2. Editing includes sign and punctuation control, and the suppression and protection of leading zeros by replacing them with blanks or asterisks. It also facilitates programmed blanking of all-zero fields. Several fields may be edited in one operation, and numeric information may be combined with text.
3. As a rule, the source is shorter than the pattern, because each 4-bit source digit is generally replaced by an 8-bit byte in the result.
4. The total number of digit selectors and significance starters in the pattern must equal the number of source digits to be edited.
5. If the fill byte 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.
6. The resulting condition code indicates whether or not the last field is all zeros and, if nonzero, reflects the state of the significance indicator. The significance indicator reflects the sign of the source field only if the last source byte examined contains a sign code in the low-order digit position. For multiple-field editing operations, the condition code reflects the sign and value only of the field following the last field separator.
7. Significant performance degradation is possible when the second-operand address of EDIT designates a location that is less than the length of the first operand to the left of a 2,048-byte boundary. This is because the machine may perform a trial execution of the instruction to determine if the second operand actually crosses the boundary. It should be noted that the second operand of EDIT, while normally shorter than the first operand, can in the extreme case have the same length as the first.

Conditions				Results	
Pattern Byte	Previous State of Significance Indicator	Source Digit	Right Four Source Bits Are Plus Code	Result Byte	State of Significance Indicator at End of Digit Examination
Digit selector	Off	0	*	Fill byte	Off
		1-9	No	Source digit	On
		1-9	Yes	Source digit	Off
	On	0-9	No	Source digit	On
		0-9	Yes	Source digit	Off
Significance starter	Off	0	No	Fill byte	On
		0	Yes	Fill byte	Off
		1-9	No	Source digit	On
		1-9	Yes	Source digit	Off
	On	0-9	No	Source digit	On
		0-9	Yes	Source digit	Off
Field separator	*	**	**	Fill byte	Off
Message byte	Off	**	**	Fill byte	Off
	On	**	**	Message byte	On

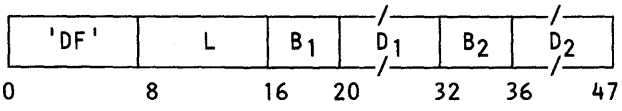
Explanation:

* No effect on result byte or on new state of significance indicator
** Not applicable because source is not examined

Summary of EDIT Functions

EDIT AND MARK

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



The second operand (the source), which normally contains one or more decimal numbers in the packed format, is changed to the zoned format and modified under the control of the first operand (the pattern). The address of each first significant result byte is inserted in general register 1. The edited result replaces the pattern.

The instruction EDIT AND MARK is identical to EDIT, except for the additional function of inserting the address of the result byte in bit positions 8-31 of general register 1 whenever the result byte is a zoned source digit and the significance indicator was off before the examination. Bits 0-7 of the register are not changed.

Resulting Condition Code:

0 Last field is zero or of zero length
1 Last field is less than zero
2 Last field is greater than zero
3 –

Program Exceptions:

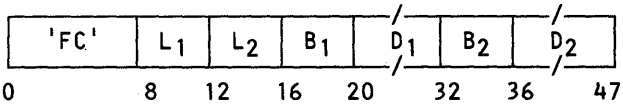
Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Notes

1. Examples of the use of EDIT AND MARK are given in Appendix A.
2. The instruction EDIT AND MARK facilitates the programming of floating currency-symbol insertion. The address inserted in general register 1 is one greater than the address where a floating currency-sign would be inserted. The instruction BRANCH ON COUNT (BCTR), with zero in the R_2 field, may be used to reduce the inserted address by one.
3. No address is inserted in general register 1 when the significance indicator is turned on as a result of encountering a significance starter with the corresponding source digit zero. To ensure that general register 1 contains a valid address when this occurs, the address of the pattern byte that immediately follows the significance starter should be placed in the register beforehand.
4. When multiple fields are edited with one EDIT AND MARK instruction, the address inserted in general register 1 applies only to the last field edited.
5. See also the programming note under EDIT regarding performance degradation due to a possible trial execution.

MULTIPLY DECIMAL

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



The product of the first operand (the multiplicand) and the second operand (the multiplier) is placed in the first-operand location. The operands and result are in the packed format.

The multiplier length cannot exceed 15 digits and sign (L_2 not greater than seven) and must be less than the multiplicand length (L_2 less than L_1); otherwise a specification exception is recognized. The operation is suppressed, and a program interruption occurs.

The multiplicand must have at least as many bytes of high-order zeros as the number of bytes in the multiplier; otherwise, a data exception is recognized, the operation is terminated, and a program interruption occurs. This restriction ensures that no product overflow occurs.

The multiplicand, multiplier, and product are all signed decimal integers, right-aligned in their fields. All sign and digit codes of the multiplicand and multiplier are checked for validity.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs, even if one or both operands are zeros.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)

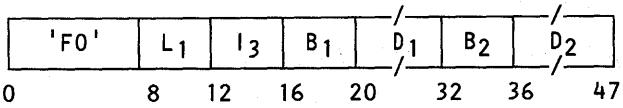
Data
Specification

Programming Notes

1. An example of the use of MULTIPLY DECIMAL is given in Appendix A.
2. The product cannot exceed 31 digits and sign. The leftmost digit of the product is always zero.

SHIFT AND ROUND DECIMAL

SRP $D_1(L_1,B_1),D_2(B_2),I_3$ [SS]



The first operand is shifted in the direction and for the number of decimal-digit positions specified by the second-operand address, and, when shifting to the right is specified, the absolute value of the first operand is rounded by the rounding digit, I_3. The first operand and the result are in the packed format.

The first operand is considered to be in the packed-decimal format. Only its digit portion is shifted; the sign position does not participate in the shifting. Zeros are supplied for the vacated digit positions. The result replaces the first operand. Nothing is stored outside of the specified first-operand location.

The second-operand address, specified by the B_2 and D_2 fields, is not used to address data; bits 26-31 are the shift value, and the high-order bits of the address are ignored.

The shift value is a six-bit signed binary integer, indicating the direction and the number of decimal-digit positions to be shifted. Positive shift values specify shifting to the left. Negative shift values, which are represented in two's complement notation, specify shifting to the right. The following are examples of the interpretation of shift values.

Shift Value	Amount and Direction
011111	31 digits to the left
000001	One digit to the left
000000	No shift
111111	One digit to the right
100000	32 digits to the right

For a right shift, the I_3 field, bits 12-15 of the instruction, are used as a decimal rounding digit. The first operand, which is treated as positive by ignoring the sign, is rounded by decimally adding the rounding digit to the leftmost of the digits to be shifted out and by propagating the carry, if any, to the left. The result of this addition is then shifted right. Except for validity checking and the participation in rounding, the digits shifted out of the low-order decimal-digit position are ignored and are lost.

If one or more significant digits are shifted out of the high-order digit positions during a left shift, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow information, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow takes place. Overflow cannot occur for a right shift, with or without rounding, or when no shifting is specified.

In the absence of overflow, the sign of a zero result is made positive. Otherwise, the sign of the result is the same as the original sign, but the code is the preferred sign code.

A data exception is recognized when the first operand does not have valid sign and digit codes or when the rounding digit is not a valid digit code. The validity of the first-operand codes is checked even when no shift is specified, and the validity of the rounding digit is checked even when no addition for rounding takes place.

Resulting Condition Code:

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

Program Exceptions:

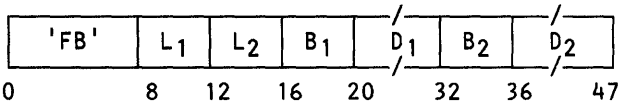
Access (fetch and store, operand 1)
Data
Decimal Overflow

Programming Notes

1. Examples of the use of SHIFT AND ROUND are given in Appendix A.
2. SHIFT AND ROUND can be used for shifting up to 31 digit positions left and up to 32 digit positions right. This is sufficient to clear all digits of any decimal number even with rounding.
3. For right shifts, the rounding digit 5 provides conventional rounding of the result. The rounding digit 0 specifies truncation without rounding.
4. When the B_2 field is zero, the six-bit shift value is obtained directly from bits 42-47 of the instruction.

SUBTRACT DECIMAL

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



The second operand is subtracted from the first operand, and the resulting difference is placed in the first-operand location. The operands and result are in the packed format.

SUBTRACT DECIMAL is executed the same as ADD DECIMAL, except that the second operand is considered to have a sign opposite to the sign in storage. The second operand in storage remains unchanged.

Resulting Condition Code:

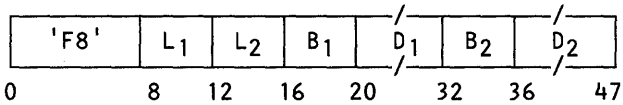
0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal Overflow

ZERO AND ADD

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



The second operand is placed in the first-operand location. The operation is equivalent to an addition to zero. The operand and result are in the packed format.

Only the second operand is checked for valid sign and digit codes. Extra high-order zeros are supplied for the shorter operand if needed.

If the first operand is too short to contain all significant digits of the second operand, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow information, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow takes place.

A zero result is positive. However, when significant high-order digits are lost because of overflow, a zero result has the sign of the second operand.

The two operands may overlap, provided the rightmost byte of the first operand is coincident with or to the right of the rightmost byte of the second operand. In this case the result is obtained as if the operands were processed right to left.

Resulting Condition Code:

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

Program Exceptions:

Access (fetch, operand 2; store, operand 1)
Data
Decimal Overflow

Programming Note

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

Chapter 9. Floating-Point Instructions

Contents

Floating-point instructions are used to perform calculations on operands with a wide range of magnitude and to yield results scaled to preserve precision.

The floating-point instructions provide for loading, rounding, adding, subtracting, comparing, multiplying, dividing, and storing, as well as controlling the sign of short, long, and extended operands. Short operands generally permit faster processing and require less storage than long or extended operands. On the other hand, long and extended operands permit greater precision in computation. Four floating-point registers are provided. Instructions may perform either register-to-register or storage-and-register operations.

Most of the instructions generate normalized results, which preserve the highest precision in the operation. For addition and subtraction, instructions are also provided that generate unnormalized results. Either normalized or unnormalized numbers may be used as operands for any floating-point operation.

Floating-Point Number Representation

A floating-point number consists of a signed hexadecimal fraction and an unsigned seven-bit binary integer called the characteristic. The characteristic represents a signed exponent and is obtained by adding 64 to the exponent value (excess-64 notation). The range of the characteristic is 0 to 127, which corresponds to an exponent range of -64 to +63. The value of a floating-point number is the product of its fraction and the number 16 raised to the power of the exponent which is represented by its characteristic.

The fraction of a floating-point number is treated as a hexadecimal number because it is considered to be multiplied by a number which is a power of 16. The name, fraction, indicates that the radix point is assumed to be immediately to the left of the leftmost fraction digit. The fraction is represented by its absolute value and a separate sign bit. The entire number is positive or negative, depending on whether the sign bit of the fraction is zero or one, respectively.

When a floating-point operation would cause the result exponent to exceed 63, the characteristic wraps around from 127 to 0, and an exponent-overflow condition exists. The result characteristic is then too small by 128. When an operation would cause the exponent to be less than -64, the characteristic wraps around from 0 to 127, and an exponent-underflow condition exists. The result characteristic is then too large by 128, except that a zero characteristic is produced when a true zero is forced.

A true zero is a floating-point number with a zero characteristic, zero fraction, and plus sign. A true zero may arise as the normal result of an arithmetic operation because of the particular

magnitude of the operands. The result is forced to be a true zero when:

1. An exponent underflow occurs and the exponent-underflow mask bit in the PSW is zero,
2. The result fraction of an addition or subtraction operation is zero and the significance mask bit in the PSW is zero, or
3. The operand of HALVE, one or both operands of MULTIPLY, or the dividend in DIVIDE has a zero fraction.

When a program interruption for exponent underflow occurs, a true zero is not forced; instead, the fraction and sign remain correct, and the characteristic is too large by 128. When a program interruption for significance occurs, the fraction remains zero, the sign is positive, and the characteristic remains correct.

The sign of a sum, difference, product, or quotient with a zero fraction is positive. The sign of a zero fraction resulting from other operations is established from the operand sign, the same as for nonzero fractions.

Normalization

A quantity can be represented with the greatest precision by a floating-point number of a given fraction length when that number is normalized. A normalized floating-point number has a nonzero leftmost hexadecimal fraction digit. If one or more leftmost fraction digits are zeros, the number is said to be unnormalized.

Unnormalized numbers are normalized by shifting the fraction left, one digit at a time, until the leftmost hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A number with a zero fraction cannot be normalized; its characteristic either remains unchanged, or it is made zero when the result is forced to be a true zero.

Floating-point operations may be performed with or without normalization. Most operations are performed only with normalization. Addition and subtraction with short or long operands may be specified as either normalized or unnormalized.

With unnormalized operations, leftmost 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. The operands for multiplication and division are normalized before the arithmetic process. For other normalized operations, normalization takes place when the intermediate arithmetic result is changed to the final result.

When the intermediate result of addition, subtraction, or rounding causes the fraction to overflow, the fraction is shifted right by one hexadecimal-digit position and the value one is placed in the vacated leftmost digit position. The fraction is then truncated to the final result length, while the characteristic is increased by one. This adjustment is made for both normalized and unnormalized operations.

Programming Note

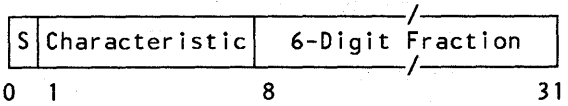
Up to three leftmost bits of the fraction of a normalized number may be zeros, since the nonzero test applies to the entire leftmost hexadecimal digit.

Floating-Point-Data Format

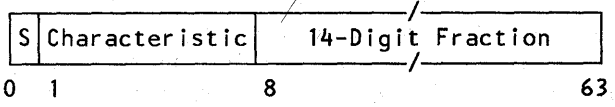
Floating-point numbers have a 32-bit (short) format, a 64-bit (long) format, or a 128-bit (extended) format. Numbers in the short and long formats may be designated as operands both in storage and in the floating-point registers, whereas operands having the extended format can be designated only in the floating-point registers.

The floating-point registers contain 64 bits each and are numbered 0, 2, 4, and 6. A short or long floating-point number requires a single floating-point register. An extended floating-point number requires a pair of these registers: either registers 0 and 2 or register 4 and 6; the two register pairs are designated as 0 or 4, respectively. When the R_1 or R_2 field of a floating-point instruction designates any register number other than 0, 2, 4, or 6 for the short or long format, or any register number other than 0 or 4 for the extended format, the operation is suppressed, and a program interruption for specification exception occurs.

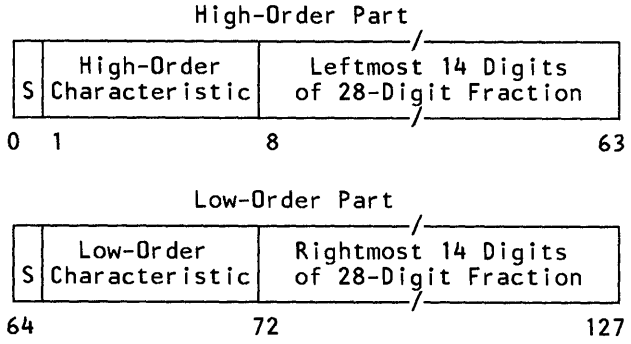
Short Floating-Point Number



Long Floating-Point Number



Extended Floating-Point Number



In all formats, the first bit (bit 0) is the sign bit (S). The next seven bits are the characteristic. In the short and long formats, the remaining bits constitute the fraction, which consists of six or 14 hexadecimal digits, respectively.

A short floating-point number occupies only the leftmost 32 bit positions of a floating-point register. The rightmost 32 bit positions of the register are ignored when used as an operand in the short format and remain unchanged when a short result is placed in the register.

An extended floating-point number has a 28-digit fraction and consists of two long floating-point numbers which are called the high-order and low-order parts. The high-order part may be any long floating-point number. The fraction of the high-order part contains the leftmost 14 hexadecimal digits of the 28-digit fraction. The characteristic and sign of the high-order part are the characteristic and sign of the extended floating-point number. If the high-order part is normalized, the extended number is considered normalized. The fraction of the low-order part contains the rightmost 14 digits of the 28-digit fraction. The sign and characteristic of the low-order part of an extended operand are ignored.

When a result in the extended format is placed in a register pair, the sign of the low-order part is made the same as that of the high-order part, and, unless the result is a true zero, the low-order characteristic is made 14 less than the high-order characteristic. When the subtraction of 14 would cause the low-order characteristic to become less than zero, the characteristic is made 128 greater than its correct value. Exponent underflow is indicated only when the high-order characteristic underflows.

When an extended result is made a true zero, both the high-order and low-order parts are made a true zero.

The range covered by the magnitude (M) of a normalized floating-point number depends on the format.

In the short format:

$$16^{65} \leq M \leq (1 - 16^{6}) \times 16^{63}$$

In the long format:

$$16^{65} \leq M \leq (1 - 16^{14}) \times 16^{63}$$

In the extended format:

$$16^{65} \leq M \leq (1 - 16^{28}) \times 16^{63}$$

In all formats, approximately:

$$5.4 \times 10^{79} \leq M \leq 7.2 \times 10^{75}$$

Although the final result of a floating-point operation has six hexadecimal fraction digits in the short format, 14 fraction digits in the long format, and 28 fraction digits in the extended format, intermediate results have one additional hexadecimal digit on the right. This digit is called the guard digit. The guard digit may increase the precision of the final result because it participates in addition, subtraction, and comparison operations and in the left shift that occurs during normalization.

The entire set of floating-point operations is available for both short and long operands. These instructions generate a result that has the same format as the operands, except that for MULTIPLY, a long product is produced from a short multiplier and multiplicand. Extended floating-point instructions are provided only for normalized addition, subtraction, and multiplication. Two additional multiplication instructions generate an extended product from a long multiplier and multiplicand. The rounding instructions provide for rounding from extended to long format and from long to short format.

Programming Notes

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

2. In the absence of an exponent overflow or exponent underflow, the long floating-point number constituting the low-order part of an extended result correctly expresses the value of the low-order part of the extended result when the characteristic of the high-order part is 14 or higher. This applies also when the result is a true zero. When the high-order characteristic is less than 14 but the number is not a true zero, the low-order part, when addressed as a long floating-point number, does not have the correct characteristic value.
3. The entire fraction of an extended result participates in normalization. The low-order part alone may or may not appear to be a normalized long floating-point number, depending on whether the 15th digit of the normalized 28-digit fraction is nonzero or zero.

Instructions

The floating-point instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of Floating-Point Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

Mnemonics for the floating-point instructions have an R as the last letter when the instruction is in the RR format. For instructions where all operands are the same length, certain letters are used to represent operand-format length and normalization, as follows:

E short normalized
U short unnormalized
D long normalized
W long unnormalized
X extended normalized

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

Name	Mne-monic	Characteristics						Op Code
ADD NORMALIZED (extended)	AXR	RR	C		SP	EU	EO LS		36
ADD NORMALIZED (long)	ADR	RR	C		SP	EU	EO LS		2A
ADD NORMALIZED (long)	AD	RX	C	A	SP	EU	EO LS		6A
ADD NORMALIZED (short)	AER	RR	C		SP	EU	EO LS		3A
ADD NORMALIZED (short)	AE	RX	C	A	SP	EU	EO LS		7A
ADD UNNORMALIZED (long)	AWR	RR	C		SP		EO LS		2E
ADD UNNORMALIZED (long)	AW	RX	C	A	SP		EO LS		6E
ADD UNNORMALIZED (short)	AUR	RR	C		SP		EO LS		3E
ADD UNNORMALIZED (short)	AU	RX	C	A	SP		EO LS		7E
COMPARE (long)	CDR	RR	C		SP				29
COMPARE (long)	CD	RX	C	A	SP				69
COMPARE (short)	CER	RR	C		SP				39
COMPARE (short)	CE	RX	C	A	SP				79
DIVIDE (long)	DDR	RR			SP	EU	EO FK		2D
DIVIDE (long)	DD	RX		A	SP	EU	EO FK		6D
DIVIDE (short)	DER	RR			SP	EU	EO FK		3D
DIVIDE (short)	DE	RX		A	SP	EU	EO FK		7D
HALVE (long)	HDR	RR			SP	EU			24
HALVE (short)	HER	RR			SP	EU			34
LOAD (long)	LDR	RR			SP				28
LOAD (long)	LD	RX		A	SP				68
LOAD (short)	LER	RR			SP				38
LOAD (short)	LE	RX		A	SP				78
LOAD AND TEST (long)	LTDR	RR	C		SP				22
LOAD AND TEST (short)	LTER	RR	C		SP				32
LOAD COMPLEMENT (long)	LCDR	RR	C		SP				23
LOAD COMPLEMENT (short)	LCER	RR	C		SP				33
LOAD NEGATIVE (long)	LNDR	RR	C		SP				21
LOAD NEGATIVE (short)	LNER	RR	C		SP				31
LOAD POSITIVE (long)	LPDR	RR	C		SP				20
LOAD POSITIVE (short)	LPER	RR	C		SP				30
LOAD ROUNDED (extended to long)	LRDR	RR			SP		EO		25
LOAD ROUNDED (long to short)	LRER	RR			SP		EO		35
MULTIPLY (extended)	MXR	RR			SP	EU	EO		26
MULTIPLY (long)	MDR	RR			SP	EU	EO		2C
MULTIPLY (long)	MD	RX		A	SP	EU	EO		6C
MULTIPLY (long to extended)	MXDR	RR			SP	EU	EO		27
MULTIPLY (long to extended)	MXD	RX		A	SP	EU	EO		67
MULTIPLY (short to long)	MER	RR			SP	EU	EO		3C
MULTIPLY (short to long)	ME	RX		A	SP	EU	EO		7C
STORE (long)	STD	RX		A	SP			ST	60
STORE (short)	STE	RX		A	SP			ST	70
SUBTRACT NORMALIZED (extended)	SXR	RR	C		SP	EU	EO LS		37
SUBTRACT NORMALIZED (long)	SDR	RR	C		SP	EU	EO LS		2B
SUBTRACT NORMALIZED (long)	SD	RX	C	A	SP	EU	EO LS		6B
SUBTRACT NORMALIZED (short)	SER	RR	C		SP	EU	EO LS		3B
SUBTRACT NORMALIZED (short)	SE	RX	C	A	SP	EU	EO LS		7B
SUBTRACT UNNORMALIZED (long)	SWR	RR	C		SP		EO LS		2F
SUBTRACT UNNORMALIZED (long)	SW	RX	C	A	SP		EO LS		6F
SUBTRACT UNNORMALIZED (short)	SUR	RR	C		SP		EO LS		3F
SUBTRACT UNNORMALIZED (short)	SU	RX	C	A	SP		EO LS		7F

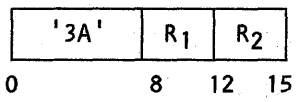
Explanation:

A Access exceptions
C Condition code is set
EO Exponent-overflow exception
EU Exponent-underflow exception
FK Floating-point-divide exception
LS Significance exception
RR RR instruction format
RX RX instruction format
SP Specification exception
ST PER storage-alteration event

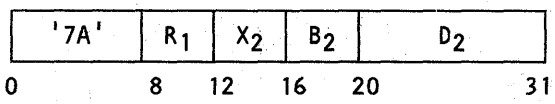
Summary of Floating-Point Instructions

ADD NORMALIZED

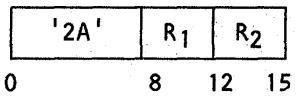
AER R_1,R_2 [RR, Short Operands]



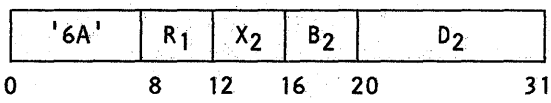
AE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



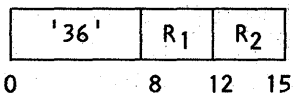
ADR R_1,R_2 [RR, Long Operands]



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



AXR R_1,R_2 [RR, Extended Operands]



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

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

When a fraction is shifted right during alignment, the leftmost hexadecimal digit shifted out is retained as a guard digit. The fraction that is not shifted is considered to be extended with a zero in the guard-digit position. When no alignment shift occurs, both operands are considered to be extended with zeros in the guard-digit position. The fractions are then added algebraically to form an intermediate sum.

The intermediate-sum fraction consists of seven (short format), 15 (long format), or 29 (extended format) hexadecimal digits, including the guard digit, and a possible carry. If a carry is present, the sum is shifted right one digit position so that the carry becomes the leftmost digit of the fraction, and the characteristic is increased by one.

If the addition produces no carry, the intermediate-sum fraction is shifted left as necessary to eliminate any leading hexadecimal zero digits resulting from the addition, provided the fraction is not zero. Vacated rightmost digit positions are filled with zeros, and the characteristic is reduced by the number of hexadecimal digits of shift. The fraction thus normalized is then truncated on the right to six (short format), 14 (long format), or 28 (extended format) hexadecimal digits. In the extended format, a characteristic is generated for the low-order part, which is 14 less than the high-order characteristic.

The sign of the sum is determined by the rules of algebra, unless all digits of the intermediate-sum fraction are zero, in which case the sign is made plus.

An exponent-overflow exception is recognized when a carry from the leftmost position of the intermediate-sum fraction would cause the characteristic of the normalized sum to exceed 127. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for exponent overflow takes place. The result sign and fraction remain correct, and, for AXR, the characteristic of the low-order part remains correct.

An exponent-underflow exception is recognized when the characteristic of the normalized sum would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the result characteristic 128 greater than the correct value. The result sign and fraction remain correct, and a program interruption for exponent underflow takes place. When exponent underflow occurs and the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the result a true zero. For AXR, no exponent underflow is recognized when the characteristic of the low-order

part would be less than zero but the characteristic of the high-order part is zero or greater.

The result fraction is zero when the intermediate-sum fraction, including the guard digit, is zero. With a zero result fraction, the action depends on the setting of the significance mask bit. If the significance mask bit is one, no normalization occurs, the intermediate and final result characteristics are the same, and a program interruption for significance takes place. If the significance mask bit is zero, the program interruption does not occur; instead, the result is made a true zero.

The R_1 field for AER, AE, ADR, and AD, and the R_2 field for AER and ADR must designate register 0, 2, 4, or 6. The R_1 and R_2 fields for AXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Resulting Condition Code:

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

Program Exceptions:

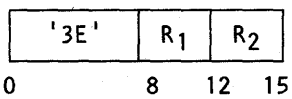
Access (fetch, operand 2 of AE and AD only)
Exponent Overflow
Exponent Underflow
Significance
Specification

Programming Notes

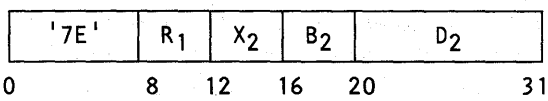
1. Interchanging the two operands in a floating-point addition does not affect the value of the sum.
2. The ADD NORMALIZED instructions normalize the sum but not the operands. Thus, if one or both operands are unnormalized, precision may be lost during fraction alignment.

ADD UNNORMALIZED

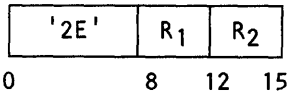
AUR R_1,R_2 [RR, Short Operands]



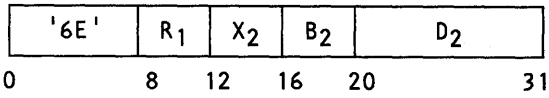
AU $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



AWR R_1,R_2 [RR, Long Operands]



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



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

The execution of ADD UNNORMALIZED is identical to that of ADD NORMALIZED, except that:

1. When no carry is present after the addition, the intermediate-sum fraction is truncated to the proper result-fraction length without a left shift to eliminate leading hexadecimal zeros and without the corresponding reduction of the characteristic.
2. Exponent underflow cannot occur.
3. The guard digit does not participate in the recognition of a zero result fraction. A zero result fraction is recognized when the fraction, that is, the intermediate-sum fraction, excluding the guard digit, is zero.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

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

Program Exceptions:

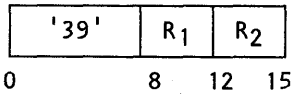
Access (fetch, operand 2 of AU and AW only)
Exponent Overflow
Significance
Specification

Programming Note

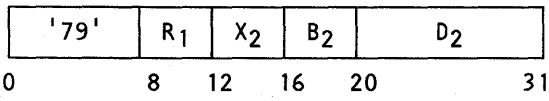
Except when the result is made a true zero, the characteristic of the result of ADD UNNORMALIZED is equal to the greater of the two operand characteristics, increased by one if the fraction addition produced a carry.

COMPARE

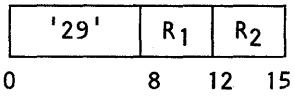
CER R_1,R_2 [RR, Short Operands]



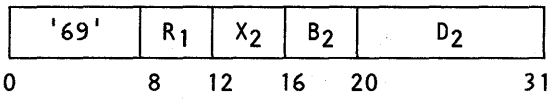
CE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



CDR R_1,R_2 [RR, Long Operands]



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



The first operand is compared with the second operand, and the condition code is set to indicate the result.

The comparison is algebraic and follows the procedure for normalized floating-point subtraction, except that the difference is discarded after setting the condition code and both operands remain unchanged. When the difference, including the guard digit, is zero, the operands are equal. When a nonzero difference is positive or negative, the first operand is high or low, respectively.

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

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

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

Program Exceptions:

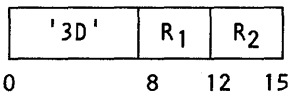
Access (fetch, operand 2 of CE and CD only)
Specification

Programming Notes

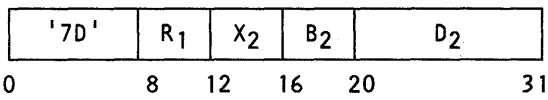
1. An exponent inequality alone is not sufficient to determine the inequality of two operands with the same sign, because the fractions may have different numbers of leading hexadecimal zeros.
2. Numbers with zero fractions compare equal even when they differ in sign or characteristic.

DIVIDE

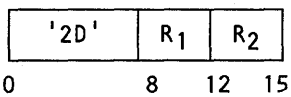
DER R_1,R_2 [RR, Short Operands]



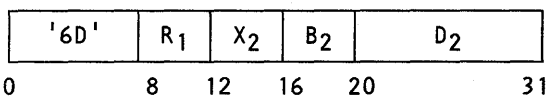
DE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



DDR R_1,R_2 [RR, Long Operands]



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



The first operand (the dividend) is divided by the second operand (the divisor), and the normalized quotient is placed in the first-operand location. No remainder is preserved.

Floating-point division consists in characteristic subtraction and fraction division. The operands are first normalized to eliminate leading hexadecimal zeros. The difference between the dividend and divisor characteristics of the normalized operands, plus 64, is used as the characteristic of an intermediate quotient.

All dividend and divisor fraction digits participate in forming the fraction of the intermediate quotient. The intermediate-quotient fraction can have no leading hexadecimal zeros, but a right-shift of one digit position may be necessary

with an increase of the characteristic by one. The fraction is then truncated to the proper result-fraction length.

An exponent-overflow exception is recognized when the characteristic of the final quotient would exceed 127 and the fraction is not zero. The operation is completed by making the characteristic 128 less than the correct value. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent overflow occurs.

An exponent-underflow exception exists when the characteristic of the final quotient would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the quotient a true zero.

Exponent underflow does not occur when an operand characteristic becomes less than zero during normalization of the operands or when the intermediate-quotient characteristic is less than zero, as long as the final quotient can be represented with the correct characteristic.

When the divisor fraction is zero, the operation is suppressed, and a program interruption for floating-point divide occurs. This includes the division of zero by zero.

When the dividend fraction is zero, but the divisor fraction is nonzero, the quotient is made a true zero. No exponent overflow or exponent underflow occurs.

The sign of the quotient is determined by the rules of algebra, except that the sign is always plus when the quotient is made a true zero.

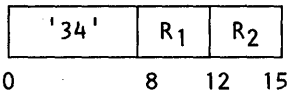
The R_1 field for DER, DE, DDR, and DD, and the R_2 field for DER and DDR, must designate register 0, 2, 4, or 6. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

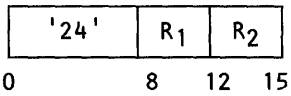
Program Exceptions:
Access (fetch, operand 2 of DD and DE only)
Exponent Overflow
Exponent Underflow
Floating-Point Divide
Specification

HALVE

HER R_1,R_2 [RR, Short Operands]



HDR R_1,R_2 [RR, Long Operands]



The second operand is divided by 2, and the normalized quotient is placed in the first-operand location.

The fraction of the second operand is shifted right one bit position, placing the contents of the rightmost bit position into the leftmost bit position of the guard digit and introducing a zero into the leftmost bit position of the fraction. The intermediate result, including the guard digit, is then normalized, and the final result is truncated to the proper length.

An exponent-underflow exception exists when the characteristic of the final result would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the result a true zero.

When the fraction of the second operand is zero, the result is made a true zero, and no exponent underflow occurs.

The sign of the result is the same as that of the second operand, except that the sign is always plus when the quotient is made a true zero.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

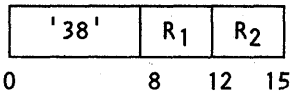
Program Exceptions:
Exponent Underflow
Specification

Programming Notes

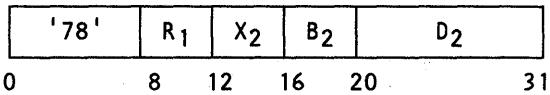
1. With short and long operands, the halve operation is identical to a divide operation with the number 2 as divisor. Similarly, the result of HDR is identical to that of MD or MDR with one-half as a multiplier. No multiply operation corresponds to HER, since no multiply operation produces short results.
2. The result of HALVE is zero only when the second-operand fraction is zero, or when exponent underflow occurs with the exponent-underflow mask set to zero. A fraction with zeros in every bit position, except for a one in the rightmost bit position, does not become zero after the right shift. This is because the one bit is preserved in the guard-digit position and becomes the leftmost bit after normalization of the result.

LOAD

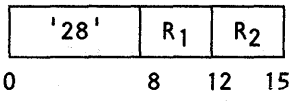
LER R_1,R_2 [RR, Short Operands]



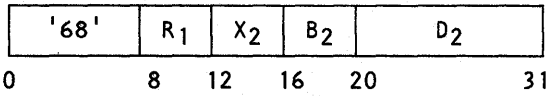
LE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



LDR R_1,R_2 [RR, Long Operands]



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



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

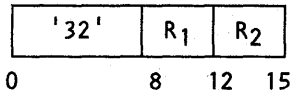
The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

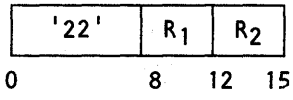
Program Exceptions:
Access (fetch, operand 2 of LE and LD only)
Specification

LOAD AND TEST

LTER R_1,R_2 [RR, Short Operands]



LTDR R_1,R_2 [RR, Long Operands]



The second operand is placed unchanged in the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

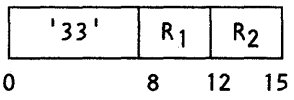
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 –

Program Exceptions:
Specification

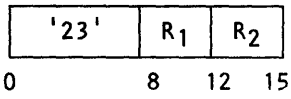
Programming Note
When the same register is specified as the first-operand and second-operand location, the operation is equivalent to a test without data movement.

LOAD COMPLEMENT

LCER R_1,R_2 [RR, Short Operands]



LCDR R_1,R_2 [RR, Long Operands]



The second operand is placed in the first-operand location with the sign bit inverted.

The sign bit is inverted, even if the fraction is zero. The characteristic and fraction are not changed.

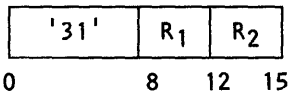
The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 –

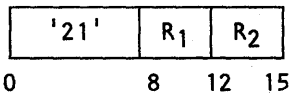
Program Exceptions:
Specification

LOAD NEGATIVE

LNER R_1,R_2 [RR, Short Operands]



LNDR R_1,R_2 [RR, Long Operands]



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

The sign bit is made one, even if the fraction is zero. The characteristic and fraction are not changed.

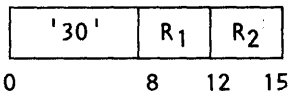
The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:
0 Result fraction is zero
1 Result is less than zero
2 –
3 –

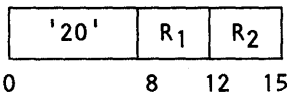
Program Exceptions:
Specification

LOAD POSITIVE

LPER R_1,R_2 [RR, Short Operands]



LPDR R_1,R_2 [RR, Long Operands]



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

The sign bit is made zero. The characteristic and fraction are not changed.

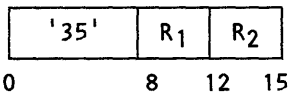
The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:
0 Result fraction is zero
1 –
2 Result is greater than zero
3 –

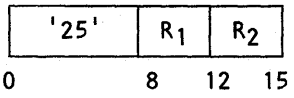
Program Exceptions:
Specification

LOAD ROUNDED

LRER R_1,R_2
[RR, Long Operand 2, Short Operand 1]



LRDR R_1,R_2
[RR, Extended Operand 2, Long Operand 1]



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

Rounding consists in adding a one in bit position 32 or 72 of the long or extended second operand, respectively, and propagating any carry to the left. The sign of the fraction is ignored, and addition is–

performed as if the fractions were positive.

If rounding causes a carry out of the leftmost hexadecimal digit position of the fraction, the fraction is shifted right one digit position so that the carry becomes the leftmost digit of the fraction, and the characteristic is increased by one.

The sign of the result is the same as the sign of the second operand. There is no normalization to eliminate leading zeros.

An exponent-overflow exception exists when shifting the fraction right would cause the characteristic to exceed 127. The operation is completed by loading a number whose characteristic is 128 less than the correct value, and a program interruption for exponent overflow occurs. The result is normalized, and the sign and fraction remain correct.

Exponent-underflow and significance exceptions cannot occur.

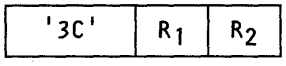
The R_1 field must designate register 0, 2, 4, or 6; the R_2 field of LRER must designate register 0, 2, 4, or 6; and the R_2 field of LRDR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

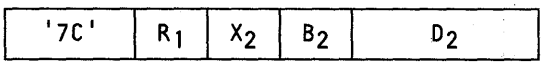
Program Exceptions:
Exponent Overflow
Specification

MULTIPLY

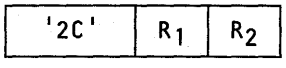
MER R_1,R_2
[RR, Short Multiplier and Multiplicand, Long Product]



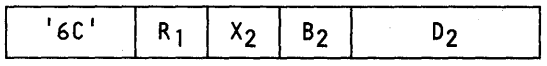
ME $R_1,D_2(X_2,B_2)$
[RX, Short Multiplier and Multiplicand, Long Product]



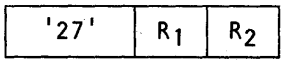
MDR R_1,R_2 [RR, Long Operands]



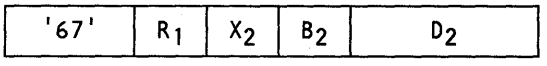
MD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]



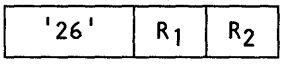
MXDR R_1,R_2
[RR, Long Multiplier and Multiplicand, Extended Product]



MXD $R_1,D_2(X_2,B_2)$
[RX, Long Multiplier and Multiplicand, Extended Product]



MXR R_1,R_2 [RR, Extended Operands]



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

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

The fraction of the intermediate product is the exact product of the normalized operand fractions. When the intermediate-product fraction has one leading hexadecimal zero digit, the fraction is shifted left one digit position, bringing the contents of the guard-digit position into the rightmost position of the result fraction, and the intermediate-product characteristic is reduced by one. The fraction is then truncated to the proper result-fraction length.

For MER and ME, the multiplier and multiplicand fractions have six hexadecimal digits;

the product fraction has the full 14 digits of the long format, with the two rightmost fraction digits always zeros. For MDR and MD, the multiplier and multiplicand fractions have 14 digits, and the final product fraction is truncated to 14 digits. For MXDR and MXD, the multiplier and multiplicand fractions have 14 digits, with the multiplicand occupying the high-order part of the first operand; the final product fraction contains 28 digits and is an exact product of the operand fractions. For MXR, the multiplier and multiplicand fractions have 28 digits, and the final product fraction is truncated to 28 digits.

An exponent-overflow exception is recognized when the characteristic of the final product would exceed 127 and the fraction is not zero. The operation is completed by making the characteristic 128 less than the correct value. If, for extended results, the low-order characteristic would also exceed 127, it, too, is decreased by 128. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent overflow occurs.

Exponent overflow is not recognized when the intermediate-product characteristic is initially 128 but is brought back within range by normalization.

An exponent-underflow exception exists when the characteristic of the final product would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, program interruption does not take place; instead, the operation is completed by making the product a true zero. For extended results, exponent underflow is not recognized when the low-order characteristic would be less than zero but the high-order characteristic is equal to or greater than zero.

Exponent underflow does not occur when the characteristic of an operand becomes less than zero during normalization of the operands, as long as the final product can be represented with the correct characteristic.

When either or both operand fractions are zero, the result is made a true zero, and no exponent overflow or exponent underflow occurs.

The sign of the product is determined by the rules of algebra, except that the sign is always zero when the result is made a true zero.

The R_1 field for MER, ME, MDR, and MD, and the R_2 field for MER, MDR, and MXDR must designate register 0, 2, 4, or 6. The R_1 field for MXDR, MXD, and MXR, and the R_2 field for MXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

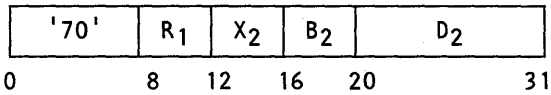
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2 of ME, MD, and MXD only)
Exponent Overflow
Exponent Underflow
Specification

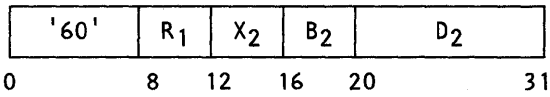
Programming Note
Interchanging the two operands in a floating-point multiplication does not affect the value of the product.

STORE

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



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



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

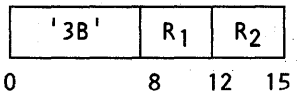
The R_1 field must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

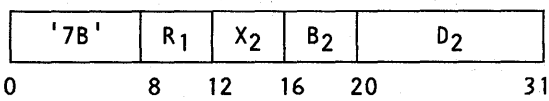
Program Exceptions:
Access (store, operand 2)
Specification

SUBTRACT NORMALIZED

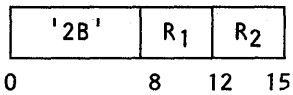
SER R_1,R_2 [RR, Short Operands]



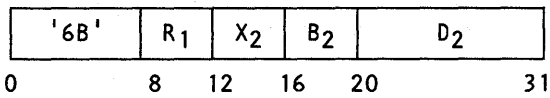
SE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



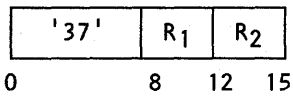
SDR R_1,R_2 [RR, Long Operands]



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



SXR R_1,R_2 [RR, Extended Operands]



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

The execution of SUBTRACT NORMALIZED is identical to that of ADD NORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

The R_1 field of SER, SE, SDR, and SD, and the R_2 field of SER and SDR must designate register 0, 2, 4, or 6. The R_1 and R_2 fields of SXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Resulting Condition Code:

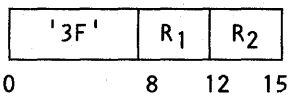
0 Result fraction is zero
1 Result is less than zero
2 Result is greater than zero
3 –

Program Exceptions:

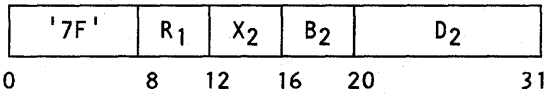
Access (fetch, operand 2 of SE and SD only)
Exponent Overflow
Exponent Underflow
Significance
Specification

SUBTRACT UNNORMALIZED

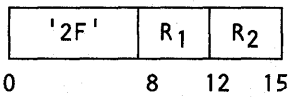
SUR R_1,R_2 [RR, Short Operands]



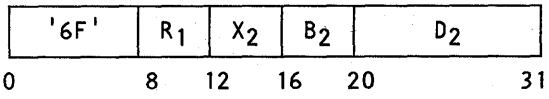
SU $R_1,D_2(X_2,B_2)$ [RX, Short Operands]



SWR R_1,R_2 [RR, Long Operands]



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



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

The execution of SUBTRACT UNNORMALIZED is identical to that of ADD UNNORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

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

Program Exceptions:

Access (fetch, operand 2 of SU and SW only)
Exponent Overflow
Significance
Specification

Chapter 10. Control Instructions

Contents

The control instructions include all privileged instructions, except the input/output instructions, which are described in Chapter 12, "Input/Output Operations."

Privileged instructions may be executed only when the CPU is in the supervisor state. An attempt to execute a privileged instruction in the problem state generates a privileged-operation exception.

The control instructions and their mnemonics, formats, and operation codes are listed in the figure "Control Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

Note: *In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For LOAD PSW, for example, LPSW is the mnemonic and $D_2(B_2)$ the operand designation.*

Name	Mnemonic	Characteristics							Op Code
CLEAR PAGE	CLRP	S		P	A[1]		PS	ST	B215
CONNECT PAGE	CTP	RS	C	P	A[1]	SP	PT	R	B0
DECONFIGURE PAGE	DEP	S		P	A[1]	SP	PT		B21B
DIAGNOSE				P	DM				83
DISCONNECT PAGE	DCTP	S	C	P	A[1]	SP	PT		B21C
INSERT PAGE BITS	IPB	RS		P	A[1]			R	B4
INSERT PSW KEY	IPK	S		P				R	B20B
INSERT STORAGE KEY	ISK	RR		P	A[1]	SP		R	09
LOAD CONTROL	LCTL	RS		P	A	SP			B7
LOAD FRAME INDEX	LFI	RS	C	P				R	B8
LOAD PSW	LPSW	S	L	P	A	SP	$		82
MAKE ADDRESSABLE	MAD	S	C	P	A[1]		PT		B21D
MAKE UNADDRESSABLE	MUN	S	C	P	A[1]	SP	PT		B21E
RESET REFERENCE BIT	RRB	S	C	P	A[1]				B213
RETRIEVE STATUS AND PAGE	RSP	SS	C	P	A			ST	D8
SET CLOCK	SCK	S	C	P	A	SP			B204
SET CLOCK COMPARATOR	SCKC	S		P	A	SP			B206
SET CPU TIMER	SPT	S		P	A	SP			B208
SET PAGE BITS	SPB	RS	C	P	A[1]				B5
SET PSW KEY FROM ADDRESS	SPKA	S		P					B20A
SET STORAGE KEY	SSK	RR		P	A[1]	SP			08
SET SYSTEM MASK	SSM	S		P	A	SP	SO		80
STORE CAPACITY COUNTS	STCAP	S		P	A			ST	B21F
STORE CLOCK COMPARATOR	STCKC	S		P	A	SP		ST	B207
STORE CONTROL	STCTL	RS		P	A	SP		ST	B6
STORE CPU ID	STIDP	S		P	A	SP		ST	B202
STORE CPU TIMER	STPT	S		P	A	SP		ST	B209
STORE THEN AND SYSTEM MASK	STNSM	SI		P	A			ST	AC
STORE THEN OR SYSTEM MASK	STOSM	SI		P	A	SP		ST	AD

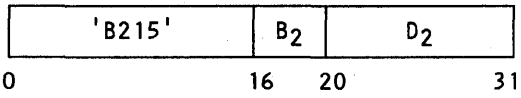
Explanation:

$ Causes serialization
A Access exceptions
A[1] Access exceptions; not all access exceptions may occur; see instruction description for details.
C Condition code is set
DM DIAGNOSE may generate various program exceptions and may change the condition code
L New condition code loaded
P Privileged-operation exception
PS Page-state exception
PT Page-transition exception
R PER general-register-alteration event
RR RR instruction format
RS RS instruction format
S S instruction format
SI SI instruction format
SO Special-operation exception
SP Specification exception
SS SS instruction format
ST PER storage-alteration event

Summary of Control Instructions

CLEAR PAGE

CLRP $D_2(B_2)$ [S]



The storage page designated by the second-operand address is cleared, which is equivalent to storing 2,048 zero bytes at that location. The page is validated.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

The page may be addressable or connected; if the page is disconnected, a page-state exception is raised, and the operation is suppressed.

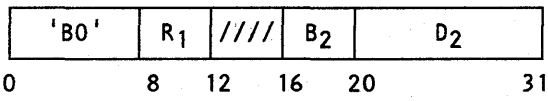
Condition Code: The code remains unchanged.

Program Exceptions:
Addressing (operand 2)
Page State
Privileged Operation
Protection (store, operand 2)

Programming Note
Page 0 may be cleared, but it can only be in the addressable state.

CONNECT PAGE

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



If disconnected, the storage page designated by the second-operand address enters the connected state. If already connected, the page remains in the connected state. The frame index of the page frame that is connected to the page is returned in the general register designated by the R_1 field.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored. Bits 12-15 of the instruction are ignored.

If bits 8-20 of the second-operand address are zeros, that is, page 0 is specified, a specification exception is recognized, and the operation is suppressed.

If the page is in the addressable state, a page-transition exception is recognized, and the operation is suppressed.

If the operation is not successful, because the page is disconnected but no page frame is free for connection (free-frame-capacity count is zero), the R_1 register remains unchanged, and condition code 2 is set.

If the operation is successful, the condition code indicates whether the page was connected (1) or disconnected (0) at the start of the operation. The frame index, which is an unsigned binary integer, is loaded right-aligned in the R_1 register, and the remaining high-order bits of the register are set to zeros. The frame index is unique and may have any value from zero to EFCC - 1, where EFCC is the existing-frame-capacity count.

If the page was disconnected before and the operation is successful, the value of the free-frame-capacity count is decreased by one.

The contents of a newly connected page frame are unpredictable.

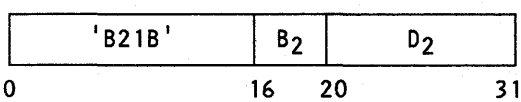
Resulting Condition Code:
0 Successful, page was disconnected, index returned
1 Page was already connected, index returned
2 Not successful, index not returned
3 –

Program Exceptions:
Addressing (operand 2)
Page Transition
Privileged Operation
Specification

Programming Notes
1. The storage key and the reference, change, and page bits of a page are not changed when the page is connected.
2. The frame index of the page frame connected to the specified page remains unchanged until that page is disconnected. The value of the frame index to be assigned by CONNECT PAGE to a previously disconnected page is unpredictable.

DECONFIGURE PAGE

DEP $D_2(B_2)$ [S]



If connected, the storage page designated by the second-operand address enters the disconnected state. The page frame that was connected to the page becomes unavailable; that is, it will no longer be available for connection to any page. The reference and change bits of the page are set to zeros.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

If bits 8-20 of the second-operand address are zeros, that is, page 0 is specified, a specification exception is recognized, and the operation is suppressed.

The page must be in the connected state at the start of the operation; otherwise, a page-transition exception is recognized, and the operation is suppressed.

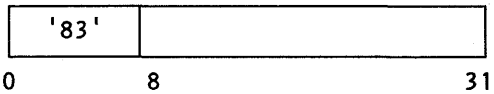
The value of the available-frame-capacity count is decreased by one. The values of the free-frame and existing-frame-capacity counts remain unchanged.

Condition Code: The code remains unchanged.

Program Exceptions:
Addressing (operand 2)
Page Transition
Privileged Operation
Specification

Programming Notes
1. DECONFIGURE PAGE allows a program to put a page frame out of operation. This may be desirable when the page frame is indicated as defective by a machine check which is caused by a storage access to the page connected to that frame or by an access to the associated storage key. The frame may become available again during a subsequent manual clear-reset operation.
2. The instruction cannot be used on the frame connected to page 0 because page 0 cannot be in the disconnected state.

DIAGNOSE



The CPU performs built-in diagnostic functions, or other model-dependent functions. The purpose of the diagnostic functions is to verify proper functioning of CPU equipment and to locate faulty components. Other model-dependent functions may include disabling of failing buffers, reconfiguration of storage and channels, and modification of control storage.

Bits 8-31 may be used as in the SI or RS formats, or in some other way, to specify the particular diagnostic function. The use depends on the model.

The execution of the instruction may affect the state of the CPU and the contents of a register or storage location, as well as the progress of an I/O operation. Some diagnostic functions may cause the test indicator to be turned on.

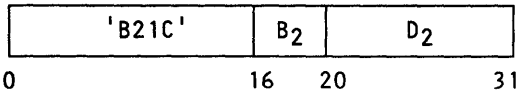
Condition Code: The code is unpredictable.

Program Exceptions:
Privileged Operation
Depending on the model, other exceptions may be recognized.

Programming Notes
1. Since the instruction is not intended for problem-program or supervisor-program use, DIAGNOSE has no mnemonic.
2. DIAGNOSE, unlike other instructions, does not follow the rule that programming errors are distinguished from equipment errors. Improper use of DIAGNOSE may result in false machine-check indications or may cause actual machine malfunctions to be ignored. It may also alter other aspects of system operation, including instruction execution and channel operation, to an extent that the operation does not comply with that specified in this publication. As a result of the improper use of DIAGNOSE, the system may be left in such a condition that the power-on reset or initial-microprogram-loading (IML) function must be performed. Since the function performed by DIAGNOSE may differ from model to model and between versions of a model, the program should avoid issuing DIAGNOSE unless the program recognizes both the model number and version code stored by STORE CPU ID.

DISCONNECT PAGE

DCTP $D_2(B_2)$ [S]



If connected, the storage page designated by the second-operand address enters the disconnected state. If already disconnected, the page remains in the disconnected state. The reference and change bits of the page are set to zeros.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

If bits 8-20 of the second-operand address are zeros, that is, page 0 is specified, a specification exception is recognized, and the operation is suppressed.

If the page is in the addressable state, a page-transition exception is recognized, and the operation is suppressed.

The condition code indicates whether the page was connected (0) or disconnected (1) before. If the page was connected before, the value of the free-frame-capacity count is increased by one.

The contents of the disconnected page frame are not necessarily cleared by the machine. The next time this frame is connected to a page by some CONNECT instruction, its contents will be unpredictable.

Resulting Condition Code:

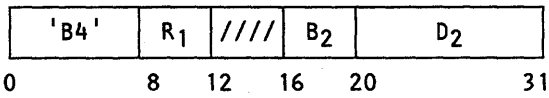
0 Page was connected
1 Page was already disconnected
2 –
3 –

Program Exceptions:

Addressing (operand 2)
Page Transition
Privileged Operation
Specification

INSERT PAGE BITS

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



The current settings of the three programmable page bits and the reference and change bits that are associated with the storage page designated by the second-operand address are inserted in the general register designated by the R_1 field.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored. Bits 12-15 of the instruction are ignored.

The current values of the three page bits are inserted in bit positions 25-27, and the reference and change bits in bit positions 29-30 of the register designated by the R_1 field. The contents of bit positions 24, 28, and 31 of that register are set to zeros. The contents of bit positions 0-23 remain unchanged.

The references to the page bits and to the reference and change bits are not subject to a protection exception. These bits can be accessed regardless of the state of the addressed page.

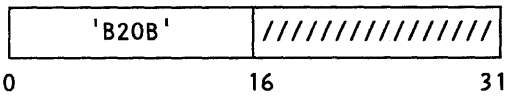
Condition Code: The code remains unchanged.

Program Exceptions:

Addressing (operand 2)
Privileged Operation

INSERT PSW KEY

IPK [S]



The four-bit PSW-key, bits 8-11 of the current PSW, is inserted in bit positions 24-27 of general register 2, and bits 28-31 of that register are set to zeros. Bits 0-23 of general register 2 remain unchanged.

Bits 16-31 of the instruction are ignored.

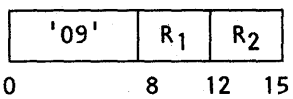
Resulting Condition Code: The code remains unchanged.

Program Exceptions:

Privileged Operation

INSERT STORAGE KEY

ISK R_1,R_2 [RR]



The storage key associated with the page that is addressed by the contents of the general register

designated by the R_2 field is inserted in the general register designated by the R_1 field.

Bits 8-20 of the register designated by the R_2 field designate the page. Bits 0-7 and 21-27 of the register are ignored. Bits 28-31 of the register must be zeros; otherwise, a specification exception is recognized, and the operation is suppressed.

The execution of the instruction depends on whether the PSW specifies the EC or BC mode. In the EC mode, the seven-bit storage key is inserted in bit positions 24-30 of the register designated by the R_1 field, and bit 31 is set to zero. In the BC mode, bits 0-4 of the storage key are placed in bit positions 24-28 of that register, and bits 29-31 of the register are set to zeros. In both modes, the contents of bit positions 0-23 of the register remain unchanged.

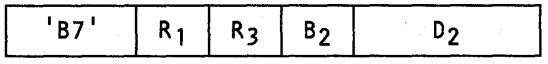
The reference to the storage key is not subject to a protection exception. The storage key can be accessed regardless of the state of the addressed page.

Condition Code: The code remains unchanged.

Program Exceptions:
Addressing (operand 2)
Privileged Operation
Specification

LOAD CONTROL

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



The set of control registers starting with the control register designated by the R_1 field and ending with the control register designated by the R_3 field is loaded from the locations designated by the second-operand address.

The storage area from which the contents of the control registers are obtained starts at the location designated by the second-operand address and continues through as many storage words as the number of control registers specified. The control registers are loaded in ascending order of their addresses, starting with the control register designated by the R_1 field and continuing up to and including the control register designated by the R_3 field, with control register 0 following control register 15. The second operand remains unchanged.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized, and the operation is suppressed.

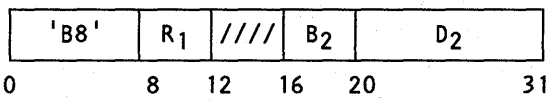
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)
Privileged Operation
Specification

Programming Note
To ensure that existing programs run if and when new facilities using additional control-register positions are defined, only zeros should be loaded in unassigned control-register positions.

LOAD FRAME INDEX

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



The frame index of the page frame that is connected to the storage page designated by the second-operand address is returned in the general register designated by the R_1 field.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored. Bits 12-15 of the instruction are ignored.

The frame index is an unsigned binary integer. It is right-aligned in the R_1 register, and the remaining high-order bits of the register are set to zeros. The frame index is unique and may have any value from zero to EFCC - 1, where EFCC is the existing-frame-capacity count.

The frame index is returned only when the page is connected or addressable. When the page is disconnected or not provided (condition codes 2 or 3), the R_1 register remains unchanged.

Condition code 0, 1, or 2 is set when the page is addressable, connected, or disconnected, respectively. Condition code 3 is set when the address is invalid, that is, the value of bits 8-20 of the second-operand address equals or exceeds the page-capacity count.

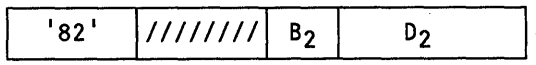
Resulting Condition Code:
0 Index returned, page is addressable
1 Index returned, page is connected
2 Index not returned, page is disconnected
3 Index not returned, address is invalid

Program Exceptions:
Privileged Operation

Programming Note
The instruction may be used to test the page address and state of a page and return its frame index, if any, without raising an access exception.

LOAD PSW

LPSW $D_2(B_2)$ [S]



The current PSW is replaced by the contents of the doubleword at the location designated by the second-operand address.

If the new PSW specifies the BC mode, information in bit positions 16-33 of the new PSW is not retained as the PSW is loaded. When the PSW is subsequently stored, these bit positions contain the new interruption code and the instruction-length code.

A serialization function is performed.

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on addressing and protection exceptions.

The value which is to be loaded by the instruction is not checked for validity before it is loaded. However, immediately after loading, a specification exception is recognized and a program interruption occurs when the newly loaded PSW specifies the EC mode and the contents of bit positions 0, 2-5, 16-17, and 24-29 are not all zeros. In these cases, the operation is completed, and the resulting instruction-length code is zero.

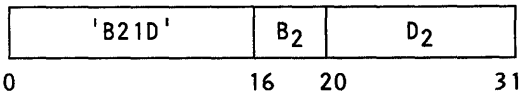
Bits 8-15 of the instruction are ignored.

Condition Code: The code is set as specified in the new PSW loaded.

Program Exceptions:
Access (fetch, operand 2)
Privileged Operation
Specification

MAKE ADDRESSABLE

MAD $D_2(B_2)$ [S]



If connected, the storage page designated by the second-operand address enters the addressable state. If already addressable, the page remains in the addressable state.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

If the page is in the disconnected state, a page-transition exception is recognized, and the operation is suppressed.

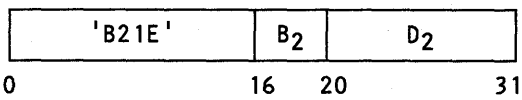
The condition code indicates whether the page was addressable (1) or connected (0) before.

Resulting Condition Code:
0 Page was connected
1 Page was already addressable
2 –
3 –

Program Exceptions:
Addressing (operand 2)
Page Transition
Privileged Operation

MAKE UNADDRESSABLE

MUN $D_2(B_2)$ [S]



If addressable, the storage page designated by the second-operand address enters the connected state. If already connected, the page remains in the connected state.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

If bits 8-20 of the second-operand address are zeros, that is, page 0 is specified, a specification exception is recognized, and the operation is suppressed.

If the page is in the disconnected state, a page-transition exception is recognized, and the operation is suppressed.

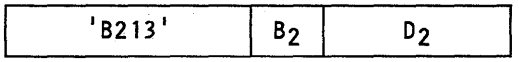
The condition code indicates whether the page was addressable (0) or connected (1) before.

Resulting Condition Code:
0 Page was addressable
1 Page was already connected
2 –
3 –

Program Exceptions:
Addressing (operand 2)
Page Transition
Privileged Operation
Specification

RESET REFERENCE BIT

RRB $D_2(B_2)$ [S]



The reference bit in the storage key associated with the storage page that is designated by the second-operand address is set to zero.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

The reference to the storage key is not subject to a protection exception. The storage key can be accessed regardless of the state of the addressed page.

The values of the remaining bits of the storage key, including the change bit, are not affected.

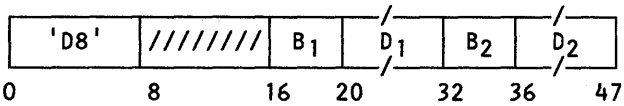
The condition code is set to reflect the state of the reference and change bits before the reference bit is set to zero.

Resulting Condition Code:
0 Reference bit zero, change bit zero
1 Reference bit zero, change bit one
2 Reference bit one, change bit zero
3 Reference bit one, change bit one

Program Exceptions:
Addressing (operand 2)
Privileged Operation

RETRIEVE STATUS AND PAGE

RSP $D_1(B_1),D_2(B_2)$ [SS]



The saved machine status is retrieved and stored at the first-operand location. The contents of the saved page are retrieved and stored at the second-operand location.

The saved machine status, as retrieved, consists of 256 bytes reflecting the state of the machine at the last time that the manual machine-save operation was performed. (See the figure "Machine Status, Retrieval Format" in Chapter 4, "Control," for the contents.) The saved page consists of the contents at that time of page 0. The storage key, page bits, and frame index for the saved page are contained in the machine status.

If the two operands overlap, the results are unpredictable.

If the saved information is valid, condition code 0 is set. If the saved information is invalid, neither storage operand is accessed, no access exceptions are recognized, and condition code 3 is set.

The saved machine status and page remain unchanged.

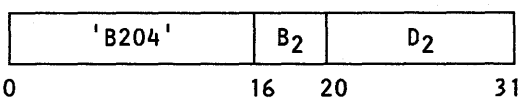
Resulting Condition Code:
0 Save information is valid
1 –
2 –
3 Save information is invalid

Program Exceptions:
Access (store, operand 1 and 2)
Privileged Operation

Programming Notes
1. The saved information may be found invalid if a partially performed machine save was canceled by resetting the machine. The saved information is invalid if a clear reset has been performed since the last machine save. RETRIEVE STATUS AND PAGE will indicate an invalid save until another machine save is performed.
2. Two executions of RETRIEVE STATUS AND PAGE will retrieve the same status and page information, as long as the information has not been made invalid by a reset and no machine save has intervened.

SET CLOCK

SCK $D_2(B_2)$ [S]



The current value of the time-of-day clock is replaced by the contents of the doubleword designated by the second-operand address, and the clock enters the set state.

The doubleword operand replaces the contents of the clock, as determined by the resolution of the clock. Only those bits of the operand are set in the clock that correspond to the bit positions which are updated by the clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock.

The value of the clock is changed and the clock is placed in the set state only if the manual TOD-clock control is set to enable-set. If the TOD-clock control is set to secure, the value and the state of the clock are not changed. The two results are distinguished by condition codes 0 and 1, respectively.

When the clock is not operational, the value and state of the clock are not changed, regardless of the setting of the TOD-clock control, and condition code 3 is set.

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed.

Resulting Condition Code:

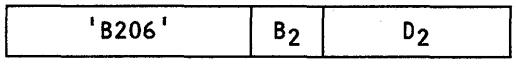
0 Clock value set
1 Clock value secure
2 –
3 Clock in not-operational state

Program Exceptions:

Access (fetch, operand 2)
Privileged Operation
Specification

SET CLOCK COMPARATOR

SCKC $D_2(B_2)$ [S]



The current value of the clock comparator is replaced by the contents of the doubleword designated by the second-operand address.

Only those bits of the operand are set in the clock comparator that correspond to the bit positions to be compared with the time-of-day clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock comparator.

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on addressing and protection exceptions.

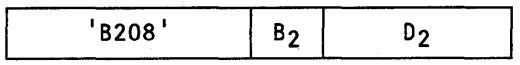
Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)
Privileged Operation
Specification

SET CPU TIMER

SPT $D_2(B_2)$ [S]



The current value of the CPU timer is replaced by the contents of the doubleword designated by the second-operand address.

Only those bits of the operand are set in the CPU timer that correspond to the bit positions to be updated; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the CPU timer.

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on addressing and protection exceptions.

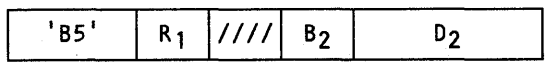
Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)
Privileged Operation
Specification

SET PAGE BITS

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



The current settings of the three programmable page bits and the reference and change bits that are

associated with the storage page designated by the second-operand address are replaced by the contents of the general register designated by the R_1 field.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored. Bits 12-15 of the instruction are ignored.

The condition code is set to reflect the state of the reference and change bits before these bits are modified.

The new values of the three page bits are obtained from bit positions 25-27, and the reference and change bits from bit positions 29-30 of the register designated by the R_1 field. The contents of bit positions 0-24, 28, and 31 of the register are ignored.

The references to the page bits and to the reference and change bits are not subject to a protection exception. These bits can be accessed regardless of the state of the addressed page.

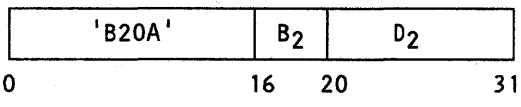
Resulting Condition Code:

0 Reference bit zero, change bit zero
1 Reference bit zero, change bit one
2 Reference bit one, change bit zero
3 Reference bit one, change bit one

Program Exceptions:
Addressing (operand 2)
Privileged Operation

SET PSW KEY FROM ADDRESS

SPKA $D_2(B_2)$ [S]



The four-bit PSW key, bits 8-11 of the current PSW, is replaced by bits 24-27 of the second-operand address.

The second-operand address is not used to address data; instead, bits 24-27 of the address form the new PSW key. Bits 8-23 and 28-31 of the second-operand address are ignored.

Condition Code: The code remains unchanged.

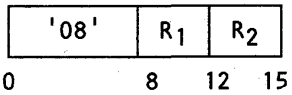
Program Exceptions:
Privileged Operation

Programming Notes

1. The format of the SET PSW KEY FROM ADDRESS instruction permits the program to set the PSW key either from the general register designated by the B_2 field or from the D_2 field in the instruction itself.
2. When a problem program requests a control program to access a location specified by the problem program, the SET PSW KEY FROM ADDRESS instruction can be used by the control program to verify that the problem program is authorized to make this access, provided the storage location of the control program is not protected against fetching. The control program can perform the verification by replacing the PSW key of the control program with the problem-program PSW key before making the access and subsequently restoring the control-program PSW key to its original value.

SET STORAGE KEY

SSK R_1,R_2 [RR]



The storage key associated with the page that is addressed by the contents of the general register designated by the R_2 field is replaced by the contents of the general register designated by the R_1 field.

Bits 8-20 of the register designated by the R_2 field designate the page. Bits 0-7 and 21-27 of the register are ignored. Bits 28-31 of the register must be zeros; otherwise, a specification exception is recognized, and the operation is suppressed.

The new seven-bit storage-key value is obtained from bit positions 24-30 of the register designated by the R_1 field. The contents of bit positions 0-23 and 31 of the register are ignored.

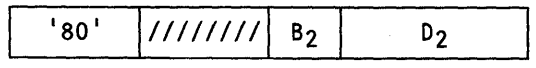
The reference to the storage key is not subject to a protection exception. The storage key can be accessed regardless of the state of the addressed page.

Condition Code: The code remains unchanged.

Program Exceptions:
Addressing (operand 2)
Privileged Operation
Specification

SET SYSTEM MASK

SSM $D_2(B_2)$ [S]



Bits 0-7 of the current PSW are replaced by the byte at the location designated by the second-operand address.

When the SSM-suppression bit, bit 1 of control register 0, is one and the CPU is in the supervisor state, a special-operation exception is recognized, and the operation is suppressed.

The operation is suppressed on protection and addressing exceptions.

The value to be loaded into the PSW is not checked for validity before loading. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, if the CPU is in EC mode and the contents of bit positions 0 and 2-5 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2. The specification exception in this case is considered to be caused as part of the execution of the instruction.

Bits 8-15 of the instruction are ignored.

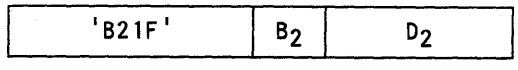
Condition Code: The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)
Privileged Operation
Special Operation
Specification

Programming Note
The SSM instruction is frequently used in the BC mode to disable or enable the CPU for I/O or external interruptions. Hence, suppressing the execution of the SSM instruction by means of the SSM-suppression bit, bit 1 of control register 0, may be useful when converting a program written for a BC-mode PSW to operate with an EC-mode PSW.

STORE CAPACITY COUNTS

STCAP $D_2(B_2)$ [S]



The current values of the page-capacity (PCC), existing-frame-capacity (EFCC), available-frame-capacity (AFCC), and free-frame-capacity (FFCC) counts are stored at the 16-byte location designated by the second-operand address. The counts are stored as 32-bit unsigned binary integers in the order, from left to right, of PCC, EFCC, AFCC, and FFCC.

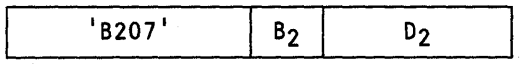
Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)
Privileged Operation

Programming Notes
1. The instruction allows the program to display the current values of the PCC, EFCC, AFCC, and FFCC for initialization purposes at IPL time and for the management of virtual storage and machine storage.
2. The high-order 16 bits of each counter value, as stored, are always zeros. The counter values cannot exceed 65,535.

STORE CLOCK COMPARATOR

STCKC $D_2(B_2)$ [S]



The current value of the clock comparator is stored at the doubleword location designated by the second-operand address.

Zeros are provided for the rightmost bit positions of the clock comparator that are not compared with the time-of-day clock.

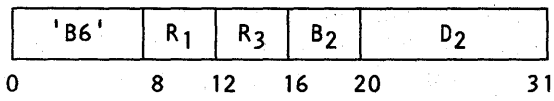
The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)
Privileged Operation
Specification

STORE CONTROL

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



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

The storage area where the contents of the control registers are placed starts at the location designated by the second-operand address and continues through as many storage words as the number of control registers specified. The contents of the control registers are stored in ascending order of their addresses, starting with the control register designated by the R_1 field and continuing up to and including the control register designated by the R_3 field, with control register 0 following control register 15. The contents of the control registers remain unchanged.

The information stored for unassigned control-register positions is unpredictable.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized, and the operation is suppressed.

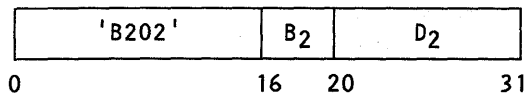
Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)
Privileged Operation
Specification

Programming Note
Although STORE CONTROL may provide zeros in the bit positions corresponding to the unassigned register positions, the program should not depend on such zeros.

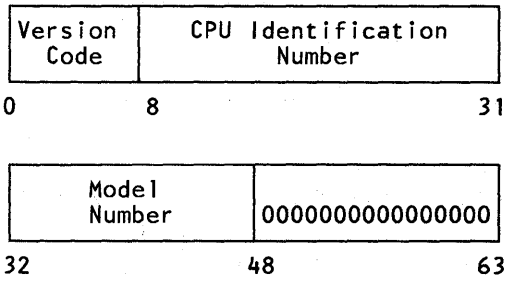
STORE CPU ID

STIDP $D_2(B_2)$ [S]



Information identifying the CPU is stored at the doubleword location designated by the second-operand address.

The format of the information is as follows:



Bit positions 0-7 contain the version code, which is information to supplement the model number.

Bit positions 8-31 contain the CPU identification number, consisting of six digits: a high-order zero digit and five digits selected from the physical serial number stamped on the CPU, or six digits selected from the serial number. The contents of the CPU identification-number field, in conjunction with the model number, permit unique identification of the CPU.

Bit positions 32-47 contain the model number of the CPU. Bit position 48-63 contain zeros.

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on addressing and protection exceptions.

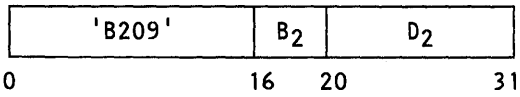
Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)
Privileged Operation
Specification

Programming Notes
1. The program should allow for the possibility that the CPU identification number may contain the digits A-F as well as the digits 0-9.
2. The CPU identification number, combined with the model number, provides a unique CPU identification that can be used in associating results with an individual machine, particularly in regard to functional differences, performance differences, and error handling.

STORE CPU TIMER

STPT $D_2(B_2)$ [S]



The current value of the CPU timer is stored at the doubleword location designated by the second-operand address.

Zeros are provided for the rightmost bit positions that are not updated by the CPU timer.

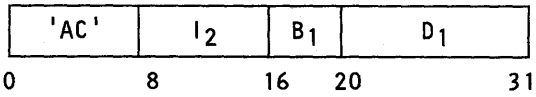
The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 2)
Privileged Operation
Specification

STORE THEN AND SYSTEM MASK

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



Bits 0-7 of the current PSW are stored at the first-operand location. Then the contents of bit positions 0-7 of the current PSW are replaced by the logical AND of their original contents and the second operand.

The operation is suppressed on addressing and protection exceptions.

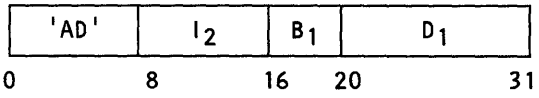
Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 1)
Privileged Operation

Programming Note
The STORE THEN AND SYSTEM MASK instruction permits the program to set selected bits in the system mask to zeros while retaining the original contents for later restoration. For example, it may be necessary that a program, which has no record of the present status, disable program-event recording for a few instructions.

STORE THEN OR SYSTEM MASK

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



Bits 0-7 of the current PSW are stored at the first-operand location. Then the contents of bit positions 0-7 of the current PSW are replaced by the logical OR of their original contents and the second operand.

The value to be loaded into the PSW is not checked for validity before loading. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, if the CPU is in the EC mode and the contents of bit positions 0 and 2-5 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2. The specification exception in this case is considered to be caused as part of the execution of the instruction.

The operation is suppressed on addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:
Access (store, operand 1)
Privileged Operation
Specification

Programming Note
The STORE THEN OR SYSTEM MASK instruction permits the program to set selected bits in the system mask to ones while retaining the original contents for later restoration. For example, the program may enable the CPU for I/O interruptions without having available the current status of the external-mask bit.

Chapter 11. Machine-Check Handling

Contents

The machine-check-handling mechanism provides extensive equipment-malfunction detection to ensure the integrity of system operation and to permit automatic recovery from some malfunctions. Equipment malfunctions and certain external disturbances are reported by means of a machine-check interruption to assist in program-damage assessment and recovery. The interruption supplies the program with information about the extent of the damage and the location and nature of the cause. Equipment malfunctions, errors, and other situations which can cause machine-check interruptions are referred to as machine checks.

Machine-Check Detection

Machine-check-detection mechanisms may take many forms, especially in control functions for arithmetic and logical processing, addressing, sequencing, and execution. For program-addressable information, detection is normally accomplished by encoding redundancy into the information in such a manner that most failures in the retention or transmission of the information result in an invalid code. The encoding normally takes the form of one or more redundant bits, called check bits, appended to a group of data bits.

Such a group of data bits and the associated check bits are called a checking block. The size of the checking block depends on the model.

The inclusion of a single check bit in the checking block allows the detection of any single-bit failure within the checking block. In this arrangement, the check bit is sometimes referred to as a "parity bit." In other arrangements, a group of check bits is included to permit detection of multiple errors, to permit error correction, or both.

For checking purposes, the entire contents of a checking block, including the redundancy, is called a checking-block code (CBC). When a CBC completely meets the checking requirements (that is, no failure is detected), it is said to be valid. When both detection and correction are provided and a CBC is not valid but satisfies the checking requirements for correction (the failure is correctable), it is said to be near-valid. When a CBC does not satisfy the checking requirements (the failure is uncorrectable), it is said to be invalid.

Correction of Machine Malfunctions

Two mechanisms may be used to provide recovery from machine-detected malfunctions: error checking and correction, and CPU retry.

Machine failures which are corrected successfully may or may not be reported as machine-check interruptions. If reported, they are system-recovery conditions, which permit the program to note the cause of CPU delay and to keep a log of such incidents.

Error Checking and Correction

When sufficient redundancy is included in circuitry or in a checking block, failures can be corrected. For example, circuitry can be triplicated, with a voting circuit to determine the correct value by selecting two matching results out of three, thus correcting a single failure. An arrangement for correction of failures of one order and for detection of failures of a higher order is called error checking and correction (ECC). Commonly, ECC allows correction of single-bit failures and detection of double-bit failures.

Depending on the model and the portion of the machine in which ECC is applied, correction may be reported as a system-recovery machine-check condition or no report may be given.

Uncorrected errors in storage and in the storage key may be reported, along with a failing-storage address, to indicate where the error occurred. Depending on the situation, these errors may be reported along with system recovery or with the damage condition resulting from the error.

CPU Retry

Some models have the capability of correcting intermittent errors by retrying CPU operations. When a malfunction is detected, recovery is attempted by returning the CPU state to that existing at the checkpoint when information about the CPU state was last saved and by proceeding from that point.

Handling of Machine Checks

A machine check is caused by a machine malfunction and not by data or instructions. This is ensured during the power-on sequence by initializing the machine controls to a valid state and by placing valid CBC in the CPU registers, in the page descriptions, and in all available page frames.

Specification of an unavailable component, such as a channel or I/O device, does not cause a machine-check indication. Instead, such a condition is indicated by the appropriate program or I/O interruption or condition-code setting.

A machine check is indicated whenever the result of an operation could be affected by information with invalid CBC, or when any other malfunction makes it impossible to establish reliably that an operation can be, or has been, performed correctly. When information with invalid CBC is fetched but not used, the condition may or may not be indicated, and the invalid CBC is preserved.

When a machine malfunction is detected, the action taken depends on the model, the nature of the malfunction, and the situation in which the malfunction occurs. Malfunctions affecting operator-facility actions may result in machine checks or may be indicated to the operator.

A malfunction detected as part of an I/O operation may cause a machine-check condition, an I/O-error condition, or both. I/O-error conditions are indicated by an I/O interruption or by the appropriate condition-code setting during the execution of an I/O instruction. When the machine reports a failing-storage location detected during an I/O operation, both I/O-error and machine-check conditions may be presented. The I/O-error condition is the primary indication to the program. The machine-check condition is a

secondary indication, which is presented as system recovery together with a failing-storage address.

Validation

Machine errors can be generally classified as solid or intermittent, according to the persistence of the malfunction. A persistent machine error is said to be solid. In the case of a register or storage location, a third type of error must be considered, called externally generated. An externally generated error is one where no failure exists in the register or storage location but invalid CBC has been introduced into the location from something external to the location. For example, the value could be affected by a power transient, or an incorrect value may have been introduced when the information was placed in the location.

Invalid CBC is preserved as invalid when information with invalid CBC is fetched or when an attempt is made to update only a portion of the checking block. When an attempt is made to replace the contents of the entire checking block and the block contains invalid CBC, it depends on the operation and the model whether the block remains with invalid CBC or is replaced. An operation which replaces the contents of a checking block with valid CBC, while ignoring the current contents, is called a validation operation. Validation is used to introduce a valid CBC into a register or location which is suffering from an intermittent or externally generated error.

Validating a checking block does not ensure that a valid CBC will be observed the next time the checking block is accessed. If the failure is solid, validation is effective only if the information placed in the checking block is such that the failing bits are set to the value to which they fail. If an attempt is made to set the bits to the state opposite to that in which they fail, then the validation will not be effective. Thus, for a solid failure, validation is only useful to eliminate the error condition, even though the underlying failure remains, thereby reducing the exposure to additional reports. The locations, however, cannot be used, since invalid CBC will result from attempts to store other values in the location. For an intermittent failure, however, validation is useful to restore a valid CBC such that a subsequent partial store into the checking block (a store into a checking block without replacing the entire checking block) by either the CPU or a channel will be permitted.

When a checking block consists of multiple bytes in storage, or multiple bits in CPU registers, the invalid CBC can be made valid only when all of the bytes or bits are replaced simultaneously.

When an error occurs in a checking block, the original information contained in the checking block should be considered lost even after validation. Automatic register validation leaves the contents unpredictable. Programmed and manual validation of checking blocks causes the contents to be changed explicitly.

Invalid CBC in Storage

The size of the checking block in storage depends on the model but is never more than 2,048 bytes.

An attempt to store into a checking block with invalid CBC, without replacing the entire checking block, leaves the data in the checking block (including the check bits) unchanged.

When the checking block consists of multiple bytes and contains invalid CBC, special procedures are necessary to place new information into the checking block. Placing valid CBC in storage is called storage validation.

Storage validation is provided as a program function and is also provided with the manual clear-reset function. Programmed storage validation is done, one page at a time, by executing the privileged instruction CLEAR PAGE. Manual storage validation by clear reset validates all pages.

Invalid CBC in Page Descriptions

When invalid CBC is detected in a page description, a machine-check interruption may occur; depending on the circumstances, the machine-check condition may be system damage, instruction-processing damage, system recovery, or external damage. The machine-check condition may or may not be accompanied by a storage-key-error-uncorrected indication. Also, if invalid CBC in a page description is detected during an I/O operation, a channel-control check is normally indicated at the end of the I/O operation.

In addition to internal storage for page descriptions, some models may have a separate lookaside storage for the storage keys of connected or addressable pages. Each entry of such a lookaside is associated with a page frame, whereas each page description is associated with a page. A storage-key error uncorrected may be indicated only when invalid CBC is detected in the lookaside storage during a reference to the storage key of a page that is in the connected or addressable state.

A storage-key error is not indicated when:

- Invalid CBC is detected in the storage key of a disconnected page
- Invalid CBC is detected in the page bits, the page state, or the frame index of a page, whether disconnected or not
- No lookaside storage is provided for storage keys

All parts of the page descriptions are validated manually by clear reset. On models which provide lookaside storage with a separate checking block for the storage key of each connected or addressable page, executing the instruction SET STORAGE KEY sets new values for and validates the storage key after a storage-key error has been indicated. The instruction CONNECT PAGE may validate the lookaside entry of a page frame which previously had invalid CBC by using the values of the storage key from the page-description entry.

No storage-key-error-uncorrected indication is given when a machine check occurs during the execution of DECONFIGURE PAGE, DISCONNECT PAGE, LOAD FRAME INDEX, MAKE ADDRESSABLE, and MAKE UNADDRESSABLE.

Any machine-check condition which would otherwise be indicated as a storage-key error uncorrected is ignored if the access key is zero when a fetch operation takes place. Depending on the model, a storage-key error uncorrected may or may not be ignored if the access key is zero when a store operation takes place or when the instruction CLEAR PAGE is executed.

The CPU enters the check-stop state when invalid CBC is detected in the page description for page 0, and also when a page description is left in an inconsistent state after an error occurs while the page description is being updated.

Programming Note

Recovery from a storage-key error uncorrected which cannot be successfully removed by issuing SET STORAGE KEY may be attempted by issuing DECONFIGURE PAGE to delete the page frame and CONNECT PAGE to use another page frame. The previous contents of the page are lost.

Invalid CBC in Registers

When invalid CBC is detected in a CPU register, a machine-check condition may be recognized. CPU registers include the general, floating-point, and control registers, the current PSW, the time-of-day clock, the CPU timer, and the clock comparator.

When a machine-check interruption occurs, whether or not it is due to invalid CBC in a CPU register, the following actions affecting the CPU registers, other than the time-of-day-clock, are taken as part of the interruption.

1. The contents of the registers are saved in assigned storage locations. Any register which is in error is identified by a corresponding validity bit of zero in the machine-check-interruption code. Malfunctions detected during register saving do not result in additional machine-check-interruption conditions; instead, the correctness of all the information stored is indicated by the appropriate setting of the validity bits.
2. Registers with invalid CBC are then validated, their actual contents being unpredictable.

CPU registers other than the time-of-day clock are also validated manually by the clear-reset function; programmed register validation is not provided.

The time-of-day clock is not stored and is not validated during a machine-check interruption, and it has no corresponding validity bit. The clock enters the error state when a malfunction is detected in the clock. It is validated by programming when a SET CLOCK instruction changes the state of the clock from the error state to the set state. The clock is also validated manually by a power-on reset.

Check-Stop State

In certain situations it is impossible or undesirable to continue operation when a machine error occurs. In these cases, the CPU may enter the check-stop state, which is indicated by the check-stop indicator.

In general, the CPU may enter the check-stop state whenever an uncorrectable error or other malfunction occurs and the machine is unable to recognize a specific machine-check-interruption condition.

The CPU always enters the check-stop state if any of the following conditions exists:

- PSW bit 13 is zero and an exigent machine-check condition is generated.
- During the execution of an interruption due to one exigent machine-check condition, another exigent machine-check condition is detected.
- During a machine-check interruption, the machine-check-interruption code cannot be stored successfully or the new PSW be fetched successfully.

- A machine-check interruption cannot be taken because of a storage error in page 0.
- Invalid CBC is detected in the page description for page 0.
- An error occurs while a page description is being updated, leaving the page description in an inconsistent state.

There may be many other conditions for particular models when an error may cause check stop.

When the CPU is in the check-stop state, instructions and interruptions are not executed, the interval timer is not updated, and channel operations may be stopped. The time-of-day clock is normally not affected by the check-stop state. The CPU timer may or may not run in the check-stop state, depending on the error and the model. The start key and stop key are not effective in this state.

The CPU may be removed from the check-stop state by program reset.

Machine-Check Interruption

A request for a machine-check interruption, which is made pending as the result of a machine check, is called a machine-check-interruption condition. There are two major types of machine-check-interruption conditions: exigent conditions and repressible conditions.

Exigent Conditions

Exigent machine-check-interruption conditions are those in which damage has or would have occurred such that the current instruction or interruption sequence cannot safely continue. Exigent conditions are identified in the machine-check-interruption code by two bits: instruction-processing damage and system damage. In addition to indicating specific exigent conditions, the system-damage bit is used to report any malfunction or error which cannot be isolated to a less severe report.

Repressible Conditions

Repressible machine-check-interruption conditions are those in which the results of the instruction-processing sequence have not been affected. Repressible conditions can be delayed, until the completion of the current instruction or even longer, without affecting the integrity of CPU operation. Repressible conditions are of three classes: recovery, alert, and repressible damage. Each class has one or more subclasses.

A malfunction in the CPU, storage, channel, or operator facilities which has been successfully corrected or circumvented internally without logical damage is called a recovery condition. Depending on the model and the type of malfunction, some or all recovery conditions may be discarded and not reported. Recovery conditions that are reported are grouped in one subclass, system recovery.

A machine-check-interruption condition not directly related to a machine malfunction is called an alert condition. The alert conditions are grouped in two subclasses: degradation and warning.

A malfunction resulting in an incorrect state of a portion of the system not directly affecting sequential CPU operation is called a repressible-damage condition. Repressible-damage conditions are divided into three subclasses, according to the function affected: timing-facility damage, interval-timer damage, and external damage.

Programming Notes

1. Even though repressible conditions are usually reported only at normal points of interruption, they may also be reported with exigent machine-check conditions. Thus, if an exigent machine-check condition causes an instruction to be abnormally terminated and a machine-check interruption occurs to report the exigent condition, any pending repressible conditions may also be reported. The meaningfulness of the validity bits depends on what exigent condition is reported.
2. Classification of a damage condition as repressible does not imply that the damage is necessarily less severe than damage classified as an exigent condition. The distinction is whether action must be taken as soon as the damage is detected (exigent) or whether the CPU can continue processing (repressible). For a repressible condition, the current instruction can be completed before taking the machine-check interruption if the CPU is enabled; if the CPU is disabled for machine checks, the condition can safely be kept pending until the CPU is again enabled for machine checks.

   For example, the CPU may be disabled for machine-check interruptions because it is handling an earlier instruction-processing-damage interruption. If, during that time, an I/O operation encounters a storage error, that condition can be kept pending because it is not

expected to interfere with the current machine-check processing. If, however, the CPU also makes a reference to the area of storage containing the error before re-enabling machine-check interruptions, another instruction-processing-damage condition is created, which is treated as an exigent condition and causes the CPU to enter the check-stop state.

Interruption Action

A machine-check interruption causes the following actions to be taken. The PSW reflecting the point of interruption is stored as the machine-check old PSW at location 48. The contents of other registers are stored in register-save areas at locations 216-231 and 352-511. After the contents of the registers are stored in register-save areas, the registers are validated with the contents being unpredictable. A failing-storage address, if any, is stored at location 248. Then a machine-check-interruption code (MCIC) of eight bytes is placed at location 232. The new PSW is fetched from location 112.

The fields accessed during the machine-check interruption are summarized in the figure "Machine-Check-Interruption Locations."

If the machine-check-interruption code cannot be stored successfully or the new PSW cannot be fetched successfully, the CPU enters the check-stop state.

A repressible machine-check condition can initiate a machine-check interruption only if both PSW bit 13 is one and the associated subclass mask bit in control register 14 is also one. When it occurs, the interruption does not terminate the execution of the current instruction; the interruption is taken at a normal point of interruption, and no program or supervisor-call interruptions are eliminated. If the machine check occurs during the execution of a machine function, such as a CPU-timer update, the machine-check interruption takes place after the machine function has been completed.

When the CPU is disabled for a particular repressible machine-check condition, the condition remains pending. Only one repressible condition is held pending for each subclass, regardless of the number of conditions that may have been detected for that subclass.

When a repressible machine-check interruption occurs because the interruption condition is in a subclass for which the CPU is enabled, pending conditions in other subclasses may also be indicated in the same interruption code, even though the CPU is disabled for those subclasses. All indicated conditions are then cleared.

If a machine check which is to be reported as a system-recovery condition is detected during the execution of the interruption procedure due to a previous machine-check condition, the system-recovery condition may be combined with the other conditions, discarded, or held pending.

An exigent machine-check condition can cause a machine-check interruption only when PSW bit 13 is one. When it occurs, the interruption terminates the execution of the current instruction and may eliminate the program and supervisor-call interruptions, if any, that would have occurred if execution had continued. Proper execution of the interruption steps, including the storing of the old PSW and other information, depends on the nature of the malfunction. When an exigent machine-check condition occurs during the execution of a machine function, such as a

Information Stored (Fetched)	Starting Location	Length in Bytes
Old PSW	48	8
New PSW (fetched)	112	8
Machine-check-interruption code	232	8
Failing-storage address	248	4
Register-save areas		
CPU timer	216	8
Clock comparator	224	8
Floating-point registers 0, 2, 4, 6	352	32
General registers 0-15	384	64
Control registers 0-15	448	64

Machine-Check-Interruption Locations

CPU-timer update, the sequence is not necessarily completed.

If, during the execution of an interruption due to one exigent machine-check condition, another exigent machine check is detected, the CPU enters the check-stop state. If an exigent machine check is detected during an interruption due to a repressible machine-check condition, system damage is reported.

When PSW bit 13 is zero, an exigent machine-check condition causes the CPU to enter the check-stop state.

Machine-check-interruption conditions are handled in the same manner regardless of whether the wait-state bit in the PSW is one or zero: a machine-check condition causes an interruption if the CPU is enabled for that condition.

Machine checks which occur while the rate control is set to instruction step are handled in the same manner as when the control is set to process; that is, recovery mechanisms are active, and machine-check interruptions occur when allowed. Machine checks occurring during a manual operation may be indicated to the operator, may generate a system-recovery condition, may result in system damage, or may cause a check stop, depending on the model.

Every reasonable attempt is made to limit the side effects of any machine check and the associated interruption. Normally, interruptions, as well as the progress of I/O operations, remain unaffected. The malfunction, however, may affect these activities, and, if the currently active PSW has bit 13 set to one, the machine-check interruption will indicate the total extent of the damage caused, and not just the damage which originated the condition.

Point of Interruption

The point in the processing which is indicated by the interruption and used as a reference point by the machine to determine and indicate the validity of the status stored is referred to as the point of interruption.

Only certain points in the processing may be used as a point of interruption. For repressible machine-check interruptions, the point of interruption must be after one unit of operation is completed and any associated program or supervisor-call interruption is taken, and before the next unit of operation is begun.

Exigent machine-check conditions are those in which damage has or would have occurred to the instruction stream. Thus, the damage can normally be associated with a point part way though an instruction and this point is called the point of damage.

In addition to the point of interruption permitted for repressible machine-check conditions, the point of interruption for an exigent machine-check condition may also be after the unit of operation is completed but before any associated program or supervisor-call interruption occurs. In this case, a valid PSW instruction address is defined as that which would have been stored in the old PSW for the program or supervisor-call interruption. Since the operation has been terminated, the values in the result fields, other than the instruction address, are unpredictable. Thus the validity bits associated with fields which are due to be changed by the instruction stream are meaningless when an exigent machine-check condition is reported.

When the point of interruption and the point of damage due to an exigent machine-check condition are separated by a LOAD PSW or an interruption, the damage has not been isolated to a particular program, and system damage is indicated.

Programming Note

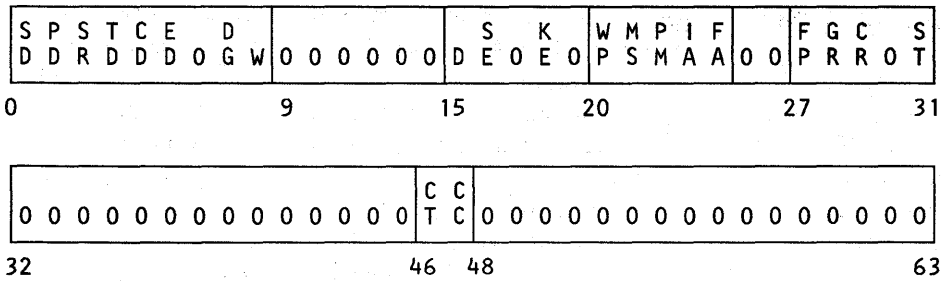
When an exigent machine-check-interruption condition occurs, the point of interruption which is chosen affects the amount of damage which must be indicated. An attempt is made, when possible, to choose a point of interruption which permits the minimum indication of damage.

When all the status information stored as a result of an exigent machine-check-interruption condition does not reflect the same point, an attempt is made when possible to choose the point of interruption so that the instruction address which is stored in the machine-check old PSW is valid.

Machine-Check-Interruption Code

On all machine-check interruptions, a machine-check-interruption code (MCIC) is stored at the doubleword starting at location 232 and has the format shown in the figure "Machine-Check Interruption-Code Format."

Bits in the MCIC which are not assigned, or not implemented by a particular model, are stored as zeros.



Bits	Name
0	System damage (SD)
1	Instruction-processing damage (PD)
2	System recovery (SR)
3	Interval-timer damage (TD)
4	Timing-facility damage (CD)
5	External damage (ED)
7	Degradation (DG)
8	Warning (W)
15	Delayed (D)
16	Storage error uncorrected (SE)
18	Storage-key error uncorrected (KE)
20	PSW-EMWP validity (WP)
21	PSW mask and key validity (MS)
22	PSW program-mask and condition-code validity (PM)
23	PSW-instruction-address validity (IA)
24	Failing-storage-address validity (FA)
27	Floating-point-register validity (FP)
28	General-register validity (GR)
29	Control-register validity (CR)
31	Storage logical validity (ST)
46	CPU-timer validity (CT)
47	Clock-comparator validity (CC)

Note: All other bits of the MCIC are unassigned and stored as zeros.

Machine-Check Interruption-Code Format

Programming Note

The program should not depend on unassigned bits in the machine-check-interruption code being zeros, so as to ensure that existing programs run if and when new facilities using these bits are defined.

Subclass

Bits 0-5, 7, and 8 are the subclass bits which identify the type of machine-check condition causing the interruption. At least one of the subclass bits is stored as a one. When multiple errors have occurred, several of the defined bits may be set to ones.

System Damage

Bit 0 (SD), when one, indicates that damage has occurred which cannot be isolated to one or more of the less severe machine-check subclasses. When system damage is indicated, the remaining bits in the machine-check-interruption code are not meaningful, and information stored in the register-save areas and failing-storage-address field is not meaningful. System damage is an exigent condition.

Instruction-Processing Damage

Bit 1 (PD), when one, indicates that damage has occurred to the instruction processing of the CPU.

For damage to be indicated as instruction-processing damage, the point of damage and the point of interruption must not be separated by an interruption or by a LOAD PSW instruction and the damaged entity must be due to be changed by the current instruction.

Instruction-processing damage is an exigent condition.

System Recovery

Bit 2 (SR), when one, indicates that malfunctions were detected but did not result in damage or have been successfully corrected. Some malfunctions detected as part of an I/O operation may result in a system-recovery condition in addition to an I/O-error condition. The presence and extent of the system-recovery capability depend on the model.

System recovery is a repressible condition.

Programming Notes

1. System recovery may be used to report a failing-storage address detected by a CPU prefetch or by an I/O operation.
2. Unless the corresponding validity bits are ones, the indication of system recovery does not imply storage logical validity, or that the fields stored as a result of the machine-check interruption are valid.

Interval-Timer Damage

Bit 3 (TD), when one, indicates that damage has occurred to the interval timer or to storage location 80. Interval-timer damage is a repressible condition.

Timing-Facility Damage

Bit 4 (CD), when one, indicates that damage has occurred to the time-of-day clock, the CPU timer, the clock comparator, or to the CPU-timer or clock-comparator external-interruption conditions. The timing-facility-damage machine-check condition is set whenever any of the following occurs:

1. The time-of-day clock enters the error or not-operational state.
2. The CPU timer is damaged, and the CPU is enabled for CPU-timer external interruptions. Depending on the model, the machine-check condition may be generated only as the CPU timer enters an error state. Or, the machine-check condition may be continuously generated whenever the CPU is enabled for CPU-timer interruptions, until the CPU timer is validated.
3. The clock comparator is damaged, and the CPU is enabled for clock-comparator external interruptions.

Timing-facility damage may also be set along with instruction-processing damage when an instruction which accesses the CPU timer or clock comparator produces incorrect results.

Timing-facility damage is a repressible condition.

Programming Note

Timing-facility-damage conditions for the CPU timer and the clock comparator are not recognized when these facilities are not in use. The facilities are considered not in use when the CPU is disabled for the corresponding external interruptions (PSW bit 7, or the subclass-mask bits, bits 20 and 21 of control register 0, are zeros), and when the corresponding set and store instructions are not being issued. Timing-facility-damage conditions that are already pending remain pending, however, when the CPU is disabled for the corresponding external interruption.

Timing-facility-damage conditions due to damage to the time-of-day clock are always recognized.

External Damage

Bit 5 (ED), when one, indicates that damage has occurred to a channel or to storage during operations not directly associated with processing the current instruction. Channel malfunctions are reported as external damage only when the channel is unable to report the malfunctions by an I/O-error condition. Depending on the model and on the type and extent of the error, an external-damage condition may be indicated as system damage instead of external damage.

External damage is a repressible condition.

Degradation

Bit 7 (DG), when one, indicates that continuous degradation of system performance, more serious than that indicated by system recovery, has occurred. Degradation may be reported when system-recovery conditions exceed a machine-preestablished threshold. The presence and extent of the degradation-report capability depends on the model.

Degradation is a repressible condition.

Warning
Bit 8 (W), when one, indicates that damage is imminent in some part of the system (for example, that power is about to fail, or that a loss of cooling is occurring). Whether warning conditions are recognized depends on the model.

If the condition responsible for the imminent damage is removed before the interruption request is honored (for example, if power is restored), the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from the same condition.

Warning is a repressible condition.

Auxiliary Bits

Bits 15, 16, and 18 of the machine-check-interruption code may occur together with one or more of the bits in the subclass field to indicate a delayed condition, an uncorrected storage error, and an uncorrected storage-key error, respectively.

Delayed
Bit 15 (D), when one, indicates that one or more of the repressible machine-check conditions being reported were delayed because, at the time a particular error was detected, the CPU was disabled for that type of interruption. The bit does not apply to exigent conditions, which cannot be delayed.

Storage Error Uncorrected
Bit 16 (SE), when one, indicates that a storage page which is in the connected or addressable state contained invalid CBC and that the information could not be corrected. The contents of the page have not been changed.

Storage-Key Error Uncorrected
Bit 18 (KE), when one, indicates invalid CBC for the storage key in lookaside storage which is associated with a storage page that is in the connected or addressable state, and indicates that the information could not be corrected. The contents of the storage key have not been changed.

Programming Note
The storage-error and storage-key-error bits do not in themselves indicate the occurrence of damage because the error detected may not have affected a result. The accompanying subclass bits of the interruption code indicate the area affected by the error.

Machine-Check Interruption-Code Validity Bits

Bits 20-24, 27-29, 31, 46, and 47 of the machine-check-interruption code are validity bits. Each bit indicates the validity of a particular field in storage. With the exception of the storage-logical-validity bit (bit 31), each bit is associated with a field stored during the machine-check interruption. When a validity bit is one, it indicates that the saved value placed in the corresponding storage field is valid with respect to the indicated point of interruption and that no error was detected when the data was stored.

When a validity bit is zero, one or more of the following conditions may have occurred: the original information was incorrect, the original information had invalid CBC, additional malfunctions were detected while storing the information, or none or only part of the information was stored. Even though the information is unpredictable, the machine will attempt, when possible, to place valid CBC in the storage field and thus reduce the possibility of additional machine checks being caused.

The validity bits for the floating-point registers, general registers, control registers, CPU timer, and clock comparator indicate the validity of the saved value placed in the corresponding save area. The information in these registers after the machine-check interruption is not necessarily correct even when the correct value has been placed in the save area and the validity bit set to one.

PSW-EMWP Validity
Bit 20 (WP), when one, indicates that the EMWP bits (bits 12-15) of the machine-check old PSW are correct.

PSW Mask and Key Validity
Bit 21, when one, indicates that the system mask, PSW key, and miscellaneous bits of the machine-check old PSW are correct. Specifically, this bit covers bits 0-11 of both EC-mode and BC-mode PSWs, and also bits 16, 17, and 24-39 of the EC-mode PSW.

PSW Program-Mask and Condition-Code Validity
Bit 22 (PM), when one, indicates that the program mask and condition code of the machine-check old PSW are correct.

PSW-Instruction-Address Validity
Bit 23 (IA), when one, indicates that the instruction address (bits 40-63) of the machine-check old PSW is correct.

Programming Note
When a machine check occurs which stores a BC-mode PSW, the contents of the interruption code and ILC in the machine-check old PSW are unpredictable, and no PSW-validity bit covers these bits. The four PSW-validity bits cover all 64 bits of the EC-mode PSW.

Failing-Storage-Address Validity
Bit 24 (FA), when one, indicates that a correct failing-storage address has been placed at location 248 after a storage error uncorrected or storage-key error uncorrected. When no such errors are reported, that is, bits 16 and 18 of the machine-check-interruption code are zeros, the failing-storage address is meaningless, even though it may be indicated as valid.

Floating-Point-Register Validity
Bit 27 (FP), when one, indicates that the contents of the floating-point-register save area at locations 352-383 reflect the correct state of the floating-point registers at the point of interruption.

General-Register Validity
Bit 28 (GR), when one, indicates that the contents of the general-register save area at locations 384-447 reflect the correct state of the general registers at the point of interruption.

Control-Register Validity
Bit 29 (CR), when one, indicates that the contents of the control-register save area at locations 448-511 reflect the correct state of the control registers at the point of interruption.

Storage Logical Validity
Bit 31 (ST), when one, indicates that the storage locations, the contents of which are modified by the instructions being executed, contain the correct information relative to the point of interruption. That is, all stores before the point of interruption are completed, and all stores, if any, after the point of interruption are suppressed. When a store before the point of interruption is suppressed because of an invalid CBC, the storage-logical-validity bit may be indicated as one, provided that the invalid CBC has been preserved as invalid.

Storage logical validity reflects only the instruction-processing activity and does not reflect errors in the state of storage as the result of interval-timer update or I/O operations, or of the storing of the old PSW and other interruption information.

CPU-Timer Validity
Bit 46 (CT), when one, indicates that the CPU timer is not in error and that the contents of the CPU-timer save area at location 216 reflect the correct state of the CPU timer at the time the interruption occurred.

Clock-Comparator Validity
Bit 47 (CC), when one, indicates that the clock comparator is not in error and that the contents of the clock-comparator save area at location 224 reflect the correct state of the clock comparator.

Programming Note
The validity bits must be used in addition to the subclass bits in order to determine the extent of the damage caused by a machine-check condition. No damage has occurred to the system when the following are true:

- The four PSW validity bits, the three register validity bits, the two timing-facility-validity bits, and the storage-logical-validity bit must all be ones.
- The damage-subclass bits 0, 1, 3, 4, and 5 must be zeros.

Machine-Check Extended Interruption Information

As part of the machine-check interruption, in some cases, extended interruption information is placed in fixed areas assigned in storage. The contents of registers associated with the CPU are placed in register-save areas. When storage error uncorrected or storage-key error uncorrected is indicated, the failing-storage address is saved.

Each of these fields has associated with it a validity bit in the machine-check-interruption code. If, for any reason, the machine cannot store the proper information in the field, the associated validity bit is set to zero.

Register-Save Areas

As part of the machine-check interruption, the current contents of the CPU registers, except for the time-of-day clock, are stored in five register-save areas assigned in storage. Each of

these areas has associated with it a validity bit in the machine-check-interruption code. If, for any reason, the machine cannot store the proper information in the field, the associated validity bit is set to zero.

The following are the five sets of registers and the locations in storage where their contents are saved during a machine-check interruption.

Locations	Registers
216-223	CPU timer
224-231	Clock comparator
352-383	Floating-point registers 0, 2, 4, 6
384-447	General registers 0-15
448-511	Control registers 0-15

The information stored for unassigned control-register positions is unpredictable.

Failing-Storage Address

When storage error uncorrected or storage-key error uncorrected is indicated in the machine-check-interruption code, the associated address, called the failing-storage address, is stored in bits 8-31 of the word at location 248. Bits 0-7 of that word are set to zeros.

The failing-storage address may be the address of any location within the page that is in error or that is associated with the storage key in error. When an error is detected in more than one location before the interruption, the failing-storage address may point to any of the failing locations.

Machine-Check Masking

All machine-check interruptions are under control of the machine-check mask, PSW bit 13. In addition, some machine-check conditions are controlled by subclass masks in control register 14.

The exigent machine-check conditions (system damage and instruction-processing damage) are controlled only by the machine-check mask, PSW bit 13. When PSW bit 13 is one, an exigent condition causes a machine-check interruption. When PSW bit 13 is zero, the occurrence of an exigent machine-check condition causes the CPU to enter the check-stop state.

The repressible machine-check conditions are controlled both by the machine-check mask, PSW bit 13, and by four subclass-mask bits in control register 14. If PSW bit 13 is one and one of the subclass-mask bits is one, the associated condition initiates a machine-check interruption. If a subclass-mask bit is zero, the associated condition does not initiate an interruption. However, when a machine-check interruption is initiated because of a condition for which the CPU is enabled, those conditions for which the CPU is not enabled may be presented along with the condition which initiates the interruption. All conditions presented are then cleared.

Control Register 14

Bits 4-7 of control register 14 are the subclass masks for repressible machine-check conditions. In addition, bit 0 is initialized to one, but it is otherwise ignored by the machine. All other bits of control register 14 are unassigned.

Programming Note

The program should avoid, whenever possible, operating with PSW bit 13, the machine-check mask, set to zero, since any exigent machine-check condition which is recognized during this situation will cause the CPU to enter the check-stop state. In particular, the program should avoid issuing I/O instructions or allowing for I/O interruptions with PSW bit 13 a zero.

Recovery-Report Mask

Bit 4 (RM) of control register 14 controls system-recovery-interruption conditions. This bit is initialized to zero.

Degradation-Report Mask

Bit 5 (DM) of control register 14 controls degradation-interruption conditions. This bit is initialized to zero.

External-Damage-Report Mask

Bit 6 (EM) of control register 14 controls timing-facility-damage, interval-timer-damage, and external-damage conditions. This bit is initialized to one.

Warning Mask

Bit 7 (WM) of control register 14 controls warning conditions. This bit is initialized to zero.

Chapter 12. Input/Output Operations

Contents

The transfer of information to or from main storage, other than to or from the central processing unit, is referred to as an input or output operation. An input/output (I/O) operation involves the use of an I/O 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 publication 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, and its mode of operation.

The formats described include provisions for functions unique to some I/O device types, such as an erase gap on a magnetic-tape unit. The way in which a device makes use of the format is defined in the System Library (SL) publication for the particular device.

All main-storage references for I/O operations are references to virtual storage. Unless indicated otherwise, "storage" means virtual storage, and "address" means virtual address. The terms "I/O address," "channel address," and "device address" are never abbreviated to "address" in this publication.

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 readers, card punches, magnetic-tape units, direct-access-storage devices (disks and drums), display units, typewriter-keyboard devices, printers, teleprocessing devices, and sensor-based equipment.

Most types of I/O devices, such as printers, card equipment, or tape devices, deal directly with external media, 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 reaches a physical recording medium outside main storage. Similarly, a transmission-control unit handles transmission of information between the data-processing system and a remote station, and its input and output are signals on a transmission line. An I/O device may be physically distinct equipment, or it may time-share equipment with other I/O devices.

An input/output device ordinarily is attached to one control unit and is accessible from one channel. Switching equipment is available to make some devices accessible to two or more channels by switching devices between control units and control units between channels. The time required for switching occurs during device-selection time and may be ignored.

Control Units

A control unit provides the logical capabilities necessary to operate and control an I/O device and adapts the characteristics of each device to the standard form of control provided by the channel.

The control unit accepts control signals from the channel, controls the timing of data transfer, and provides indications concerning the status of the device.

The I/O device attached to the control unit may be designed to perform only certain limited operations, or it may perform many different operations. A typical operation is moving 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 or the CPU. In 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 some electronic I/O devices such as the channel-to-channel adapter, the control unit does not have an identity of its own.

From the programmer's point of view, most functions performed by the control unit can be merged with those performed by the I/O device. Therefore, this publication normally 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 describing how sharing of the control unit among a number of devices affects the execution of I/O operations.

Channels

A 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.

A channel provides a means for connecting different types of I/O devices to the CPU and to storage. The channel 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 and device. Similarly, when an I/O device provides signals that should be brought to the attention of the program, the channel transforms the signals to information that can be used in the CPU.

A channel contains facilities for the control of I/O operations. During execution of an I/O operation involving data transfer, the channel assembles or disassembles data and synchronizes the transfer of data bytes with storage cycles. To accomplish this, the channel maintains and updates an address and a count that describe the destination or source of data in storage. When the channel 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 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 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 storage area.

If separate equipment is not provided, facilities of the CPU are used for controlling I/O devices. When the CPU and channels, or the CPU, channels, and control units, share common facilities, I/O operations cause interference to the CPU, varying in intensity from occasional delay of a CPU cycle to 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

An I/O operation occurs in one of two modes: burst or byte-multiplex.

In burst mode, the I/O device monopolizes the channel and stays logically connected to the channel for the transfer of a burst of information. No other device can communicate with the channel during the time a burst is transferred. The burst can consist of a few bytes, a whole block of data, a sequence of blocks with associated control and status information (the block lengths may be zero), or status information which monopolizes the channel. The facilities in a channel capable of operating in burst mode may be shared by a number of concurrently operating I/O devices.

Some channels can tolerate an absence of data transfer during a burst-mode operation, such as occurs when reading a long gap on magnetic tape, for not more than approximately 1/2 minute. Equipment malfunction may be indicated when an absence of data transfer exceeds this time.

In byte-multiplex mode, the I/O device stays logically connected to the channel only for a short interval of time. The facilities in a channel capable of operating in byte-multiplex mode may be shared by a number of concurrently operating I/O devices. In this mode, all I/O operations are split into short intervals of time during which only a segment of information is transferred. During such 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.

Operation in burst and byte-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 byte-multiplex mode is capable of initiating an operation on another device. If a channel that can operate in either mode is 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. Furthermore, the new I/O operation is initiated only after the channel has serviced all outstanding requests from devices previously placed in operation.

The distinction between a short burst of data occurring in the byte-multiplex mode and an operation in the burst mode is in the length of the

bursts of data. A channel that can operate in either mode determines its mode of operation by timeout. 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.

Ordinarily, devices with a high data-transfer rate operate with the channel in burst mode, and slower devices run in byte-multiplex mode. Some control units have a manual switch for setting the mode of operation.

Types of Channels

A system can be equipped with three types of channels: selector, byte multiplexer, and block multiplexer.

The channel facilities required for sustaining a single I/O operation are termed a *subchannel.* The subchannel consists of internal storage used for recording the addresses, count, and any status and control information associated with the I/O operation. The capability of a channel to permit multiplexing depends upon whether it has more than one subchannel.

A selector channel, which contains a minimum of facilities, 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. A selector channel cannot perform any multiplexing and therefore can be involved in only one I/O operation or chain of operations 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 backspacing tape. 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.

A byte-multiplexer channel contains multiple subchannels and can operate at any one time in either byte-multiplex or burst mode. The channel operates most efficiently when running I/O devices that are designed to operate in byte-multiplex mode. The mode of operation is determined by the I/O device, and, during data transfer, the mode can change at any time. Unless data transfer is occurring, the mode of operation has no meaning. The data transfer associated with an operation can occur partially in the byte-multiplex mode and partially in the burst mode.

A block-multiplexer channel contains multiple subchannels and can only operate in burst mode. The channel operates most efficiently when running devices that are designed to operate in burst mode. When multiplexing is not inhibited, the channel permits multiplexing between blocks, between bursts, or when command retry is performed. On most models, the burst is forced to extend over the block of data, and multiplexing is permitted either between blocks of data or when command chaining is specified. Whether or not multiplexing occurs depends on the design of the channel and I/O device and on the state of the block-multiplexing-control bit.

When the block-multiplexing-control bit, bit 0 of control register 0, is zero, multiplexing is inhibited; when it is one, multiplexing is allowed.

Whether a block-multiplexer channel executes an I/O operation with multiplexing inhibited or allowed is determined by the state of the block-multiplexing-control bit at the time the operation is initiated by START I/O or START I/O FAST RELEASE and applies to that operation until the involved subchannel becomes available.

For brevity, the term "multiplexer channel" is used hereafter when describing a function or facility that is common for both byte-multiplexer and block-multiplexer channels. Multiplexer channels vary in the number of subchannels they contain. When multiplexing, they 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 in those aspects of communication that pertain to the physical channel (for example, individual subchannels on a multiplexer channel are not distinguished as such by the TEST CHANNEL instruction or by the masks controlling I/O interruptions from the channel). When a multiplexer channel is not servicing an I/O device, it monitors its devices for data and for status information.

Subchannels on a multiplexer channel may be either *nonshared* or *shared.*

A subchannel is referred to as nonshared if it is associated with and can be used only by a single I/O device. A nonshared subchannel is used with devices that do not have any restrictions on the concurrency of channel-program operations, such as the IBM 3211 Printer Model 1 or one drive of an IBM 3330 Disk Storage.

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 some 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. I/O devices may share a control unit without necessarily sharing a subchannel. 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.

Programming Notes

A block-multiplexer channel can be made to operate as a selector channel by the appropriate setting of the block-multiplexing-control bit. However, since a block-multiplexer channel inherently can interleave the execution of multiple I/O operations and since the state of the block-multiplexing-control bit can be changed at any time, it is possible to have one or more operations that permit multiplexing and an operation that inhibits multiplexing being executed simultaneously by a channel.

Therefore, to ensure complete compatibility with selector channel operation, all operational subchannels on the block-multiplexer channel must be available or operating with multiplexing inhibited when the use of that channel as a selector channel is begun. All subsequent operations should then be initiated with the block-multiplexing-control bit inhibiting multiplexing.

I/O-System Operation

Input/output operations are initiated and controlled by information with two types of formats: instructions and channel-command words (CCWs). *Instructions* are decoded by the CPU and are part of the CPU program. *CCWs* are decoded and executed by the channels and I/O devices and initiate I/O operations, such as reading and writing. One or more CCWs arranged for sequential execution form a channel program. Both instructions and CCWs are fetched from storage and their formats are common for all types of I/O devices, although the modifier bits in the command code of a CCW may specify device-dependent operations.

The CPU program initiates I/O operations with the instruction START I/O or START I/O FAST RELEASE. These instructions identify the channel and device and cause the channel to fetch the channel-address word (CAW) from a fixed location in storage. The CAW contains the subchannel key and designates the location in storage from which the channel subsequently fetches the first CCW. The CCW specifies the command to be executed and the storage area, if any, to be used.

When the CAW has been fetched, some channels consider the execution of START I/O FAST RELEASE complete. The results of the execution of the instruction to that point are indicated by setting the condition code in the program-status word (PSW) and, in certain situations, by storing pertinent information in the channel-status word (CSW).

If the channel is not operating in burst mode and if the subchannel associated with the addressed I/O device is available, 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 the address of the selected device. The channel subsequently sends the command-code part of the CCW to the control unit, and the device responds with a status byte indicating whether it can execute the command.

At this time, the execution of START I/O and of START I/O FAST RELEASE, if not previously considered complete, is completed. The results of the attempt to initiate the execution of the command are indicated by setting the condition code in the PSW and, in certain situations, by storing pertinent information in the CSW.

If the I/O 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 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, generally a list of CCWs is used for execution of the I/O operation, each CCW designating a contiguous storage area, and the CCWs are 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 at least as far as that CCW.

The conclusion of an I/O operation normally is indicated by channel end and device end. Channel end indicates that the I/O device has received or provided all data associated with the operation and no longer needs channel facilities. Device end indicates that the I/O device has concluded execution of the operation. Device end can occur concurrently with channel end or later.

Operations that keep the control unit busy after releasing channel facilities may, in some situations, cause a third indication called control-unit end. Control-unit end may occur only concurrently with or after channel end and indicates that the control unit has become available for initiation of another operation.

Concurrent with channel end, both the channel and the I/O device can provide indications of unusual situations. Control-unit end and device end can be accompanied by error indications from the I/O device.

The indication of the conclusion of an I/O operation can be brought to the attention of the program by I/O interruptions or, when the CPU is disabled for I/O interruptions from the channel, by programmed interrogation of the I/O device. An indication that will result in an interruption or that can be observed through interrogation is called an interruption condition. In either case, a CSW is stored, which contains additional information concerning the execution of the operation. When channel end is indicated in the CSW, the CSW identifies the last CCW used and provides its residual byte count, thus indicating the extent of storage used.

Facilities are provided for the program to initiate the execution of a chain of I/O operations with a single START I/O or START I/O FAST RELEASE. 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 in the same way as the first operation. Channel end and device end are not presented to the program when chaining causes another operation to follow. However, unusual situations can cause premature termination of command chaining and generation of an interruption condition.

Activities that cause I/O-interruption conditions are asynchronous to activity in the CPU, and more than one interruption condition can exist at the same time. The channel and the CPU establish priority among the conditions so that only one condition is presented to the CPU at a time. The conditions are preserved in the I/O devices or subchannels until accepted by the CPU.

The 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 or START I/O FAST RELEASE, 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 time the CPU sets condition code 0 for the START I/O or START I/O FAST RELEASE instruction until the interruption condition caused by the signal that terminates 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 control-unit end when it becomes free.
4. The I/O device is busy from the initiation of the first operation until the interruption condition caused by the device end associated with the operation is accepted or cleared by the CPU.

An interruption condition caused by device end makes the device appear busy, but normally does not affect the state of any other part of the system. An interruption condition caused by control-unit end may block communications through the control unit to any device attached to it, and an interruption condition caused by 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 CPU model to which it is attached. 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 CCWs are fetched and on the address resolution of the first byte in each new storage area. Furthermore, since a channel shares storage with the CPU and other channels, activity in the rest of the system affects the accessibility of 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 elements in 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 differing in performance but 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 eight I/O instructions: CLEAR I/O, HALT DEVICE, HALT I/O, START I/O, START I/O FAST RELEASE, STORE CHANNEL ID, TEST CHANNEL, and TEST I/O.

The instructions TEST CHANNEL and STORE CHANNEL ID address a channel; they do not address an I/O device. The other six 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 16-bit I/O address consists of two parts: a channel address in the leftmost eight bit positions and a device address in the rightmost eight bit positions.

The channel address provides for identifying up to 256 channels. Channels are numbered 0-255. Channel 0 is a byte-multiplexer channel, and each of channels 1-255 may be a byte-multiplexer, block-multiplexer, or selector channel.

The number and type of channels and subchannels available, as well as their address assignment, depend on the system model and the particular installation.

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. An exception is some multiplexer channels that provide fewer than the maximum configuration of subchannels and hence eliminate the corresponding unassignable device addresses.

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 (for example, 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 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 the values 0000 to 1001 in the low-order bit positions of the device address, it does not recognize addresses containing 1010 to 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. 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 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.

The assignment of channel and device addresses is arbitrary, subject to the rules described and any model-dependent restrictions. 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 figure "Input/Output System States." The name of the state is followed by its abbreviation and a brief definition.

A channel, subchannel, or I/O device that is available, interruption-pending, or working is called "operational." A channel, subchannel, or I/O device that is interruption-pending, working, or not-operational is called "not available."

In a multiplexer channel, the channel and subchannel are easily distinguishable and, if the channel is operational, any combination of channel and subchannel states is 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 selector and multiplexer channels to be covered uniformly by a single description.

The device referred to in the figure "Input/Output-System States" 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 "not available" shared control unit imposes its

Name	Abbreviation	Definition
Channel		
Available	A	None of the following states
Interruption pending	I	Interruption condition immediately available from channel
Working	W	Channel operating in burst mode
Not operational	N	Channel not operational
Subchannel		
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		
Available	A	None of the following states
Interruption pending	I	Interruption condition in device
Working	W	Device executing an operation
Not operational	N	Device not operational

Input/Output-System States

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 by the state of the channel or subchannel, the components further removed are not interrogated. Thus, 10 composite states may be distinguished as conditions for the execution of I/O instructions. Each composite state is identified by three letters. The first letter specifies the state of the channel, the second letter specifies the state of the subchannel, and the third letter specifies the state of the device. Each letter may be A, I, W, or N, denoting the state of the component. The letter X 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.

Because of internal activity, some block-multiplexer channels may at times appear to be working even though they are not engaged in the execution of a previously initiated operation and do not contain any interruption condition. This will result in a WXX state instead of the AAA state.

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 an interruption condition. These situations are possible:

1. The device is executing an operation, such as rewinding magnetic tape or seeking on a disk file, after signaling channel end.
2. The control unit associated with the device is executing an operation, such as backspacing file on a magnetic-tape unit, after signaling channel end.
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 a channel-end condition associated with a terminated operation.

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 disconnected from the system. 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. (See also the section "Input/Output Device Addressing" in this chapter.)

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, 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 the section "Unit Check" in this chapter.)

Interruption Pending in Subchannel (AIX): The addressed channel is available. An interruption condition is pending in the addressed subchannel. The subchannel is able to provide information for a CSW. The interruption information indicates status associated with the addressed device or 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 interruption condition, 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 and has not yet received channel end for the last operation. The state of the addressed device is not significant, except when HALT I/O or HALT DEVICE is issued. During the execution of HALT I/O and HALT DEVICE, the state of the device may be interrogated and will then be indicated in either the CSW or the condition code.

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 multiplexer 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 in which the channel contains an 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, with the exception of STORE CHANNEL ID, consider the channel available when it contains an interruption condition. A channel with an interruption condition may be considered to be working by the instruction STORE CHANNEL ID. When the channel assigns priority for interruptions among devices, the interruption condition is preserved in the I/O device or subchannel. (See the section "Interruption Conditions" in this chapter.)

Channel Working (WXX): The addressed channel is operating in the burst mode. In the multiplexer channel, a burst of bytes is currently being handled. In 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 signaled. The states of the addressed device and, in the multiplexer channel, of the subchannel are not significant. In addition, because of internal activity, some block-multiplexer channels may at times appear to be working even though they are not operating in burst mode. Depending on the model and the channel type, TEST I/O and HALT DEVICE may consider the channel to be available when the channel is working with a device other than the addressed device.

Channel Not Operational (NXX): The addressed channel is not operational. A channel is not operational when it is not provided in the system, when power is off in the channel, or when it is not configured to the CPU. 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: an I/O system reset and an I/O selective reset. The response of each type of I/O device to the two kinds of reset is specified in the SL publication for the device.

I/O-System Reset

I/O-system reset is performed in the channel and on the associated I/O interface when the CPU performs a program reset, initial-program reset, clear reset, or power-on reset, and when a power-on sequence is performed by the channel.

I/O-system reset causes the channel to conclude operations on all subchannels. Status information and all interruption conditions in all subchannels are reset, and all operational subchannels are placed in the available state. The channel signals system reset to all I/O devices attached to it.

I/O Selective Reset

The I/O selective reset is performed by some channels when they detect certain equipment malfunctions.

I/O selective reset causes the channel to signal selective reset to the device that is connected to the channel at the time the malfunction is detected. No subchannels are reset.

Effect of Reset on a Working Device

With either type of reset, if the device is currently communicating with a channel, the device immediately disconnects from the channel. Data transfer and any operation using the facilities of the control unit are immediately concluded, and the I/O device is not necessarily positioned at the beginning of a block. Mechanical motion not involving the use of the control unit, such as rewinding magnetic tape or positioning a disk-access mechanism, proceeds to the normal stopping point, if possible. The device 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, but an interruption condition may be generated upon completing any mechanical operation.

Reset Upon Malfunction

When a malfunction occurs and the program is alerted by an I/O interruption, or when a malfunction occurs during the execution of an I/O instruction and the program is alerted by the setting

of a condition code, then an I/O selective reset may have been performed. A CSW is stored identifying the cause of the malfunction.

The device addressed by the I/O instruction is not necessarily the device that is reset.

When a malfunction occurs and the program is alerted by a machine-check interruption, then an I/O selective reset may have been performed. This may or may not be accompanied by an I/O interruption. When no I/O interruption occurs, a CSW is not stored and a device is not identified.

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 the PSW. The condition code is set at the time the execution of the instruction is concluded, that is, the time the CPU is released to proceed with the next instruction. The condition code ordinarily indicates whether or not the function specified by the instruction has been performed and, if not, the reason for the rejection. In the case of START I/O FAST RELEASE executed independent of the device, a condition code 0 may be set that is later superseded by a deferred condition code stored in the CSW.

The figure "Condition-Code Settings for I/O States and Instructions" lists the I/O-system states and the corresponding condition codes for each I/O instruction. The I/O-system states and associated abbreviations were defined in the section "States of the Input/Output System" earlier in this chapter. The digits in the figure represent the decimal value of the code.

I/O-System States	I/O State	Condition-Code Settings						
		SIO SIOF	TIO	CLRIO	HIO	HDV	TCH	STIDC
Available	AAA	0,1*@	0	0	1*	1*	0	0
Interruption pending in device	AAI	1*@	1*	0	1*	1*	0	0
Device working	AAW	1*@	1*	0	1*	1*	0	0
Device not operational	AAN	3@	3	0	3	3	0	0
Interruption pending in subchannel	AIX							
For the addressed device		2	1*	1*	0	0	0	0
For another device		2	2	0	0	0	0	0
Subchannel working	AWX							
With the addressed device		2	2	1*	1*#	1*#	0	0
With another device		2	2	0	1*#	0	0	0
Subchannel not operational	ANX	3	3	3	3	3	0	0
Interruption pending in channel	IXX	See Note					1	##
Channel working	WXX							
With the addressed device		2	2	***	2	+	2	##
With another device		2	2•	**	2	≠	2	##
Internal activity		2	2•	**	2	≠	2	##
Channel not operational	NXX	3	3	3	3	3	3	3

Explanation:

* Whenever condition code 1 is set, the CSW or its status portion is stored at location 64 during execution of the instruction.

** When CLEAR I/O encounters the WXX state, either condition code 2 is set, or the channel is treated as available and the condition code is set according to the state of the subchannel. When the channel is treated as available, the condition codes for the WXX states are the same as for the AXX states.

*** Condition code 1 (with the CSW stored) or 2 may be set, depending on the channel.

≠ The condition code depends on the state of the subchannel, the channel type, and the system model. If the subchannel is not operational, condition code 2 or 3 is set. If the subchannel is available or working with the addressed device, condition code 2 is set. Otherwise, condition code 0 or 2 is set.

When a "device not operational" response is received in selecting the addressed device, condition code 3 is set.

@ START I/O FAST RELEASE may cause the same condition code to be set as for START I/O or may cause condition code 0 to be set.

\+ If the channel ascertains that the device received the signal to terminate, condition code 1 is set and the CSW stored. Otherwise, condition code 2 is set.

When the channel is unable to store the channel ID because of the working or interruption-pending state, condition code 2 is set. If the working or interruption-pending state does not preclude storing the channel ID, condition code 0 is set.

• If the subchannel is interruption-pending for the addressed device, condition code 1 may be set depending on the channel type.

Note: For the purpose of executing START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT DEVICE, and HALT I/O, a channel containing an interruption condition appears the same as an available channel, and the condition-code setting depends on the states of the subchannel and device. The condition codes for the IXX states are the same as for the AXX states, where the Xs represent the states of the subchannel and the device. As an example, the condition code for the IAW state is the same as for AAW.

Condition-Code Settings for I/O States and Instructions

The available state results in condition code 0 only when no errors are detected during the execution of the I/O instruction.

When a subchannel on a multiplexer channel contains an interruption condition (state AIX), the I/O device associated with the concluded operation normally is in the interruption-pending state. When the channel detects during the 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 or HALT DEVICE is addressed to a subchannel in the working state (state AWX), but the device is not operational.

Error conditions, including all equipment or programming errors detected by the channel or the I/O device during execution of the I/O instruction, generally cause the CSW to be stored. However, when the nature of the error causes a machine-check interruption, but no I/O interruption, to occur, the CSW is not stored. Three types of errors can occur:

Channel-Equipment Error: The channel can detect the following equipment errors during execution of START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, and HALT DEVICE:

1. The channel received an address from the device during initial selection that either had a parity error or was 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 during initial selection had a parity error.
3. A signal from the I/O device occurred at an invalid time or had invalid duration.
4. The channel detected an error in its control equipment. (This is also true for STORE CHANNEL ID and TEST CHANNEL.)

The channel may perform an I/O selective reset or generate a halt signal, 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 or START I/O FAST RELEASE. All of the errors are indicated during START I/O, and during START I/O FAST RELEASE when it is executed as START I/O, by the condition-code setting and by the status portion of the CSW. When the SIOF function is performed, the first two errors are indicated as for START I/O, and the remaining errors are indicated in a subsequent interruption.

1. Invalid CCW-address specification in CAW
2. Invalid CAW format
3. Storage location of first CCW not provided
4. First-CCW location in a disconnected page
5. First-CCW location protected against fetching
6. First CCW specifies transfer in channel
7. Invalid command code in first CCW
8. Invalid count in first CCW
9. Invalid format for first CCW

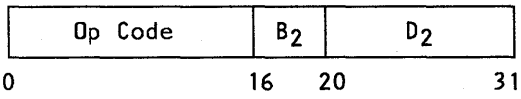
The CSW indicates program check, except for items 4 and 5, for which protection check is indicated.

Device Error: Programming or equipment errors detected by the device during the execution of START I/O, or START I/O FAST RELEASE are indicated by unit check or unit exception in the CSW.

The causes of unit check and unit exception for each type of I/O device are detailed in the SL publication for the device.

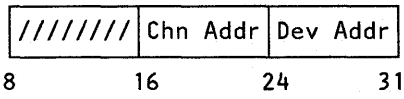
Instruction Formats

All I/O instructions use the following S format:



Except for STORE CHANNEL ID, bit positions 8-14 of these instructions are ignored. Bit position 15 is ignored by the instruction TEST CHANNEL but is decoded as part of the operation code for CLEAR I/O, HALT DEVICE, HALT I/O, START I/O, START I/O FAST RELEASE, and TEST I/O.

The second-operand address specified by the B_2 and D_2 fields is not used to designate data but instead is used to identify the channel and I/O device. Address computation follows the rules of address arithmetic. The address has the following format:



Bit positions 16-31 contain the 16-bit I/O address. Bit positions 8-15 are ignored.

Instructions

All I/O instructions cause a serialization function to be performed. See the section "Serialization" in Chapter 5, "Program Execution."

The names, mnemonics, format, and operation codes of the I/O instructions are listed in the figure "Input/Output Instructions." The figure also indicates that all I/O instructions cause a program interruption when they are encountered in the problem state, that all I/O instructions set the condition code, and that all I/O instructions are in the S instruction format.

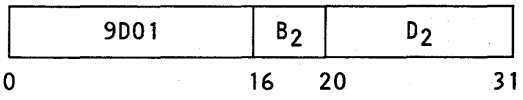
Note: *In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. In the case of START I/O, for example, SIO is the mnemonic and $D_2(B_2)$ the operand designation.*

Programming Note

The instructions CLEAR I/O, HALT DEVICE, HALT I/O, START I/O, START I/O FAST RELEASE, STORE CHANNEL ID, and TEST I/O 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, the CPU must be disabled for all I/O interruptions before CLEAR I/O, HALT DEVICE, HALT I/O, START I/O, START I/O FAST RELEASE, STORE CHANNEL ID, and TEST I/O is issued and must remain disabled until the information in the CSW provided by the instruction has been acted upon or stored elsewhere for later use.

CLEAR I/O

CLRIO $D_2(B_2)$ [S]



Either a TIO or CLRIO function is performed, depending on the block-multiplexing control, bit 0 of control register 0. The TIO function is performed when the block-multiplexing-control bit is zero.

The TIO function is described in the definition of the instruction TEST I/O.

Bits 8-14 of the instruction are ignored. Bit positions 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies.

The CLRIO function causes the current operation with the addressed device to be discontinued and the state of the operation at the time of the discontinuation to be indicated in the stored CSW.

When the subchannel is available, interruption-pending with another device, or working with another device, no channel action is taken, and condition code 0 is set. Channels not capable of determining subchannel states while in the working state may instead set condition code 2.

Name	Mnemonic	Characteristics				Op Code
CLEAR I/O	CLRIO	S	C	P	$	9D01*
HALT DEVICE	HDV	S	C	P	$	9E01*
HALT I/O	HIO	S	C	P	$	9E00*
START I/O	SIO	S	C	P	$	9C00*
START I/O FAST RELEASE	SIOF	S	C	P	$	9C01*
STORE CHANNEL ID	STIDC	S	C	P	$	B203
TEST CHANNEL	TCH	S	C	P	$	9F00≠
TEST I/O	TIO	S	C	P	$	9D00*

Explanation:

- C Condition code is set.
- P Privileged-operation exception.
- S S instruction format.
- * Bits 8-14 of the operation code are ignored.
- ≠ Bits 8-15 of the operation code are ignored.
- $ Causes serialization.

Summary of Input/Output Instructions

When the subchannel is either working with the addressed device or interruption-pending with the addressed device, the CLRIO function causes condition code 1 to be set and causes the channel to discontinue the operation with the addressed device by storing the status of the operation in the CSW and making the subchannel available. When the channel is working with the addressed device, the device is signaled to terminate the current operation. Some channels may, instead, indicate busy and cause no channel action.

When any of the following conditions occurs, the CLRIO function causes the CSW at location 64 to be stored. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

1. The channel is available or interruption-pending, and the subchannel contains an interruption condition for the addressed device or is working with the addressed device. The subchannel-key, command-address, and count fields describe the state of the operation at the time of the execution of the instruction.
2. The channel is working with the addressed device. The subchannel-key, command-address, and count fields describe the state of the operation at the time the instruction is executed. (Some channels alternatively indicate busy under this condition.)
3. The channel is working with a device other than the one addressed, and the subchannel contains an interruption-pending condition for the addressed device or is working with the addressed device. The subchannel-key, command-address, and count fields describe the state of the operation at the time CLEAR I/O is executed. (Some channels alternatively indicate busy under these conditions.)
4. The channel detected an equipment error during the execution of the instruction. The CSW identifies the error condition. The states of the channel and the I/O operations in progress are unpredictable. The limited channel logout, if stored, indicates a sequence code of 000.

When CLEAR I/O cannot be executed because of a pending logout that affects the operational capability of the channel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending and channel-control-check bits, which are set to ones. No channel logout is associated with this status.

Program Exceptions:
Privileged Operation

Resulting Condition Code:

0 No operation in progress for the addressed device
1 CSW stored
2 Channel busy
3 Not operational

The condition code set when CLEAR I/O causes the CLRIO function to be performed is shown for all possible states of the I/O system in the figure "Condition Codes Set by CLEAR I/O." The condition code set when CLEAR I/O causes the TIO function to be performed is shown for all possible state of the I/O system in the figure "Condition Codes Set by TEST I/O" in the definition of the instruction TEST I/O. See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

Channel	A						I						W≠						W#	N
																			++	3
Subchannel	A	I≠	I#	W≠	W#	N	A	I≠	I#	W≠	W#	N	A	I≠	I#	W≠	W#	N		
	0	0	1*	0	1*	3	0	0	1*	0	1*	3	+	+	++	+	++	+++		

A Available
I Interruption pending
I≠ = Interruption pending for a device other than the one addressed
I# = Interruption pending for the addressed device
W Working
W≠ = Working with a device other than the one addressed
W# = Working with the addressed device
N Not operational
* CSW stored

\+ In the W≠AX, W≠I≠X, and W≠W≠X states, a condition code 0 or 2 may be set, depending on the channel.

++ In the W≠I#X, W≠W#X, and W#XX states, a condition code 1 (with the CSW stored) or 2 may be set, depending on the channel.

+++ In the W≠NX state, a condition code 2 or 3 may be set, depending on the channel.

Note: Underscored codes pertain to situations that can occur only on the multiplexer channel.

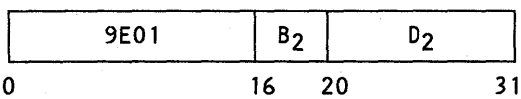
Condition Codes Set by CLEAR I/O

Programming Notes

1. Since some channels cause a condition code 2 to be set when the instruction is received and the channel is working, it may be useful to issue a halt instruction and then CLEAR I/O to the desired address. Using HALT DEVICE will ensure that condition code 2 is received on the CLEAR I/O only when the channel is working with a device other than the one addressed. Using HALT I/O will ensure that the current working state, if any, is terminated without regard for the address.
2. Because of the inability of CLEAR I/O to terminate operations on some channels when in the working state, the instruction is not a suitable substitute for HALT I/O or HALT DEVICE.
3. The combination of HALT DEVICE followed by CLEAR I/O can be used to clear out all activity on a channel by executing the two instructions for all device addresses on the channel.

HALT DEVICE

HDV D_2 (B_2) [S]



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. Bits 8-14 of the instruction are ignored.

Bits 16-31 of the second-operand address identify the channel, the subchannel, and the I/O device to which the instruction applies.

When the channel is either available or interruption-pending with the subchannel 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 signal termination of the device operation the next time the device requests or offers a byte of data. If chaining is indicated for the I/O operation using the subchannel, it is suppressed. If the subchannel is available, the subchannel is not affected.

When the channel is either available or interruption-pending with the subchannel either working with a device other than the one addressed or interruption-pending, no action is taken.

When the channel is working in burst mode with the addressed device, data transfer for the operation is immediately terminated, and the device immediately disconnects from the channel. If

chaining is indicated for the I/O operation using the subchannel, it is suppressed.

When the channel is working in burst mode with a device other than the one addressed, and the subchannel is available, interruption-pending, or working with a device other than the one addressed, no action is taken. If the subchannel is working with the addressed device, the subchannel signals termination of the device operation the next time the device requests or offers a byte of data, if any. If chaining is indicated for the I/O operation using the subchannel, it 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 interruption-pending, 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 status information 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, take place as for a selector channel or may allow the subchannel to remain in the working state until the device provides ending status.

When any of the three situations numbered below occurs, HALT DEVICE causes the 16-bit unit-status 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 the I/O operation, at the addressed subchannel, that is terminated. The extent of data transfer and the status at the termination of the operation at the subchannel are provided in the CSW associated with the interruption condition caused by the termination. The three situations 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 ones in the busy and status-modifier bit positions. 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 type of malfunction. The state of the channel and the progress of the I/O operation are unpredictable.

If HALT DEVICE cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

When HALT DEVICE causes data transfer to be terminated, the control unit associated with the operation remains not available until the data-handling portion of the operation in the control unit is concluded. Conclusion of this 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 on that channel until the channel-end condition is accepted by the CPU. The I/O device executing the terminated operation remains in the working state until the end 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.

When HALT DEVICE is issued at a time 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, 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, the device interprets the signal as one to terminate the operation. Pending attention or device-end conditions at the device are not reset.

Program Exceptions:
Privileged Operation

Resulting Condition Code:

0 Subchannel busy with another device or interruption pending
1 CSW stored
2 Channel working
3 Not operational

The condition code set by HALT DEVICE for all possible states of the I/O system is shown in the figure "Condition Codes Set by HALT DEVICE." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

Programming Note

The execution of HALT DEVICE always causes data transfer between the addressed device and the channel to be terminated. The condition code and the CSW (when stored) indicate whether the control unit was signaled to terminate its operation during the execution of the instruction. If the control unit was not signaled to terminate its operation, the condition code and the CSW (when stored) imply the situations 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 channel is not working. The end of the working state 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 in order to allow HALT DEVICE to signal the control unit to terminate.

Channel	A										I										W≠				W#	N
																									@	3
Subchannel	A				I	W≠	W#			N	A				I	W≠	W#			N	A	I	W≠	W#	N	
					0	0				3					0	0				3	2	+	+	2	•	
Control Unit – Device	A	I	W	N			A	I	W	N	A	I	W	N			A	I	W	N							
	1*	1*	1*	3			1*	1*	1*	3	1*	1*	1*	3			1*	1*	1*	3							

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

@ In the W#XX state, either condition code 1 (with CSW stored) or condition code 2 may be set, depending on the channel. However, condition code 1 (with CSW stored) can be set only if the control unit has received the signal to terminate or if control-unit-busy status is received by the channel.

\+ In the W≠IX and W≠W≠X states, either condition code 0 or 2 may be set.

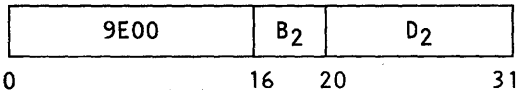
• In the W≠NX state, either condition code 2 or 3 may be set, depending on the model and the channel type.

Note: Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by HALT DEVICE

HALT I/O

HIO $D_2(B_2)$ [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. Bits 8-14 of the instruction are ignored.

Bits 16-31 of the second-operand address 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 interruption-pending, 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 byte-multiplexer 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 byte-multiplexer channel to be terminated, the subchannel associated with the burst operation remains in the working state until the device signals channel end, whereupon the subchannel enters the interruption-pending state. The termination of a burst operation by HALT I/O on a block-multiplexer channel may, depending on the model and the type of subchannel, take place as for a selector channel or may allow the subchannel to remain in the working state until the device provides ending status.

On the byte-multiplexer channel operating in the byte-multiplex mode, the device is selected and the instruction 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 is interruption-pending, with the channel available or interruption-pending, HALT I/O does not cause any action.

When any of the following conditions occurs, HALT I/O causes the status portion, bits 32-47, 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 I/O pertains only to the execution of HALT I/O and does not describe the I/O operation, at the addressed subchannel, that is terminated. The extent of data transfer, and the status at the 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 was 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 had 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, will signal termination the next time the device identifies itself. The CSW unit-status field contains ones in the busy and status-modifier bit positions. 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 cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

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 the data-transfer portion of the operation is signaled by the 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 signal is accepted by the CPU. The I/O device executing the terminated operation remains in the working state until the end 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 SL publication for the device. The HALT I/O 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, 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. Attention or device-end signals at the device are not reset.

Program Exceptions:
Privileged Operation

Resulting Condition Code:
0 Interruption pending in subchannel
1 CSW stored
2 Burst operation terminated
3 Not operational

The condition code set by HALT I/O for all possible states of the I/O system is shown in the figure "Condition Codes Set by HALT I/O." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

Programming Note
The instruction HALT I/O provides the program with a means of terminating an I/O operation before all data specified in the operation has 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 byte-multiplexer channel, HALT I/O provides a means of controlling real-time operations and permits the program to terminate data transmission on a communication line.

Channel	A								I								W	N
Subchannel	A				I	W	N		A				I	W	N		2	3
Control Unit – Device	A	I	W	N	0	1*#	3		A	I	W	N	0	1*#	3			
	1*	1*	1*	3					1*	1*	1*	3						

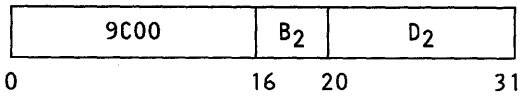
A Available
I Interruption pending
W Working
N Not operational
* CSW stored
# When a device-not-operational response is received in selecting the addressed device, a condition code 3 is set.

Note: Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by HALT I/O

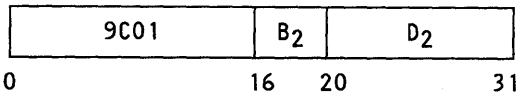
START I/O

SIO $D_2(B_2)$ [S]



START I/O FAST RELEASE

SIOF $D(B_2)$ [S]



A write, read, read backward, control, or sense operation is initiated with the addressed I/O device and subchannel. Bits 8-14 of the instruction are ignored.

Either an SIO or SIOF function is performed, depending on the instruction, the channel, and the block-multiplexing control, bit 0 of control register 0. The instruction START I/O always causes the SIO function to be performed, as does START I/O FAST RELEASE when the block-multiplexing-control bit is zero. When the bit is one, START I/O FAST RELEASE may, depending on the channel, cause either the SIO or the SIOF function to be performed.

Bits 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies. The CAW, at location 72, contains the subchannel key and the address of the first CCW. This CCW specifies the operation to be performed, the storage area to be used, and the action to be taken when the operation is completed.

For the SIO function, the I/O operation is initiated if the addressed I/O device and subchannel are available, the channel is available or interruption-pending, and errors or exceptional situations 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 situations during execution of the instruction.

For the SIOF function, the I/O operation is initiated if the subchannel is available, the channel is available or interruption-pending, and errors or exceptional situations have not been detected. The I/O operation is not initiated when the subchannel and channel are in any other state or when the channel or device detects any error or exceptional situation during execution of the instruction. The device state or device-detected errors are not relevant during instruction execution but are indicated in a CSW stored during a subsequent interruption.

When the channel is available or interruption-pending, and the subchannel is available before the execution of the instruction, the following situations cause a CSW to be stored. How the CSW is stored depends on whether an SIO or SIOF function is performed. The SIO function causes the status portion of the CSW 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. When the SIOF function is performed, the first situation causes the same action as for the SIO function; also, the control-unit and device state may be tested, and so situation 5 may cause the same action as for the SIO function, or the situation may be indicated in a subsequent I/O interruption during which the entire CSW will be stored. The remaining situations for the SIOF function will be indicated in a subsequent interruption, during which the entire CSW will be stored.

1. The channel detects a programming error in the contents of the CAW or detects an equipment error during execution of the instruction. The CSW identifies the error. If selection of the device occurred prior to detection of the error or if the error condition was detected during the selection of the device, the device status is indicated in the CSW.
2. The channel detects a programming error associated with the first CCW or, for the SIOF function, the channel detects an equipment error after completion of the instruction. The CSW identifies the error. If selection of the device occurred prior to detection of the error, or if the error condition was detected during the selection of the device, the device status is indicated in the CSW.
3. An immediate operation was executed, and either (1) no command chaining is specified and no command retry occurs, or (2) chaining is suppressed because of unusual situations detected during the operation. In the CSW, the channel-end bit is one, the busy bit is zero, and other status may be indicated. The PCI bit is one if PCI was specified in the first CCW. The I/O operation is initiated, but no information

has been transferred to or from the storage area designated by the CCW. No interruption conditions are generated at the subchannel, and the subchannel is available for a new I/O operation. If device end is not indicated, the device remains busy, and a subsequent device-end condition is generated.

4. The I/O device is interruption-pending, or the control unit is interruption-pending for the addressed device. The CSW unit-status field contains one in the busy-bit position, identifies the interruption condition, and may contain other bits provided by the device or control unit. The interruption condition is cleared. The I/O operation is not initiated. The channel-status field indicates any errors detected by the channel, and the PCI bit is one if PCI was specified in the first CCW.
5. The I/O device or the control unit is executing a previously initiated operation, or the control unit is interruption-pending for a device other than the one addressed. The CSW unit-status field contains one in the busy-bit position or, if the control unit is busy, the busy and status-modifier bits are ones. The I/O operation is not initiated. The channel-status field indicates any errors detected by the channel, and the PCI bit is one if specified in the first CCW.
6. The I/O device or control unit detected an equipment or programming error during the initiation, or the addressed device is not ready. The CSW identifies the error. The channel-end and busy bits are zeros, unless the device was busy, in which case the busy bit, as well as any bits causing interruption conditions, are ones. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation is not initiated. No interruption conditions are generated at the I/O device or subchannel. The PCI bit in the CSW is one if PCI was specified in the first CCW.

When the SIO or SIOF function cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

When the SIOF function causes condition code 0 to be set and subsequently a situation is encountered which would have caused a condition code 1 to be set had the function been SIO, a deferred-condition-code-1 I/O-interruption condition is generated. When the SIOF function causes condition code 0 to be set and, subsequently, it is determined that the device is not operational, a deferred-condition-code-3 I/O-interruption condition is generated. In both of the above cases, in the resulting I/O interruption, a full CSW is stored, and the deferred condition code appears in the CSW.

On the byte-multiplexer channel, both the SIO and SIOF functions cause 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.

Program Exceptions:
Privileged Operation

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

The condition code set by START I/O and START I/O FAST RELEASE for all possible states of the I/O system is shown in the figure "Condition Codes Set by START I/O and START I/O FAST RELEASE." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

Channel	A								I								W	N
																	2	3
Subchannel	A				I	W	N		A				I	W	N			
					2	2	3						2	2	3			
Control Unit – Device	A	I	W	N					A	I	W	N						
	≠	1*@	1*@	3@					≠	1*@	1*@	3@						

A Available
I Interruption pending
W Working
N Not operational
* CSW stored
≠
- When a nonimmediate I/O operation has been initiated, and the channel is proceeding with its execution, condition code 0 is set.
- When an immediate operation has been initiated, and no command chaining or command retry is taking place, or the device is not ready, or an error has been detected by the control unit or device, for the SIO function condition code 1 is set, and the CSW is stored. Under the same circumstances, for the SIOF function, condition code 0 is set, and a deferred-condition-code-1 I/O-interruption condition is generated.

@ The SIOF function may cause condition code 0 to be set, in which case the other condition code shown will be specified as a deferred condition code.

Note: Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

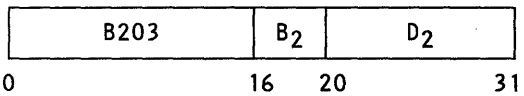
Condition Codes Set by START I/O and START I/O FAST RELEASE

Programming Notes

1. The instruction START I/O FAST RELEASE has the advantage over START I/O that the CPU can be released after the CAW is fetched, rather than after completion of the lengthy device-selection procedure. Thus, the CPU is freed for other activity earlier. A disadvantage, however, is that if a deferred condition code is presented, the resultant CPU execution time may be greater than that required in executing START I/O.
2. When the channel detects a programming error during execution of the SIO function, when the addressed device contains an interruption condition, and when the channel and subchannel are available, the instruction 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.
3. Two major differences exist between the SIO and SIOF functions:
   a. Unchained immediate commands on certain channels (that is, those which execute SIOF independent of the device) result in a condition code 0 for the SIOF function, whereas condition code 1 is set for the SIO function. See also Programming Note 2 in the section "Command Retry" of this chapter.
   b. Condition code 0 is set by these certain channels for the SIOF function, even though the addressed device is not available or the command is rejected by the device. The device information will be supplied by means of an interruption.

STORE CHANNEL ID

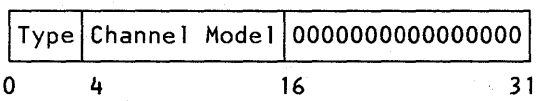
STIDC $D_2(B_2)$ [S]



Information identifying the designated channel is stored in the four-byte field at storage location 168.

Bits 16-23 of the second-operand address identify the channel to which the instruction applies. Bit positions 24-31 of the address are ignored.

The format of the information stored at location 168 is:



Bits 0-3 specify the channel type. When a channel can operate as more than one type, the code stored identifies the channel type at the time the instruction is executed. The following codes are assigned:

0000 Selector
0001 Byte multiplexer
0010 Block multiplexer

A block-multiplexer channel operates as a selector channel if the most recently initiated yet uncompleted I/O operation in the channel had block multiplexing inhibited at the time the I/O operation was initiated.

Bits 4-15 identify the channel model. When the channel model is implied by the channel type and the CPU model, zeros are stored in the field.

Bits 16-31 are set to zeros.

When the channel detects an equipment malfunction during the execution of STORE CHANNEL ID, the channel causes the status portion, bits 32-47, of the CSW to be replaced by a new set of status bits. With the exception of the channel-control-check bit (bit 45), which is stored as a one, all bits in the status field are stored as zeros. The contents of the other fields of the CSW are not changed.

When STORE CHANNEL ID cannot be executed because of a pending logout which affects the operational capability of the channel, a full CSW is stored. The fields in the CSW are all set to zero, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

Program Exceptions:
Privileged Operation

Resulting Condition Code:
0 Channel ID correctly stored
1 CSW stored
2 Channel activity prohibited storing ID
3 Not operational

The condition code set by STORE CHANNEL ID for all possible states of the I/O system is shown in the figure "Condition Codes Set by STORE CHANNEL ID." See "States of the Input/Output System" for a detailed definition of the A, I, W, and N states.

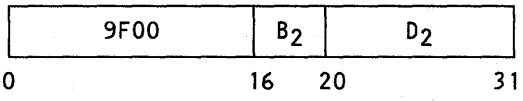
Channel	A	I	W	N
	0	•	•	3

A Available
I Interruption pending
W Working
N Not operational
• When the channel is unable to store the channel ID because of its working state or because it contains a pending-interruption condition, condition code 2 is set. If the working or interruption-pending state does not preclude the storing of the channel ID, condition code 0 is set.

Condition Codes Set by STORE CHANNEL ID

TEST CHANNEL

TCH $D_2(B_2)$ [S]



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. Bits 8-14 of the instruction are ignored.

Bits 16-23 of the second-operand address 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 interruption-pending, or is not operational. When the channel is operating in the burst mode and contains an interruption condition, the condition code is set as for operation in the burst mode. When none of these situations exist, the available

state is indicated. No device is selected, and, on the multiplexer channel, the subchannels are not interrogated.

Program Exceptions:
Privileged Operation

Resulting Condition Code:
0 Channel available
1 Interruption or logout condition in channel
2 Channel operating in burst mode
3 Channel not operational

The condition code set by TEST CHANNEL for all possible states of the addressed channel is shown in the figure "Condition Codes Set by TEST CHANNEL." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

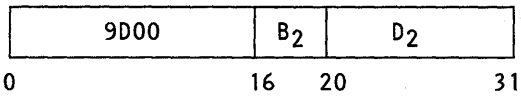
Channel	A	I	W	N
	0	1	2	3

A Available
I Interruption pending
W Working
N Not operational

Condition Codes Set by TEST CHANNEL

TEST I/O

TIO $D_2(B_2)$ [S]



The state of the addressed channel, subchannel, and device is indicated by setting the condition code in the PSW and, in certain situations, by storing the CSW. Interruption conditions may be cleared. Bits 8-14 of the instruction are ignored.

Bits 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies.

The TIO function is performed by the instruction TEST I/O and, under certain circumstances, by CLEAR I/O.

When the channel is operating in burst mode and the addressed subchannel contains an interruption condition, the TIO function causes condition code 1 or 2 to be set, depending on the model and channel type. If condition code 1 is set, the CSW is stored at location 64 to identify the interruption condition, and the interruption condition is cleared.

When the situation described in the following paragraph occurs with the channel either available or interruption-pending or, on some channels, working, the TIO function causes the CSW to be stored. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

The subchannel contains an interruption condition due to a terminated operation at the addressed device. The CSW identifies the interruption condition, and the interruption condition is cleared. The subchannel key, CCW address, and count fields contain the final values for the I/O operation, and the status field may include 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 channel is working and has not yet accepted the interruption condition from the device.

When any of the following situations occurs with the channel either available or interruption-pending, the TIO function causes the CSW to be stored. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

1. The subchannel is available, and the I/O device contains an interruption condition or the control unit contains control-unit end for the addressed device. 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 zero. The other fields of the CSW contain zeros unless an equipment error is detected.
2. The subchannel is available, and the I/O device or the control unit is executing a previously initiated operation or the control unit has an interruption condition associated with a device other than the one addressed. The CSW unit-status field contains one in the busy-bit position or, if the control unit is busy, the busy and status-modifier bits are ones. Other fields of the CSW contain zeros unless an equipment error is detected.
3. 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 not ready and does not have any interruption condition. The CSW identifies the error. 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 cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout is associated with this status.

When the TIO function is used to clear an interruption condition from the subchannel and the channel has not yet accepted the condition from the device, the function 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. Some magnetic-tape control units, for example, are in such a state when they have provided channel end and are 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 has the busy and status-modifier bits set to ones, 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, 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 has the status-modifier bit set to one, 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 the TIO function 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 byte-multiplexer channel, the TIO function causes the addressed device to be selected only after the channel has serviced all outstanding requests for data transfer for previously initiated operations.

Program Exceptions:
Privileged Operation

Resulting Condition Code:
0 Available
1 CSW stored
2 Channel or subchannel busy
3 Not operational

The condition code set by the TIO function for all possible states of the I/O system is shown in the figure "Condition Codes Set by TEST I/O." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

Channel	A								I								W≠				W#	N
																					2	3
Subchannel	A				I≠	I#	W	N	A				I≠	I#	W	N	A	I≠	I#	W	N	
					2	1*	2	3					2	1*	2	3	2	2	@	2	2	
Control Unit – Device	A	I	W	N					A	I	W	N										
	0	1*	1*	3					0	1*	1*	3										

A Available
I Interruption pending
I≠ = Interruption pending for a device other than the one addressed
I# = Interruption pending for the addressed device
W Working
W≠ = Working with a device other than the one addressed
W# = Working with the addressed device
N Not operational
* CSW stored
@ In the W≠I#X state, either condition code 1 may be set with the CSW stored, or condition code 2 may be set, depending on the channel and the activity in the channel.

Note: Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by TEST I/O

Programming Notes

1. Disabling the CPU for I/O interruptions provides the program with a means of controlling the priority of I/O interruptions selectively by channels. The priority of devices attached on a channel cannot be controlled by the program. The instruction TEST I/O in some cases permits the program to clear interruption conditions selectively by I/O device.
2. When a CSW is stored by the TIO function, the interface-control-check and channel-control-check indications may be due to an interruption condition already existing in the channel or may be due to an interruption condition created by the TIO function. Similarly, the unit-check bit set to one with the channel-end, control-unit-end, or device-end bits set to zeros may be due to a situation created by the preceding operation, the I/O device being not ready, or an equipment error detected during the execution of TEST I/O. The instruction TEST I/O cannot be used to clear an interruption condition due to the PCI flag while the subchannel is working.
3. The use of a TEST I/O loop on a multiplexer channel to retrieve ending status for a channel program should, in general, be avoided. TEST I/O loops may be used to return ending status to a sense command when that command was initiated by a START I/O that received condition code 0. TEST I/O loops under other conditions may result in hang conditions.

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 situations detected at this time cause a program 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 subchannel 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 is signaled.

Blocking of Data

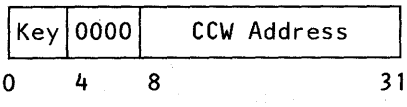
Data recorded by an I/O device may be divided into blocks. The length of a block depends on the device; for example, a block can be a card, a line of printing, or the information recorded between two consecutive gaps on magnetic tape.

The maximum amount of information that can be transferred in one I/O operation is one block. An I/O operation is terminated when the associated storage area is exhausted or the end of the block is reached, whichever occurs first. For some operations, such as writing on a magnetic-tape unit or at an inquiry station, blocks are not defined, and the amount of information transferred is controlled only by the program.

Channel-Address Word

The channel-address word (CAW) specifies the subchannel key and the address of the first CCW associated with START I/O or START I/O FAST RELEASE. The channel refers to the CAW only during the execution of START I/O or START I/O FAST RELEASE. The CAW is fetched from storage location 72. The pertinent information thereafter is stored in the subchannel, and the program is free to change the contents of the CAW. Fetching of the CAW by the channel does not affect the contents of the location.

The CAW has the following format:



The fields in the CAW are allocated for the following purposes:

Subchannel Key: Bits 0-3 form the access key for all fetching of CCWs and output data and for the storing of input data associated with START I/O and START I/O FAST RELEASE. This key is matched with a storage key during these storage references. For details, see the section "Key-Controlled Protection" in Chapter 3, "Storage."

CCW Address: Bits 8-31 designate the location of the first CCW in storage.

Bit positions 4-7 of the CAW must contain zeros. The three low-order bits of the CCW address must be zeros to specify the CCW on integral boundaries for doublewords. If any of these restrictions is violated, or if the CCW address specifies a storage location which is not provided or is protected against fetching or is in a disconnected page, START I/O and, in some cases, START I/O FAST RELEASE, cause the status portion of the CSW to be stored, with the protection-check or program-check bit set to one. In this event, the I/O operation is not initiated.

Programming Note

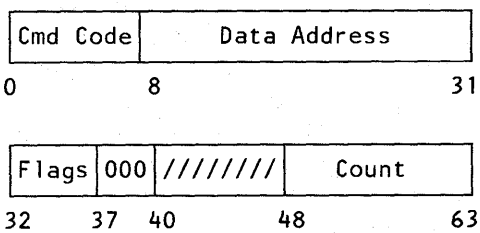
Bit positions 4-7 of the CAW, which presently must contain zeros, may in the future be assigned to the control of new functions. It is, therefore, recommended that these bit positions not be set to ones for the purpose of obtaining an intentional program-check indication.

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 CCWs can be located anywhere in storage, and more than one can be associated with a START I/O or START I/O FAST RELEASE.

The first CCW is fetched during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When START I/O FAST RELEASE is executed independent of the device, the first CCW is fetched subsequent to the execution of START I/O FAST RELEASE. 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 CCWs by the channel does not affect the contents of the location in 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 a location in 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 incorrect-length is to be indicated to the program. When this bit is one and the CD flag is zero, the incorrect-length indication is suppressed. When both the CC and SLI flags are one, command chaining takes place regardless of any incorrect-length situation.

Skip (SKIP) Flag: Bit 35, when one, specifies suppression of the 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 bytes 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. Otherwise, a program-check condition is generated. When the first CCW designated by the CAW does not contain zeros in bit positions 37-39, 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 or START I/O FAST RELEASE being executed as START I/O. Detection of this condition during data chaining causes the I/O device to be signaled to conclude the operation. When the absence of these zeros is detected during command chaining or subsequent to the execution of START I/O FAST RELEASE, the new operation is not initiated, and an interruption condition is generated.

The contents of bit positions 40-47 of the CCW are ignored.

Programming Note

Bit positions 37-39 of the CCW, which presently must contain zeros, may in the future be assigned to the control of new functions. It is recommended, therefore, that these bit positions not be set to ones for the purpose of obtaining an intentional program-check indication.

Command Code

The command code, bit positions 0-7 of the CCW, specifies to the channel and the I/O device the operation to be performed. A detailed description of each command appears under "Commands."

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 leftmost bit positions contain modifier bits. The modifier bits specify to the device how the command is to be executed. They may, for example, cause the device to compare data received during a write operation with data previously recorded, and they may specify such information 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 SL 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
XXXX 0000	Invalid
MMMM MM01	Write
MMMM MM10	Read
MMMM 1100	Read Backward
MMMM MM11	Control
MMMM 0100	Sense
XXXX 1000	Transfer in Channel

Whenever the channel detects an invalid command code during the initiation of a command,

a program check 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 or START I/O FAST RELEASE being executed as START I/O. When the invalid code is detected during command chaining or subsequent to the execution of START I/O FAST RELEASE, 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.

Designation of Storage Area

The storage area associated with an I/O operation is defined by one or more CCWs. A CCW defines an area by specifying the address of the first byte to be transferred and the number of consecutive bytes contained in the area. The address of the first byte appears in the data-address field of the CCW. The number of bytes contained in the storage area is specified in the count field.

In write, read, control, and sense operations, storage locations are used in ascending order of addresses. As information is transferred to or from storage, the address from the address field is incremented, and the count from the count field is decremented. The read-backward operation places data in storage in a descending order of addresses, and both the count and the address are decremented. When the count reaches zero, the storage area defined by the CCW is exhausted.

Any storage location that is provided can be used in the transfer of data to or from an I/O device if the location is in a page that is in the addressable or connected state and is not protected against the type of reference. Similarly, a CCW can be located in any part of storage if the location is in a page that is in the addressable or connected state and is not protected against a fetch-type reference.

When the first CCW is designated by the CAW as being at a storage location that is not provided, 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 or START I/O FAST RELEASE being executed as START I/O. When, subsequently, during the operation or chain of operations, the channel refers to a storage location that is not provided, an interruption condition indicating program check is generated, and the device is signaled to terminate the operation.

When the first CCW designated by the CAW is in a disconnected page or in a location that is protected against a fetch-type reference, the I/O operation is not initiated, and the status portion of the CSW with the protection-check indication is stored during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When, subsequently, during the I/O operation or chain of operations, the channel refers to a disconnected page or a protected location, an interruption condition indicating protection check is generated, and the device is signaled to terminate the operation.

During an output operation, the channel may fetch data from storage before the time the I/O device requests the data. Any number of bytes specified by the current CCW may thus be prefetched. When data chaining during an output operation, and for some block-multiplexer channels when data chaining during an input operation, the channel may prefetch the next CCW at any time during the execution of the current CCW.

Prefetching may cause the channel to refer to storage locations that are protected or not provided or in disconnected pages. Such errors detected during prefetching of data or CCWs do not affect the execution of the operation and do not cause error indications until the I/O operation actually attempts to use the data or until the CCW takes control. If the operation is concluded by the I/O device or by HALT I/O, HALT DEVICE, or CLEAR I/O before the invalid information is needed, no program check or protection check is generated.

The count field in the CCW can specify any number of bytes from one to 65,535. Except for a CCW specifying transfer in channel, which has no count field, the count field may not contain the value zero. Whenever the count field in the CCW initially contains a zero, a program check 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 or START I/O FAST RELEASE being executed as 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 or subsequent to the execution of START I/O FAST RELEASE 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 or START I/O FAST RELEASE by fetching a new CCW. Such fetching of a new CCW is called chaining, and the CCWs belonging to such a sequence are said to be chained.

Chaining takes place between CCWs located in successive doubleword locations in storage. It proceeds in an ascending order of addresses; that is, the address of the new CCW is obtained by adding 8 to the address of the current CCW. Two chains of CCWs located in noncontiguous storage areas can be coupled for chaining purposes by a transfer-in-channel command. All CCWs in a chain apply to the I/O device specified in the original START I/O or START I/O FAST RELEASE.

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 the figure "Channel-Chaining Action."

The specification of chaining is effectively propagated through a transfer-in-channel command. When in the process of chaining a transfer-in-channel command is fetched, the CCW designated by the transfer in channel is used for the type of chaining specified in the CCW preceding the transfer in channel. The CD and CC flags are ignored in the transfer-in-channel command.

Flags in Current CCW			Action in Channel upon Exhaustion of Count or Receipt of Channel End			
				Regular Operation		
CD	CC	SLI	Immediate Operation	I	II	III
0	0	0	End, NIL	Stop, IL	End, NIL	End, IL
0	0	1	End, NIL	Stop, NIL	End, NIL	End, NIL
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, NIL	Chain Data	*	End, IL
1	0	1	End, NIL	Chain Data	*	End, IL
1	1	0	End, NIL	Chain Data	*	End, IL
1	1	1	End, NIL	Chain Data	*	End, IL

Explanation:

I	Count exhausted, end of block at device not reached.
II	Count exhausted and channel end from device.
III	Count not exhausted and channel end from device.
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 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.
NIL	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 residual count is zero but data chaining is indicated at the time the device provides channel end cannot validly occur. When data chaining is indicated, the 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.

Channel-Chaining Action

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. When all data

designated by the current CCW has been transferred to storage or to the device, data chaining causes the operation to continue, using the storage area designated by the new CCW. The contents of the command-code field of the new CCW are ignored, unless they specify transfer in channel.

Data chaining is considered to occur immediately after the last byte of data designated by the current CCW has been transferred to storage or to the device. When the last byte of the transfer has been placed in 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 signals 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 concluded operation pertains to the new CCW.

If programming errors are detected in the new CCW or during its fetching, the error indication is generated, and the device is signaled to conclude the operation when it attempts to transfer data designated by the new CCW. If the device signals channel end after the new CCW takes control but before transferring any data designated by the new CCW, program check or protection check is indicated in the CSW associated with the termination. The contents of the CSW pertain to the new CCW unless a program check or protection check is generated while fetching the new CCW or while fetching or executing an intervening transfer-in-channel command. A data address which causes a program check or protection check gives an error indication only after the I/O device has attempted to transfer data to or from the addressed storage location.

When data chaining during an output operation, the channel may fetch the new CCW from storage ahead of the time data chaining occurs. Similarly, some block-multiplexer channels may prefetch the new CCW when data chaining during input. Any programming errors in a prefetched CCW, however, do not affect the execution of the operation until all data designated by the current CCW has been transferred to the I/O device on output or to storage on input. If the device concludes the operation before all data designated by the current CCW has been transferred or if data chaining is suppressed for any other reason, the errors associated with the prefetched CCW are not indicated to the program.

Only one CCW describing a data area may be prefetched. If the prefetched CCW specifies transfer in channel, only one more CCW may be fetched before the exhaustion of the current CCW.

Programming Note

Data chaining may be used to rearrange data as it is transferred between storage and an I/O device. Data chaining permits data to be transferred to or from noncontiguous areas of storage, and, when used in conjunction with the skipping function (see the section "Skipping" later in this chapter), data chaining enables the program to place in storage selected portions of a block of data.

When, during an input operation for a channel that does not prefetch CCWs on input, the program specifies data chaining to a location into which data has been placed under the control of the current CCW, the channel, in fetching the next CCW, fetches the new contents of the location. This is true even if the location contains the last byte transferred under the control of the current CCW. When, on input, a channel program data-chains to a CCW placed in storage by the CCW specifying data chaining, the block is said to be self-describing. A self-describing block contains one or more CCWs that specify storage locations and counts for subsequent data in the same block.

The use of self-describing blocks is equivalent to the use of unchecked data. An I/O data-transfer malfunction that affects validity of a block is signaled only at the completion of data transfer. The error 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 is in error, the use of the CCW in the current operation may cause subsequent data to be placed in wrong storage locations with resultant destruction of the contents of those locations.

Self-describing blocks cannot be used with a channel that prefetches CCWs when data chaining on input.

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 receipt of the device-end signal for the current operation. When command chaining takes place, the completion of the current operation does not generate an interruption condition, 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 at the device.

Command chaining takes place and the new operation is initiated only if no unusual situations 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 signaling one of the following status 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, channel end must have been signaled before device end, with all other status bits set to zeros. If status such as attention, unit check, unit exception, incorrect length, program check, or protection check has occurred, the sequence of operations is concluded, and the status associated with the current operation causes an interruption condition to be generated. The new CCW in this case is not fetched. Incorrect length does not suppress command chaining if the current CCW has the SLI flag set to one.

An exception to sequential chaining of CCWs occurs when the I/O device presents status modifier with device end. When no unusual conditions have been detected and command chaining is specified or when command retry has been previously signaled and an immediate retry could not be performed, the combination of status modifier and device end causes the channel to alter the sequential execution of CCWs. If command chaining was specified, the status causes the channel to chain to the CCW whose storage address is 16 higher than that of the CCW that specified chaining. If command retry was previously signaled and immediate retry could not be performed, the status causes the channel to command-chain to the CCW whose storage address is 8 higher than that of the CCW for which retry was initially signaled.

When both command and data chaining are used, the first CCW associated with the operation specifies the operation to be executed, and the last CCW indicates whether another operation follows.

Programming Note

Command chaining makes it possible for the program to initiate transfer of multiple blocks by means of a single START I/O or START I/O FAST RELEASE. It also permits a subchannel to be set up for the 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 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 storage. Chaining is not precluded by skipping. In the case of data chaining, normal operation is resumed if the skip flag in the new CCW is zero.

When the skip flag is set to one, the data address in the CCW is not checked.

Programming Note

Skipping, when combined with data chaining, permits the program to place in storage selected portions of a block from an I/O device.

Program-Controlled Interruption

The program-controlled-interruption (PCI) function permits the program to cause an I/O interruption during the 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 START I/O FAST RELEASE or in a CCW fetched during chaining. Neither the PCI flag nor the associated interruption affects the execution of the current operation.

Whenever the PCI flag in the CCW is one, an interruption condition is generated in the channel. 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 has 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 I/O interruptions are allowed. No predictable relationship exists between the time the interruption due to the PCI flag occurs and the

;ress of data transfer to or from the area gnated by the CCW, but the fields within the V pertain to the same instant of time.
f chaining occurs before the interruption due to PCI flag has taken place, the PCI interruption dition is carried over to the new CCW. This yover occurs both on data and command ining and, in either case, the interruption dition is propagated through the transfer-in- nnel command. The interruption conditions due he PCI flags are not stacked; that is, if another W is fetched with a PCI flag before the erruption due to the PCI flag of the previous W has occurred, only one interruption takes ce.
A CSW containing the PCI bit set to one may be red by an interruption while the operation is still ceeding or by an interruption, TEST I/O, or EAR I/O upon the termination of the operation. CSW cannot be stored by TEST I/O while the channel is in the working state.
When the CSW is stored by an interruption fore the operation or chain of operations has en concluded, the CCW address is 8 greater than address of the current CCW, and the count is predictable. All unit-status bits in the CSW are ro. If the channel has detected any unusual uations, such as channel-data check, program eck, or protection check by the time the terruption occurs, the corresponding channel- atus bit is one, although the status in the sub- annel is not reset and is indicated again upon the rmination of the operation.
A unit-status bit set to one in the CSW indicates at the operation or chain of operations has been ncluded. The CSW in this case has its regular rmat with the PCI bit set to one.
However, when the interruption due to the PCI ag is delayed until the operation at the subchannel concluded, two interruptions from the subchannel ay still take place. The first interruption indicates nd clears the interruption condition due to the CI flag, and the second provides the CSW ssociated with the ending status. Whether one or wo interruptions occur depends on the model and n whether the interruption condition due to the CI flag has been assigned the highest priority for nterruption at the time of conclusion. TEST I/O r CLEAR I/O addressed to the device associated vith an interruption condition in the subchannel lears the interruption condition due to the PCI lag, as well as the one associated with the onclusion.

The setting of the PCI flag is inspected in every CCW except those specifying transfer in channel, where it is ignored. The PCI flag is also ignored during initial program loading.

Programming Notes

1. Since no unit-status bits are set to ones in the CSW associated with the conclusion of an operation of a selector channel by HALT I/O or HALT DEVICE, unit-status bits and the PCI bit set to ones are not necessary for the operation to be concluded. When status in a selector channel includes PCI at the time the operation is concluded by HALT I/O or HALT DEVICE, the CSW associated with the concluded operation is indistinguishable from the CSW provided by an interruption during execution of the operation.
2. Program-controlled interruption provides a means of alerting the program to the progress of chaining during an I/O operation. It permits programmed dynamic storage allocation.

Commands

The figure "Channel-Command Codes" 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	Flags
Write	MMMM MM01	CD CC SLI PCI
Read	MMMM MM10	CD CC SLI SKIP PCI
Read backward	MMMM 1100	CD CC SLI SKIP PCI
Control	MMMM MM11	CD CC SLI PCI
Sense	MMMM 0100	CD CC SLI SKIP PCI
Transfer in channel	XXXX 1000	

Explanation:

CD	Chain data
CC	Chain command
SLI	Suppress length indication
SKIP	Skip
PCI	Program-controlled interruption
M	Modifier bit
X	Ignored

Channel-Command Codes

All flags have individual significance, except that the CC and SLI flags are ignored when the CD flag is set to one. The SLI flag is ignored on immediate operations, in which case the incorrect-length indication is suppressed, regardless of the setting of

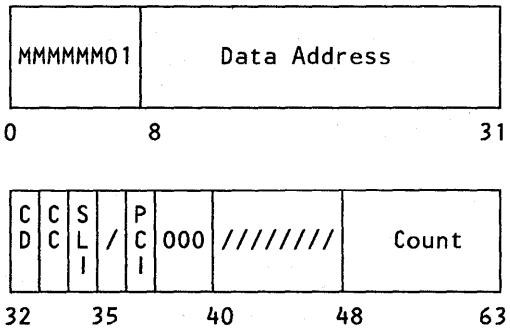
the flag. The PCI flag is ignored during initial program loading.

Each command is described below, and the format is illustrated.

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 storage should not be destroyed until the program has verified that no malfunction affecting the transfer and recording of data was detected.

Write



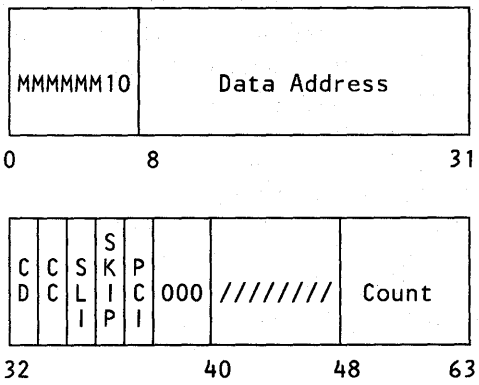
A write operation is initiated at the I/O device, and the subchannel is set up to transfer data from storage to the I/O device. Data in storage is fetched in an ascending order of addresses, starting with the address specified in the CCW.

A CCW used in a write operation is inspected for the CD, CC, SLI, and 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.

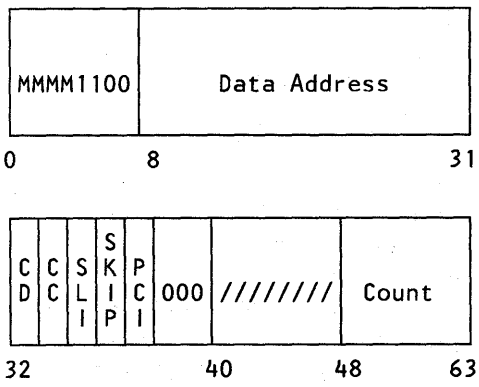
Read



A read operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to 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 is placed in storage 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 flag–CD, CC, SLI, SKIP, and PCI. Bit positions 0-5 of the CCW contain modifier bits.

Read Backward

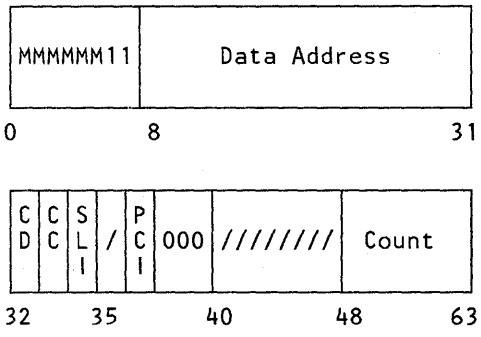


A read-backward operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to storage. On magnetic-tape units, read backward causes reading to be performed with the tape moving backward. The bytes of data within a block are sent to the channel in a sequence opposite to that on writing. The channel places the bytes in storage in a descending order of address, starting with the address specified

in the CCW. The bits within a 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 flag–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 storage to the device. The device interprets the data as control information. The control information, if any, is fetched from storage in an ascending order of addresses, starting with the address specified in the CCW. A control command may be used to initiate at the I/O device an operation not involving transfer of data, such as backspacing or rewinding magnetic tape or positioning a disk-access mechanism.

For many control functions, the entire operation is specified by the modifier bits in the command code, and the function is performed as an immediate operation (see the section "Immediate Operations" later in this chapter). If the command code does not specify the entire control function, the data-address field of the CCW designates the location containing the required additional information. This control information may include a code further specifying the operation to be performed or an external address, such as the disk address for the seek function, and is transferred in response to requests by the device.

A control command code containing zeros for the six modifier bits is defined as a *no-operation.* The no-operation order causes the addressed device to respond with channel end and device end without causing any action at the device. The control command can be executed as an immediate operation, or the device can delay the status until after the initial selection sequence is completed.

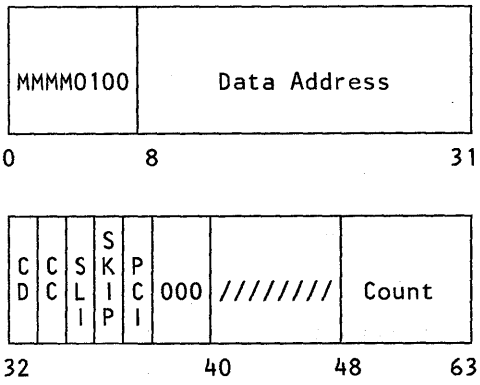
Other operations that can be initiated by means of the control command depend on the type of I/O device. These operations and their codes are specified in the SL publication for the device.

A CCW used in a control operation is inspected for the CD, CC, SLI, and 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 (other than transfer in channel) 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 has 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 storage. The data is placed in storage in an ascending order of addresses, starting with the address specified in the CCW.

Data transferred during a sense operation provides 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 in the CSW 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 situations 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 indicated when the program issues an invalid sequence of commands, such as write to a direct-access storage device without previous designation of the block. Command reject may also be indicated when invalid data is transferred and the data is treated as an extension of the command. For example, command reject is indicated when an invalid seek argument is transferred to a direct-access storage device.

Intervention Required: The last operation could not be executed because of a situation requiring some type of intervention at the device. This bit indicates situations such as the hopper in a card punch being empty or the printer being out of paper. It is also turned on when the addressed device is not ready, 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 from the channel. During writing, bus-out check indicates that incorrect data has been recorded at the device, but this 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 suppress checking for situations that would cause command reject and intervention required.

Equipment Check: During the last operation, the device or the control unit has detected equipment malfunctioning, such as an invalid card-hole count or a 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 errors 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 storage. The control unit forces correct parity on data sent to the channel. On writing, data check 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 is transferred to or from a nonbuffered 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 transferred to storage may be used to fill the gap. On an output operation, overrun indicates that data recorded at the device may be invalid. The overrun bit is also set to one 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 SL 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 bit to be set to one. If the control unit detects an equipment malfunction, or invalid parity of the sense command code, the equipment-check or bus-out-check bit is set to one, and unit check is indicated in the unit-status byte.

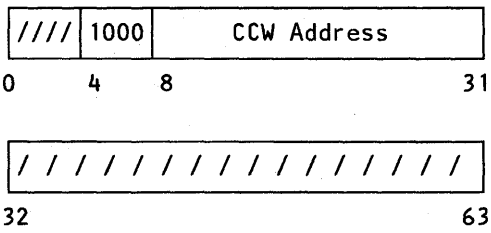
Devices that can provide special diagnostic sense information or can be instructed to perform other

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 that pertains to the last I/O operation or other action at a device may be reset any time after the completion of a sense command addressed to that device. Any command addressed to the control unit of a device, other than the no-operation command and the command which results from a TEST I/O instruction, may be allowed to reset the sense information, provided that the busy bit is not included in the initial status. The sense information may also be changed as a result of asynchronous actions, such as when attention or not-ready-to-ready device-end status is generated.

A CCW used in a sense operation is inspected for every flag–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 in storage designated by the data-address field of the CCW specifying transfer in channel. The transfer-in-channel command does not initiate any I/O operation at the channel, and the I/O device is not signaled. The purpose of the transfer-in-channel command is to provide chaining between CCWs not located in adjacent doubleword locations in an ascending order of addresses. The command can occur in both data and command chaining.

The first CCW designated by the CAW must not specify transfer in channel. When this restriction is violated, no I/O operation is initiated, and a program check is generated. The error causes the status portion of the CSW, with the program-check status bit set to one, to be stored during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When START I/O FAST RELEASE is executed independent of the device, the error causes an interruption condition to be generated.

To address a CCW on integral boundaries for doublewords, a CCW specifying transfer in channel must contain zeros in bit positions 29-31. Furthermore, a CCW specifying a transfer in channel must not be fetched from a location designated by an immediately preceding transfer in channel. When either of these errors is detected, a program check is 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.

Command Retry

Some channels have the capability to perform command retry, 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 presenting either of two status-bit combinations by means of a special communication sequence with the channel. When immediate retry can be performed, the control unit signals a channel-end, unit-check, and status-modifier status-bit combination, together with device end. When immediate retry cannot be performed, the presentation of device end is delayed until the control unit is prepared. If device end and no other status bits are signaled, command retry is performed. If device end is accompanied by status modifier, command retry is not performed, and the channel command-chains to the CCW following the one for which retry was signaled. When any other status bit accompany device end or device end and status modifier, an interruption condition is generated. In this situation, the CSW will contain the status indications causing the interruption condition.

When the channel is not capable of performing command retry, the retry is suppressed, and an interruption condition is generated. The CSW will contain the channel-end, unit-check, and status-modifier status indications, along with any other appropriate status.

During command retry, the channel action is similar to that taken when command chaining. Thus, when command retry is performed, a START I/O initiating an immediate operation for which command chaining is not indicated in the CCW

causes a condition code 0, rather than a condition code 1, to be set. The subsequent termination of the I/O operation causes an interruption condition to be generated. During command retry, the CCW may be refetched.

Programming Note
The following possible results of a command retry must be anticipated by the program:

1. A CCW with the PCI flag set to one 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 0 rather than a condition code 1 to be set. This setting of the condition code occurs if the control unit signals command retry at the time initial status is signaled to the channel. An interruption condition is generated upon completion of the operation.
3. If a CCW used in an operation is changed before that operation has been successfully completed, the results are unpredictable.
4. A CSW stored after the initiation of a retry but before the presentation of device end, as when an interruption condition due to the PCI flag is taken, contains the address of the command to be retried plus 8.
5. If a HALT I/O, HALT DEVICE, or CLEAR I/O instruction is issued after the initiation of a retry but before the presentation of device end, the CSW contains the address of the command to be retried plus 8.
6. On a multiplexer channel, chained CCWs which might ordinarily have been executed in a burst may, upon the occurrence of command retry, cause multiplexing to occur, with the result that the channel becomes unexpectedly available.
7. Command chaining may occur even though the CCW does not indicate command chaining. This can occur if immediate retry is not requested and the control unit or device presents a status of device end and status modifier.

Conclusion of Input/Output Operations

When the operation or sequence of operations initiated by START I/O or START I/O FAST RELEASE is ended, the channel and the device generate status. Status can be brought to the attention of the program by means of an I/O interruption, by TEST I/O or CLEAR I/O, or, in certain cases, by START I/O or START I/O FAST RELEASE. This status, 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 Conclusion

Normally an I/O operation at the subchannel lasts until the device signals channel end. Channel end can be signaled during the sequence initiating the operation, or later. When the channel detects equipment malfunctioning or an I/O system reset is performed, the channel disconnects the device without receiving channel end. The program can force a device to be disconnected prematurely by issuing CLEAR I/O, HALT I/O, or HALT DEVICE.

Conclusion at Operation Initiation
After the addressed channel and subchannel have been verified to be in a state where START I/O or START I/O FAST RELEASE 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 during the execution of START I/O, either during or subsequent to the execution of START I/O FAST RELEASE, 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 situations during the initiation of an operation, the command is said to be rejected.

Rejection of the command during the execution of START I/O or START I/O FAST RELEASE is indicated by the setting of the condition code in the PSW. Unless the device is not operational, the reasons for the rejection are detailed by the portion of the CSW stored by START I/O or START I/O FAST RELEASE. The device is not started, no interruption conditions are generated, and the subchannel is available subsequent to the initiation sequence. The device is immediately available for the initiation of another operation, provided the command was not rejected because the device was busy or not operational.

When an unusual situation 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 reasons for the rejection are indicated to the program by means of the corresponding status bits in the CSW. The not-operational state of the I/O device, which during the execution of START I/O and sometimes during the execution of START I/O FAST RELEASE causes condition code 3 to be set, instead causes the interface-control-check bit to be set to one. The new operation at the I/O device is not started.

When START I/O FAST RELEASE is executed by a channel independent of the addressed device, tests for most program-specified information, for control-unit and device availability, for control-unit and device status, and for most errors are performed subsequent to the execution of START I/O FAST RELEASE. Some situations which would have caused a condition code 1 or 3 to be set had the instruction been START I/O instead cause an interruption condition to be generated. The CSW, when stored, indicates that the interruption condition is a deferred condition code 1 or 3.

Immediate Operations

Some control commands cause the I/O device to signal channel end immediately upon receipt of the command code. An I/O operation causing channel end to be signaled during the initiation sequence is called an *immediate operation.*

When the first CCW designated by the CAW during a START I/O or START I/O FAST RELEASE executed as a START I/O initiates an immediate operation with command chaining not indicated and command retry not occurring, no interruption condition is generated. In this case, channel end is brought to the attention of the program by causing START I/O or START I/O FAST RELEASE to store the CSW status portion. 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.

An immediate operation initiated by the first CCW designated by the CAW during a START I/O FAST RELEASE executed independent of the addressed device appears to the program as a nonimmediate command. That is, any status generated by the device for the immediate command, or for a subsequent command if command chaining occurs, causes an interruption condition to be generated.

When command chaining is specified after an immediate operation and no unusual situations have been detected during the execution, or when command retry occurs for an immediate operation, neither START I/O nor START I/O FAST RELEASE causes the immediate storing of CSW status. The subsequent commands in the chain are handled normally, and channel end 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 has been transferred to or from the device.

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 SL publication for the device.

Conclusion of Data Transfer

When the device accepts a command, the subchannel is set up for data transfer. The subchannel is in the working state during this period. Unless the channel detects equipment malfunctioning or the operation is concluded by CLEAR I/O, or, on the selector channel, the operation is concluded by CLEAR I/O, HALT I/O, or HALT DEVICE, 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 situations, channel end 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 any unusual situations. The device can signal channel end at any time after initiation of the operation, and the signal may occur before any data has been transferred.

For operations not involving data transfer, the device normally controls the timing of channel end. The duration of data-transfer operations may be

variable and may be controlled by the device or the channel.

Excluding equipment errors, CLEAR I/O, HALT DEVICE, and HALT I/O, the channel signals the device to conclude data transfer whenever any of the following events occurs:

1. The storage areas specified for the operation are exhausted or filled.
2. A program check is detected.
3. A protection check is detected.
4. A chaining check is detected.

The first event occurs when the channel has stepped the count to zero in the last CCW associated with the operation. A count of zero indicates that the channel has transferred all information specified by the program. The other three events are due to errors and cause premature conclusion of data transfer. In every case, the conclusion is signaled in response to a service request from the device and causes data transfer to cease. If the device has no blocks defined for the operation (such as writing from magnetic tape), it concludes the operation and generates channel end.

The device can control the duration of an operation and the timing of channel end. On certain operations for which blocks are defined (such as reading from 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 conclude data transfer.

If the initial data address in the CCW refers to a storage location that is not provided or to a disconnected or protected page, no data is transferred during the operation, and the device is signaled to conclude the operation in response to the first service request. On writing, devices such as magnetic-tape units request the first byte of data before any mechanical motion is started and, if the initial data address refers to a storage location that is not provided or to a disconnected or protected page, the operation is concluded 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 is transferred depends on the type of device and is specified in the SL publication for the device.

When command chaining takes place, the subchannel is in the working state from the time the first operation is initiated until the device signals channel end for the last operation of the chain. On the selector channel, the device executing the operation stays connected to the channel and the whole channel is in the working state during the entire execution of the chain of operations. On the multiplexer channel, an operation in the burst mode causes the channel to be in the working state only while transferring a 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.

Any unusual situations cause command chaining to be suppressed and an interruption condition to be generated. The unusual situations 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 situation by the time the channel-end signal for the operation is received, the chain is ended as if the operation during which the situation 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 or HALT DEVICE

The instructions HALT I/O and HALT DEVICE cause the current operation at the addressed channel or subchannel to be immediately terminated. 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 or HALT DEVICE is not necessarily contingent on the receipt of a service request from the device.

When HALT I/O is issued to a channel operating in burst mode, the channel issues the halt signal to the device currently operating with the channel, regardless of the device address specified with the HALT I/O instruction. 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 command chaining to be immediately suppressed.

When HALT DEVICE is issued to a channel operating in burst mode, the halt signal is issued to the device involved in the burst-mode operation only if that device is the one to which the HALT DEVICE is addressed. If the operation thus terminated is in the data-transfer portion of the operation, data transfer is immediately terminated, and the device is disconnected from the channel. If the terminated burst involves a selector channel executing a chain of operations and the device has already provided channel end for the current operation, HALT DEVICE causes the device to be disconnected and command chaining to be immediately suppressed. If, on a selector channel, the device involved in the burst is not the one to which the HALT DEVICE is addressed, no action is taken. If, on a multiplexer channel, the device involved in the burst is not the one to which the HALT DEVICE is addressed, HALT DEVICE causes any operation for the addressed device to be terminated at the addressed subchannel by suppressing any further data transfer or command chaining for that device.

When HALT I/O or HALT DEVICE is issued to a channel not operating in burst mode, the addressed device is selected, and the halt signal is issued as the device responds. On a multiplexer channel, command chaining, if indicated in the subchannel, is immediately suppressed.

The termination of an operation by HALT I/O or HALT DEVICE 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 channel-status bits reflect the unusual situations, if any, detected during the operation. If HALT I/O or HALT DEVICE is issued before all data specified for the operation has been transferred, incorrect length is indicated, subject to the control of the SLI flag in the current CCW. The execution of HALT I/O or HALT DEVICE 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 signals channel end. 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 subchannel-key, CCW-address, count, and channel-status fields of the associated CSW. Channel end is not made available to the program when HALT I/O or HALT DEVICE 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 signaled. These signals are handled as for any other I/O operation.

The termination of an operation by HALT I/O or HALT DEVICE on a multiplexer 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 or HALT DEVICE is issued after the device has signaled channel end and the subchannel is executing a chain of operations, channel-end 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 byte provided by the device and the channel. If HALT I/O or HALT DEVICE 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.

The termination of a burst operation by HALT I/O or HALT DEVICE on a block-multiplexer channel may, depending on the model and the type of subchannel, take place as for a selector channel or may allow the subchannel to remain in the working state until the device provides ending status.

Programming Note

The count field in the CSW associated with an operation terminated by HALT I/O or HALT DEVICE is unpredictable.

Termination by CLEAR I/O

The termination of an operation by CLEAR I/O causes the subchannel to be set to the available state and causes a CSW to be stored. The validity of the CSW fields is defined in the instruction CLEAR I/O earlier in this chapter.

When CLEAR I/O terminates an operation at a subchannel in the interruption-pending state, up to three subsequent interruption conditions related to the operation can occur. Since CLEAR I/O causes the subchannel to be made available, these interruption conditions will result in only the unit-status portion of the CSW being indicated.

The first interruption condition arises on a selector channel when channel end is signaled to the channel. This occurs only when the interruption-pending states of the channel and subchannel at the execution of CLEAR I/O were due to the previous execution of HALT I/O or HALT DEVICE.

The second and third interruption conditions arise when control-unit end, if any, and device end are signaled to the channel.

When CLEAR I/O terminates an operation at a subchannel in the working state, up to four subsequent interruption conditions related to the operation can occur. For all of these conditions, only the status portion of the CSW is indicated.

The first interruption condition arises on certain channels when the terminated operation was in the midst of data transfer. Since the device is not signaled to terminate the operation during the execution of CLEAR I/O unless the channel is working with the addressed device when the instruction is received, the device may, subsequent to the CLEAR I/O, attempt to continue the data transfer. The channel responds by signaling the device to terminate data transfer. Depending on the channel, the need to signal the device to terminate data transfer may be ignored or may be considered an interface-control check which creates an interruption condition. Only channel status is indicated in the CSW.

The second interruption condition occurs when channel-end status is received from the device. The third and fourth conditions occur when control-unit end, if any, and device end are presented to the channel. In these three cases, only unit status is indicated in the CSW.

Termination Due to Equipment Malfunction

When channel-equipment malfunctioning is detected or invalid signals are received from a device, 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 to the termination by an I/O interruption, and the associated CSW indicates channel-control check or interface-control check. However, when the nature of the malfunction prevents an I/O interruption, a machine-check interruption occurs, and a CSW is not stored. A malfunction may cause the channel to perform the I/O selective reset or to generate the halt signal.

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. The conditions are indicated in an associated CSW which is stored at the time of interruption. These conditions can be caused by the program or by an external event at the device.

Interruption Conditions

A request for an I/O interruption is called an I/O-interruption condition, or, in this chapter, simply an interruption condition. An interruption condition can be brought to the attention of the program only once and is cleared when it causes an interruption. Alternatively, an interruption condition can be cleared by TEST I/O or CLEAR 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 or START I/O FAST RELEASE. The latter include interruption conditions caused by attention, device end, and control-unit end, and channel end when provided by a device after conclusion of the operation.

The device attempts to initiate a request to the channel for an I/O interruption whenever it detects any of the following:

Channel end
Control-unit end
Device end
Attention

The channel combines the above status with information in the subchannel and either causes an I/O interruption or continues command chaining. When command chaining takes place, channel end and device end do not cause an interruption and are not made available.

The channel may also, if command chaining exists, create an interruption condition, which can be due to the following:

Unit check
Unit exception
Busy indication from device
Program check

When an operation initiated by command chaining is terminated because of an unusual situation 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. This interruption condition is created at the device only 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 situation is detected by the device (unit check or unit exception) the unit-status field of the associated CSW identifies the condition. If the unusual situation is detected by the channel, as in the case of program and protection check, the identification of the error is preserved in the subchannel and appears in the channel-status field of the associated CSW.

An interruption condition caused by the device may be accompanied by channel and other unit status. Furthermore, more than one condition associated with the same device can be cleared at the same time. As an example, when channel end is not cleared at the device by the time device end is generated, both may be indicated in the CSW and cleared at the device concurrently.

However, at the time the channel assigns highest priority for interruptions to an interruption 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 when the CSW is stored, even if the status is generated before the interruption condition is cleared.

When the channel is not working, a device that is interruption-pending may attempt to initiate a request to the channel for an I/O interruption by presenting a nonzero status byte to the channel. Depending on the channel, some models may accept the status in the subchannel. Alternatively, some models may signal the device to hold the status until the channel is capable of causing an interruption. In this case, the channel selects the device to obtain the status when the interruption occurs. The status stored by the channel is the status presented by the device at interruption time and, because of changed conditions at the device, may not be the same status presented by the device initially. Specifically, a status of zero, busy, or busy and status modifier may be stored.

When the channel detects any of the following, it generates an interruption condition without necessarily communicating with or having received the status byte from the device:

- PCI flag in a CCW
- Execution of HALT I/O or HALT DEVICE on a selector channel
- Channel-available interruption (CAI)
- A programming error associated with the CCW or first IDAW following the SIOF function

The interruption conditions from the channel, except for CAI, can be accompanied by other channel-status indications, but none of the device status bits is on when the channel initiates the interruption.

Channel-Available Interruption

The channel-available-interruption (CAI) condition is provided on block-multiplexer channels and causes the entire CSW to be replaced by a new set of bits. All fields of the CSW are set to zero. The I/O address stored contains a zero device address and a channel address identifying the interrupting channel.

The channel generates the CAI condition only if it previously had responded with a condition code 2 to an I/O instruction other than HALT I/O or HALT DEVICE and if the working state thus indicated no longer exists. When the working state which caused condition code 2 was due to a subchannel busy with a device other than the one addressed, the conclusion of the working state is not signaled by a CAI. Since any other interruption condition (except PCI) accomplishes the same function as CAI, a CAI condition is reset upon the occurrence of any interruption (except PCI) on that channel. Some channels also reset a CAI condition when another interruption condition (except PCI) is cleared by a TEST I/O on the same channel. The occurrence of another channel-working state before the CAI causes the CAI condition to be suspended until the working state ends.

Programming Note

The CAI is designed to inform the program that a channel which previously indicated busy is no longer busy. The CAI condition pending in a

channel does not cause the rejection of a subsequent START I/O or START I/O FAST RELEASE but does cause a condition code 1 to be returned to TEST CHANNEL. The CAI can therefore be used as a tool for keeping I/O requests in sequence by using it in conjunction with TEST CHANNEL. A channel which responded with condition code 2 because the channel was busy does not subsequently respond with a condition code 0 to a TEST CHANNEL without clearing an interruption condition in the interim.

Priority of Interruptions

Generation of interruption conditions is 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 interruption conditions is controlled by two types of mechanisms–one establishes the priority among interruption conditions within a channel, and another establishes priority among interruption conditions from different channels. A channel requests an I/O interruption only after it has established priority among interruption conditions. The status associated with interruption conditions is preserved in the devices or channels until accepted by the CPU.

Assignment of priority among requests for interruption associated with devices on any one channel is a function of the type of channel, the type of interruption condition, and the position of the device on the I/O interface. A device's position on the interface is not related to its address. Interruption conditions from different devices do not necessarily occur in the sequence in which they are generated. However, multiple interruption conditions for a single device are presented in the sequence in which they are generated.

The priorities among requests for I/O interruptions from different channels depend on channel addresses. The priorities of channels 1-15 are in the order of their addresses, with channel 1 having the highest priority. The priority of byte-multiplexer channel 0 is undefined. Its priority may be above, below, or between those priorities of channels 1-15.

Interruption Action

An I/O interruption can occur only when the CPU is enabled for I/O interruptions. The interruption occurs at the completion of a unit of operation. If a channel has established the priority among interruption conditions, while the CPU is disabled for I/O interruptions, the interruption occurs immediately after the completion of the instruction enabling the CPU and before the next instruction is executed. This interruption is associated with the highest priority condition for the channel. If interruptions are allowed from more than one channel 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 interruption is allowed, the interruption does not necessarily occur immediately after the completion of the instruction enabling the CPU. 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. In EC mode, the channel and device causing the interruption are identified by the I/O address which is stored at locations 186-187. In BC mode, the channel and device causing the interruption are identified by the I/O address in bit positions 16-31 of the I/O old PSW.

If a limited-channel logout is present, it is stored at locations 176-179.

Subsequently, a new PSW is loaded from location 120, and processing resumes in the state indicated by this PSW. The CSW associated with the interruption identifies the interruption condition responsible for the interruption and provides further details about the progress of the operation and the status of the device.

Programming Note

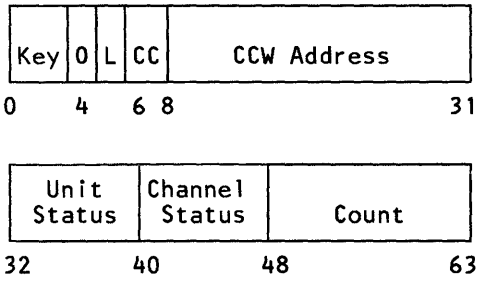
When a number of I/O devices on a shared control unit are concurrently executing operations such as rewinding tape or positioning a disk-access mechanism, the initial device-end signals generated on completion of the operations are provided in the order of generation, unless command chaining is specified 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 because the CPU is disabled for I/O interruptions 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 indication of the reasons for which an I/O operation has been concluded. The CSW is formed, or parts of it are replaced, in the process of I/O interruptions and possibly during the execution of START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, HALT DEVICE, and STORE CHANNEL ID. The CSW is stored 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 contents to be replaced, whichever occurs first.

The information placed in the CSW by an I/O interruption pertains to the device which is identified by the I/O address stored during the interruption. The information placed in the CSW by START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, or HALT DEVICE pertains to the device addressed by the instruction.

The CSW has the following format:



The fields in the CSW are allocated as follows:

Subchannel Key: Bits 0-3 form the access key used in the chain of operations at the subchannel.

Logout Pending (L): Bit 5, when one, indicates that an I/O instruction cannot be executed until a logout has been cleared. Bit 45, channel-control check, will always be one when bit 5 is one.

Deferred Condition Code (CC): Bits 6 and 7 indicate whether situations have been encountered subsequent to the setting of a condition code 0 for START I/O FAST RELEASE that would have caused a different condition-code setting for START I/O. The possible setting of these bits, and their meanings, are as follows:

Setting of		Meaning
Bit 6	Bit 7	
0	0	Normal I/O interruption
0	1	Deferred condition code is 1
1	0	(Reserved)
1	1	Deferred condition code is 3

CCW Address: Bits 8-31 form an address that is 8 higher than the address of the last CCW used.

Status: Bits 32-47 identify the status of the device and the channel that caused the storing of the CSW. Bits 32-39, the unit status, indicate situations detected by the device or control unit. Bits 40-47, the channel status, are provided by the channel and indicate situations associated with the subchannel. The 16 bits are designated as follows:

Bit	Designation
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

Count: Bits 48-63 form the residual count for the last CCW used.

Unit Status

The following status indications are generated by the I/O device or control unit. The timing and causes of these status indications for each type of device are specified in the SL 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 status 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 situation and is specified in the SL publication for the device.

Attention

Attention is signaled when the device detects an asynchronous situation that is significant to the program. Attention is interpreted by the program and is not associated with the initiation, execution, or conclusion of an I/O operation.

The device can signal attention to the channel when no operation is in progress at the I/O device, control unit, or subchannel. Attention can be signaled with device end upon completion of an operation, and it can be signaled to the channel during the initiation of a new I/O operation. Attention along with device end and unit exception can also be signaled whenever a device changes from the not-ready to the ready state. The handling and presentation of attention 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 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, when the control unit is busy, when the normal sequence of commands has to be modified, or when command retry is to be initiated.

When status modifier is signaled in response to TEST I/O and status modifier is the only status bit that is set to one, this 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 subchannel-key, CCW-address, and count fields.

When the status-modifier bit in the CSW is set to one together with the busy bit, it indicates that the busy status 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, HALT I/O, and HALT DEVICE or when it contains an interruption condition for a device other than the one addressed. The interruption condition may be due to control-unit end, due to channel end following the execution of CLEAR I/O, or, on the selector channel, due to channel end following the execution of HALT I/O or HALT DEVICE. The busy state occurs for operations such as backspace file, in which case the control unit remains busy after providing channel end, for operations concluded by CLEAR I/O, and for operations concluded on the selector channel by HALT I/O or HALT DEVICE, 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 status modifier and device end means that the normal sequence of commands must be modified. The handling of this status combination by the channel depends on the operation. If command chaining is specified in the current CCW and no unusual situations have been detected, presence of status modifier and device end causes the channel to fetch and chain to the CCW whose storage address is 16 higher than that of the current CCW. If the I/O device signals status modifier at a time when no command chaining is specified, or when any unusual situations have been detected, no action is taken in the channel, and the status-modifier bit and any other status bit presented by the device are set to ones in the CSW.

Status modifier is set to one in combination with unit check and channel end to initiate the command-retry procedure.

Control-Unit End

Control-unit end indicates that the control unit has become available for use for another operation.

Control-unit end is provided only by control units shared by I/O devices or control units accessible by two or more channels, and only when one or both of the following have 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, HALT I/O, or HALT DEVICE 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 the section "Status Modifier" earlier in this chapter.

2. The control unit detected an unusual situation during the portion of the operation after channel end had been signaled to the channel. The indication of the unusual situation accompanies control-unit end.

If the control unit remains busy with the execution of an operation after signaling channel end but has not detected any unusual situations and 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.

Control-unit end can be signaled with channel end, with 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, the interruption may be identified by any address assigned to the control unit. A control-unit end may cause the control unit to appear busy for the initiation of new operations with any attached device. Alternatively, a control-unit end may be assigned by the control unit to a specific device address, and only that device would appear busy for the initiation of new operations.

Busy

Busy indicates that the I/O device or control unit cannot execute the command or instruction because (1) it is executing a previously initiated operation, (2) it contains an interruption condition, (3) it is shared by channels or I/O devices and the shared facility is not available, or (4) a self-initiated function is being performed. The status associated with the interruption condition for the addressed device, if any, accompanies the busy status. If busy applies to the control unit, busy is accompanied by status modifier.

The figure "Indications of Busy in CSW" lists the situations for devices connected to only one channel when the busy bit is set to one in the CSW and when busy is accompanied by status modifier. For devices shared by more than one channel, operations related to one channel may cause the control unit or device to appear busy to the other channels.

Condition	CSW Status Stored by				
	SIO or SIOF≠	TIO	CLRIO+	HIO or HDV	I/O IRPT#
Subchannel available					
DE or attention in device	B,cl	NB,cl	*	*	NB,cl
Device working, CU available	B	B	*	*	B
CU end or channel end in CU:					
for the addressed device	B,cl	NB,cl	NB	*	NB,cl
for another device	B,SM	B,SM	NB	*	NB,cl
CU working	B,SM	B,SM	NB	*	B,SM
Interruption condition in subchannel for the addressed device because of:					
chaining terminated by busy	*	B,cl	NB,cl	*	B,cl
other type of termination	*	NB,cl	NB,cl	*	NB,cl
Subchannel working					
CU available	*	*	NB	NB	*
CU working	*	*	NB	B,SM	*

Explanation:

B Busy bit in CSW is one.

cl Interruption condition cleared; status is placed in CSW.

CU Control unit.

DE Device end.

NB Busy bit is zero.

SM Status-modifier bit appears in CSW.

* CSW not stored, or I/O interruption cannot occur.

≠ When a channel executes START I/O FAST RELEASE as START I/O, the CSW status stored for the two instructions is identical. When START I/O FAST RELEASE is executed independently of the device, the same status is stored by an I/O interruption with the CSW also indicating deferred condition code 1.

Except when the I/O interruption is caused by a deferred condition code 1 for START I/O FAST RELEASE.

+ The entries in this column apply only when the CLRIO function is executed. When CLEAR I/O causes the TIO function to be executed, the entries in the TIO column apply.

Indications of Busy in CSW

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 channel end to be signaled, and there is only one channel end for an operation. Channel end is not signaled 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. Channel end is not made available to the program when a chain of commands is prematurely concluded because of an unusual situation 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 signaled depends on the operation

and the type of device. For operations such as writing on magnetic tape, channel end 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, channel end occurs when the gap on tape reaches the read-write head. On devices equipped with buffers, channel end 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 channel end may be delayed until completion of the operation. Operations that do not cause any data to be transferred can provide channel end during the initiation sequence.

Channel end in the control unit may cause the control unit to appear busy for the initiation of new operations.

Channel end is presented in combination with status modifier and unit check to initiate the command-retry procedure.

Device End

Device end is caused by the completion of an I/O operation at the device, by manually changing the device from the not-ready to the ready state, or by the termination of an activity which previously caused a response of busy to the channel. Device end normally indicates that the I/O device has become available for use in another operation.

Each I/O operation causes device end, and there is only one device end to an operation. Device end 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 situation is detected during the initiation of a chained command, in which case the chain is concluded without device end.

Device end associated with an I/O operation is generated either simultaneously with channel end or later. For data-transfer operations on devices such as magnetic-tape units, the device concludes the operation at the time channel end is generated, and both device end and channel end occur together. On buffered devices, device end 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, receipt of the device-end signal, in the absence of any unusual situations, causes the channel to initiate a new I/O operation.

When the state of a device is changed from not ready to ready, a device end is generated. Some devices generate attention and unit exception along with device end when they change from the not-ready to ready state. A device is considered to be not-ready when operator intervention is required in order to make the device available. A not-ready condition can occur, for example, because of any of the following:

1. An unloaded condition for magnetic tape
2. Card equipment out of cards or with the stacker full
3. A printer out of paper
4. Error conditions that need operator intervention
5. The unit having changed from the enabled to the disabled state

Unit Check

Unit check indicates that the I/O device or control unit has detected an unusual situation that is detailed by the information available to a sense command. Unit check may indicate that a programming or 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 situation other than the one identified by unit exception has occurred. The unit-check bit provides a summary indication of the sense data.

An error 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 pertains to the activity initiated by a command and is of immediate significance to the program, the error 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. Selection of a device that is not ready does not cause a unit check when the sense command is issued or when 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 signaled to the channel 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 situation 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 situation 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 an interruption condition at the time of selection. Under these circumstances, the device responds by providing busy status and indicating the interruption condition, if any. The command-code invalidity is not indicated.

Concluding an operation with the unit-check indication causes command chaining to be suppressed.

Unit check is presented in combination with channel end and status modifier to initiate the command-retry procedure.

Programming Notes

1. 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.
2. In order that sense indications set in conjunction with unit check are preserved by the device until requested by a sense command, some devices inhibit certain functions until a command other than test I/O or no-operation is received. Furthermore, any command other than sense, test I/O, or no-operation causes the device to reset any sense information. To avoid degradation of the device and its control unit and to avoid inadvertent resetting of the sense information, a sense command should be issued immediately to any device signaling unit check.

Unit Exception

Unit exception is caused when the I/O device detects a situation that usually does not occur. Unit exception includes situations 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.

Unit exception can be generated only when the device is executing an I/O operation, or when the device is involved with some activity associated with an I/O operation and the situation 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 situation 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 situation was detected. Any unusual situation 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 with busy status to the command during the initial selection.

Concluding an operation with the unit-exception indication causes command chaining to be suppressed.

Unit exception along with device end and attention can also be generated whenever a device changes from the not-ready state to the ready state.

Channel Status

The following status bits are generated by the channel. Except for the status bits resulting from equipment malfunction, they can occur only while the subchannel is involved with the execution of an I/O operation.

Program-Controlled Interruption

A program-controlled interruption occurs when the channel fetches a CCW with the program-

controlled-interruption (PCI) flag set to one. The I/O 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 I/O interruptions are disallowed or because of other activity in the system.

The interruption condition due to the PCI flag 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 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 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.

Incorrect length is not indicated when the current CCW has the SLI flag set to one and the CD flag set to zero. The indication does not occur for immediate operations and for operations rejected during the initiation sequence.

When incorrect length occurs, command chaining is suppressed, unless the SLI flag in the CCW is one or unless the operation is immediate. See the figure "Channel-Chaining Action" in this chapter for the effect of the CD, CC, and SLI flags on the indication of incorrect length.

Programming Note

The setting of incorrect length is unpredictable in the CSW stored during CLEAR I/O.

Program Check

Program check occurs when programming errors are detected by the channel. Program check 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 doublewords. The three rightmost bits of the CCW address are not zeros.

CCW Location Not Provided: The channel has attempted to fetch a CCW from a storage location that is not provided. This may occur because the program has specified in the CAW or in the transfer-in-channel command a page address (bits 8-20) equal to or greater than the page-capacity count (PCC), or because on chaining the channel has attempted to fetch a CCW from a page with a page address equal to PCC.

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.

Data Location Not Provided: The channel has attempted to transfer data to or from a storage location that is not provided. This may occur because the program has specified in the CCW a page address (bits 8-20) equal to or greater than the page-capacity count (PCC) or because the channel attempts during data transfer to access a page with a page address equal to PCC.

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 CCWs both of which specify transfer in channel.

Detection of program check during the initiation of an operation causes execution of the operation to be suppressed. When program check is detected after the device has been started, the device is

signaled to conclude the operation the next time it requests or offers a byte of data. Program check causes command chaining to be suppressed.

Protection Check

Protection check occurs when the channel attempts a storage access that is prohibited by key-controlled storage protection. Protection applies to the fetching of CCWs and output data, and to the storing of input data. Storage accesses associated with each channel program are performed using the subchannel key provided in the CAW associated with that channel program. For details, see the section "Key-Controlled Protection" in Chapter 3, "Storage."

Protection check also occurs when it is detected that the channel has attempted to access a CCW or data from a page that is in the disconnected state. For details, see the section "Page States" in Chapter 3, "Storage."

When protection check 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 conclude the operation the next time it requests or offers a byte of data. Protection check causes command chaining to be suppressed.

Channel-Data Check

Channel-data check indicates that a machine error has been detected in the information transferred to or from storage during an I/O operation, or that a parity error has been detected on the data on bus-in during an input 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 in the channel, in storage, or on the path between the two. Channel-data check may be indicated for data with an invalid checking-block code in storage when the data is referred to by the channel but the data does not participate in the operation.

Whenever a parity error on I/O input data is indicated by means of channel-data check, the channel forces correct parity on all data received from the I/O device, and all data placed in storage has valid checking-block code. When, on an input operation, the channel attempts to store less than a complete checking block, and when invalid checking-block code is detected on the checking block in storage, the contents of the location remain unchanged with invalid checking-block code. On an output operation, whenever a channel-data check is indicated, all bytes that came from a checking block with invalid checking-block code have been transmitted with parity errors.

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, if possible, and an interruption condition is generated when the device presents channel end. A logout may be performed, depending on the channel. Accordingly, the detection of the error may affect the state of the channel and the device.

Channel-Control Check

Channel-control check is caused by machine malfunction affecting channel controls. It may be caused by invalid checking-block code on CCW and data addresses and invalid checking-block code on the contents of the CCW. Channel-control check may also include those channel-detected errors associated with data transfer that are not indicated as channel-data check, as well as those I/O interface errors detected by the channel that are not indicated as interface-control check. Errors 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 valid checking-block code.

Detection of channel-control check causes the current operation, if any, to be immediately concluded.

Channel-control check is set whenever CSW bit 5, logout pending, is set to one.

In some situations, machine malfunctions affecting channel control may instead be reported as an external-damage or system-damage machine-check condition.

Interface-Control Check

Interface-control check indicates that an invalid signal has been received by the channel when communicating with a control unit or device. This check is detected by the channel and usually indicates malfunctioning of an I/O device. It can be due to the following:

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.

5. A device signaled I/O error alert.

The interface-control-check condition may also include those channel-detected errors associated with bus-in during data transfer that are not indicated as channel-data check.

Detection of interface-control check causes the current operation, if any, to be immediately concluded.

Chaining Check

Chaining check is caused by channel overrun during data chaining on input operations. Chaining check occurs when the I/O data rate is too high to be handled by the channel and by storage under current conditions. Chaining check cannot occur on output operations.

Chaining check causes the I/O device to be signaled to conclude the operation. It causes command chaining to be suppressed.

Contents of Channel-Status Word

The contents of the CSW depend on the reason the CSW was stored and on the programming method by which the information is obtained. The status portion always identifies the reason the CSW was stored. The subchannel-key, CCW-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

Interruption conditions resulting from the execution or conclusion 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 or CLEAR I/O. Except for situations associated with command chaining and equipment malfunctioning, the storing can be caused by PCI or channel end and by the execution of HALT I/O or HALT DEVICE on the selector channel. The contents of the CSW are related to the current values of the corresponding quantities, although the count is unpredictable after program check, protection check, and chaining check, and after an interruption due to the PCI flag.

A CSW stored upon the execution of a chain of operations pertains to the last operation which 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 situation causes command chaining to be suppressed, the premature conclusion of the chain is not explicitly indicated in the CSW. A CSW associated with a conclusion due to a situation occurring at channel-end time contains channel end and identifies the unusual situation. When the device signals the unusual situation with control-unit end or device end, the channel-end indication is not made available to the program, and the channel provides the current subchannel key, CCW address, and count, as well as the unusual indication, with control-unit end or device end in the CSW. The CCW-address and count fields pertain to the operation that was executed.

When the execution of a chain of commands is concluded by an unusual situation detected during initiation of a new operation, the CCW-address and count fields pertain to the rejected command. Except for situations resulting from equipment malfunctioning, conclusion at 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 set to zeros.

A CSW associated with status signaled after the operation at the subchannel has been concluded contains zeros in the subchannel-key, CCW-address, and count fields, provided the status is not cleared during START I/O or START I/O FAST RELEASE and provided logout pending is not indicated. This status includes attention, control-unit end, and device end (and channel end when it occurs after the conclusion of an operation on the selector channel by HALT I/O or HALT DEVICE).

When the above status indications, other than logout pending, are cleared during START I/O or START I/O FAST RELEASE, only the status portion of the CSW is stored, and the original contents of the subchannel-key, CCW-address, deferred-condition-code, logout-pending, 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 concluded during the execution of START I/O or START I/O FAST RELEASE or whenever HALT I/O or HALT DEVICE causes CSW status to be stored.

Errors detected during execution of the I/O operation do not affect the validity of the CSW unless channel-control check or interface-control check are indicated. Channel-control check indicates that equipment errors have been detected which can cause any part of the CSW, as well as the I/O address, 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. The validity of these fields can be ascertained by inspecting the limited channel logout.

When any I/O instruction cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones.

Subchannel Key

A CSW stored to reflect the progress of an operation at the subchannel contains the subchannel key used in that operation. The contents of this field are not affected by programming errors detected by the channel or by the situations causing termination of the operation.

CCW Address

When the CSW is formed to reflect the progress of the I/O operation at the subchannel, the CCW address is normally 8 higher than the address of the last CCW used in the operation.

The figure "Contents of the CCW-Address Field in the CSW" lists the contents of the CCW-address field for all situations that can cause the CSW to be stored. They are listed in order of priority; that is, if two situations occur, the CSW appears as indicated for the situation higher on the list. When a CSW has been stored and the situation exists that a command-retry request has been recognized but the CCW has not been re-executed, the "last-used CCW + 8" is the CCW that is to be retried. When a program check is caused by two CCWs in sequence both of which specify transfer in channel, the second CCW is the one considered invalid. In the figure, the three cases of disconnected location and the two cases of invalid key are all protection checks.

Situations	Contents of Field
Channel-control check	Unpredictable
Status stored by START I/O or START I/O FAST RELEASE	Unchanged
Status stored by HALT I/O or HALT DEVICE	Unchanged
Program check because CCW location in TIC not provided	Address of TIC + 8
Program check (all others)	Address of invalid CCW + 8
Disconnected CCW location in TIC	Address of TIC + 8
Disconnected CCW location generated	First invalid CCW address + 8
Disconnected data location	Address of invalid CCW + 8
Invalid key on CCW fetch	Address of protected CCW + 8
Invalid key on data access	Address of current 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
Termination by CLEAR 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
Deferred condition code 1 or 3 for START I/O FAST RELEASE	Address of CCW specifying the new operation + 8
PCI flag in CCW	Address of last-used CCW + 8
Interface-control check	Unpredictable
Channel end after HALT I/O on selector channel	Zero
Channel end after CLEAR I/O	Zero
Control-unit end	Zero
Device end	Zero
Attention	Zero
Busy	Zero
Status modifier	Zero

Contents of the CCW-Address Field in the CSW

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 concluded, 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 storage; on an output operation, the difference is equal to the number of bytes transferred to the I/O device.

The figure "Contents of the Count Field in the CSW" lists the contents of the count field for all situations that can cause the CSW to be stored. They are listed in the order of priority; that is, if two situations occur, the CSW appears as for the situation higher on the list.

Status

The status bits identify the situations 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 errors, all corresponding status bits in the CSW may be set to ones or only one may be set, depending on the error and model. Errors 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 error. When an operation is concluded by program check, protection check, or chaining check, the channel identifies the situation responsible for the

Situations	Contents of Field
Channel-control check	Unpredictable
Status stored by START I/O or START I/O FAST RELEASE	Unchanged
Status stored by HALT I/O or HALT DEVICE	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 or HALT DEVICE	Unpredictable
Termination by CLEAR I/O	Unpredictable
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.
Deferred condition code 1 or 3 for START I/O FAST RELEASE	Correct. Original count of CCW specifying the new operation.
PCI flag in CCW	Unpredictable
Interface-control check	Unpredictable
Channel end after HALT I/O on selector channel	Zero
Channel end after CLEAR I/O	Zero
Control-unit end	Zero
Device end	Zero
Attention	Zero
Busy	Zero
Status modifier	Zero

Contents of the Count Field in the CSW

conclusion and may or may not indicate incorrect length. When a data error has been detected and the operation is concluded 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 has the PCI flag set to one but a programming error in the contents of the CCW precludes the initiation of the operation, it is unpredictable whether the PCI bit is one in the CSW associated with the interruption condition. Similarly, if a programming error in the contents of the CCW causes the command to be rejected during execution of START I/O or START I/O FAST RELEASE, the CSW stored by the instruction may or may not have the PCI bit set to one. Furthermore, when the channel detects a programming error in the CAW or in the first CCW, the PCI bit is unpredictable in a CSW stored by START I/O or START I/O FAST RELEASE even when the PCI flag is zero in the first CCW associated with the instruction.

However, if the CCW fetched on command chaining has the PCI flag set to one but an unusual situation detected by the device precludes the initiation of the operation, the PCI bit is one in the CSW associated with the interruption condition. Likewise, if device status causes the command to be rejected during execution of START I/O or START I/O FAST RELEASE, the CSW stored by the instruction contains the PCI bit set to one.

Situations detected by the channel are not related to those identified by the I/O device.

The figure "Contents of the CSW Status Fields" summarizes the handling of status bits. The figure 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 Subch is Working	Upon Termination of Operation at			During Cmd Chaining	By SIO or SIOF	By TIO	By CLRIO +	By HIO or HDV	By I/O Interruption
			Subch	Ctrl Unit	I/O Dev						
Attention	C*				C*	C*	S	S	S		S
Status modifier					C	C	CS	CS	S	CS	S
Control-unit end				C*			CS	CS	S	CS	S
Busy						C	CS	CS	S	CS	S
Channel end			C*	C*H		C*≠	CS≠	S	S		S
Device end	C*				C*	C ≠	CS≠	S	S		S
Unit check	C		C	C	C	C*	CS	CS	S		CS
Unit exception			C	C	C	C*	CS	S	S		S
Program-controlled interruption		C*	C*			C	CS	S	S		S
Incorrect length		C	C					S	S		S
Program check		C	C			C*	CS	S	S		S
Protection check		C	C			C*	CS	S	S		S
Channel-data check		C	C					S	S		S
Channel-control check	C*	C*	C*	C*	C*	C*	CS	CS	CS	CS	CS
Interface-control check	C*	C*	C*	C*	C*	C*	CS	CS	CS	CS	CS
Chaining check		C	C					S	S		S
Deferred cond code 1							C*#	S	S		S
Deferred cond code 3							C*#	S	S		S

Explanation:

C The channel or device can create or present status at the indicated time. A CSW or its status portion is not necessarily stored at this time.

Status such as channel end or device end is created at the indicated time. Other status bits may have been created previously but are made accessible to the program only at the indicated time. Examples of such status bits are program check and channel-data check, which are detected while data is transferred but are made available to the program only with channel end, unless the PCI flag or an equipment malfunction has 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 status has been created previously. The letter C appearing with the S indicates that the status 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, an equipment malfunction may be detected during an I/O interruption, causing channel-control or interface-control check to be indicated; or a device such as the 2702 may signal control-unit busy 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 generates an interruption condition.

Channel end and device end do not result in interruption conditions when command chaining is specified and no unusual situations have been detected.

≠ This indication is created at the indicated time only by an immediate operation.

Applies only to SIOF.

H When an operation on the selector channel has been concluded by HALT DEVICE or HALT I/O, or an operation has been concluded by CLEAR I/O, channel end indicates the conclusion of the data-handling portion of the operation at the control unit.

+ The entries in this column apply only when the CLRIO function is executed. When CLEAR I/O causes the TIO function to be executed, the entries in the TIO column apply.

Contents of the CSW Status Fields

Channel Logout

When a channel stores a CSW that indicates channel-control check in the absence of logout pending, or interface-control check, or, on some channels, channel-data check, a limited channel logout accompanies the storing of the CSW. Such a logout is useful for error recovery.

The limited channel logout contains model-independent information and is stored at locations 176-179. When it is stored, bit 0 of the logout is always stored as a zero.

I/O-Communication Area

Storage locations 160-191 comprise a permanently assigned area of storage used for I/O, designated the I/O-communication area (IOCA). (See the figure "I/O-Communication Area.")

Locations 160-167, 172-175, 180-184, and 188-191 are reserved for future I/O use.

Channel ID (Locations 168-171): Locations 168-171, when stored during the execution of a STORE CHANNEL ID instruction, contain information which describes the addressed channel.

Limited Channel Logout (Locations 176-179): The limited-channel-logout field (locations 176-179) contains model-independent information related to equipment errors detected by the channel. This information is used to provide detailed machine status when errors have affected I/O operations. The field may be stored only when the CSW or a portion of the CSW is stored.

The bits of the field are defined as follows:

0 This bit is always stored as a zero when a limited channel logout is stored. If the program ensures that this bit is set to one and any channel-control check, interface-control check, or channel-data check occurs, a test of this bit can determine if the LCL was stored by the channel. The LCL cannot be stored by a channel unless one of these three channel-status bits is set to one.

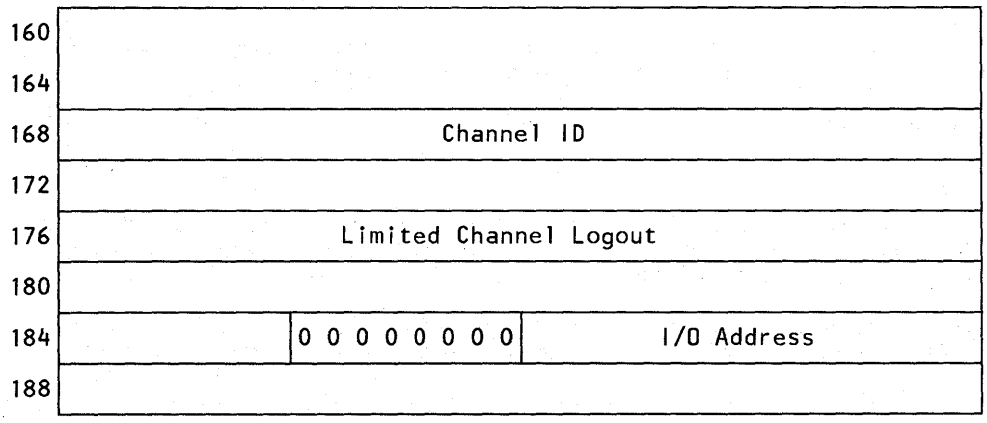
1-3 *Identity of the storage-control unit (SCU)* identifies the SCU through which storage references were directed when an error was detected. This identity is not necessarily the identity of the storage unit involved with the transfer. When only one physical path exists between channel and storage, the storage-control unit has the identity of the CPU. If more than one path exists, the storage-control unit has its own identity.

When bit 3 is zero, bits 1 and 2 are undefined. In this case, the SCU identity is implied to be the same as the CPU identity. When bit 3 is one, the binary value of bits 1 and 2 identifies a physical SCU. Each SCU in the system has a unique identity.

4-7 *Detect field* identifies the type of unit that detected the error. At least one bit is present in this field, and multiple bits may be set when more than one unit detects the error.

Bit 4–CPU
Bit 5–Channel
Bit 6–Main-storage control
Bit 7–Main storage

8-12 *Source field* indicates the most likely source of the error. The determination is made by the channel on the basis of the type of error check, the location of



I/O-Communication Area

the checking station, the information flow path, and the success or failure of transmission through previous check stations.

Normally, only one bit will be present in this field. However, when interunit communication cannot be resolved to a single unit, such as when the interface between units is at fault, multiple bits (normally two) may be set to ones in this field. When a reasonable determination cannot be made, all bits in this field are set to zeros.

If the detect and source fields indicate different units, the interface between them can also be considered suspect.

Bit 8– CPU
Bit 9– Channel
Bit 10–Main-storage control
Bit 11–Main storage
Bit 12–Control unit

13-18 *Reserved.* Stored zero.

19-23 *Field-validity flags.* These bits indicate the validity of the information stored in the designated fields. When the validity bit is set to one, the field is stored and usable. When the validity bit is set to zero, the field is not usable.

The fields designated are:

Bit 19–Sequence code
Bit 20–Unit status
Bit 21–CCW address and key in CSW
Bit 22–Channel address
Bit 23–Device address

24-25 *Type of termination* that has occurred is indicated by these two bits.

This encoded field has meaning only when a channel-control check or an interface-control check is indicated in the CSW. When neither of these two checks is indicated, no termination has been forced by the channel.

00 Interface disconnect
01 Stop, stack, or normal termination
10 Selective reset
11 Reserved

26-27 *Reserved.* Stored zero.

28 *I/O-error alert.* This bit, when set to one, indicates that the limited channel logout resulted from the signaling of I/O-error alert by the indicated unit.

The I/O-error-alert signal indicates that the control unit has detected a malfunction which prevents it from communicating properly with the channel. The channel, in response, performs a malfunction reset and causes interface-control check to be set.

29-31 *Sequence code* identifies the I/O sequence in progress at the time of error. It is meaningless if stored during the execution of HALT I/O or HALT DEVICE.

For all cases, the CCW address in the CSW, if validly stored and nonzero, is the address of the current CCW plus 8.

The sequence code assignments are:

000 A channel-detected error occurred during the execution of a TEST I/O or CLEAR I/O instruction.

001 Command-out with a nonzero command byte on bus-out has been sent by the channel, but device status has not yet been analyzed by the channel. This code is set with a command-out response to address-in during initial selection.

010 The command has been accepted by the device, but no data has been transferred. This code is set by a service-out or command-out response to status-in during an initial selection sequence, if the status is either channel end alone, or channel end and device end, or channel end, device end, and status modifier, or all zeros.

011 At least one byte of data has been transferred between the channel and the device. This code is set with a service-out response to service-in and, when appropriate, may be used when the channel is in an idle or polling state.

100 The command in the current CCW has either not yet been sent to the device or else was sent but not accepted by the device. This code is set when one of the following situations occurs:

1. When the CCW address is updated during command chaining or a START I/O.
2. When service-out or command-out is raised in

response to status-in during an initial selection sequence with the status on bus-in including attention, control-unit end, unit check, unit exception, busy, status modifier (without channel end and device end), or device end (without channel end).

3. When a short, control-unit-busy sequence is signaled.
4. When command retry is signaled.
5. When the channel issues a test-I/O command rather than the command in the current CCW.

101 The command has been accepted, but data transfer is unpredictable. This code applies from the time a device comes on the interface until the time it is determined that a new sequence code applies. The code may thus be used when a channel goes into the polling or idle state and it is impossible to determine that code 010 or 011 applies. The code may also be used at other times when a channel cannot distinguish between code 010 or 011.

110 *Reserved.*

111 *Reserved.*

Reserved (Location 185): Zero is stored at location 185 whenever an I/O address is stored at locations 186-187.

I/O Address (Locations 186-187): A two-byte field is provided for storing the I/O address on each I/O interruption in the EC mode.

Chapter 13. Operator Facilities

Contents

Manual Operation

The operator facilities provide functions for the manual operation and control of the machine. The functions include operator-to-machine communication, indication of machine status, control over the setting of the time-of-day clock, initial program loading, resets, and other manual controls for operator intervention in normal machine operation.

A model may provide additional operator facilities which are not described in this chapter. Examples are the means to indicate specific error conditions in the equipment, to change equipment configurations, and to facilitate maintenance. Furthermore, controls covered in this chapter may have additional settings which are not described here. Such additional facilities and settings are contained in the appropriate System Library (SL) publication.

Most models provide, in association with the operator facilities, a console device which may be used as an I/O device for operator communication with the program; this console device may also be used to implement some or all of the facilities described in this chapter.

The operator facilities may be implemented on different models in various technologies and configurations. On some models, more than one set of physical representations of some keys, controls, and indicators may be provided, such as on multiple local or remote operating stations, which may be effective concurrently.

A machine malfunction that prevents a manual operation from being performed correctly, as defined for that operation, may cause the CPU to enter the check-stop state or give some other indication to the operator that the operation has failed. Alternatively, a machine malfunction may cause a machine-check-interruption condition to be recognized.

Basic Operator Facilities

Address-Compare Controls

The address-compare controls provide a way to stop the CPU when a preset address matches the address used in a specified type of main-storage reference.

One of the address-compare controls is used to set up the address to be compared with the storage address.

Another control provides at least two settings to specify the action, if any, to be taken when the address match occurs. The two settings are normal and stop. When this control is set to stop, the test indicator is turned on.

1. The normal setting disables the address-compare operation.
2. The stop setting causes the CPU to enter the stopped state on an address match. Depending on the model and the type of reference, pending I/O, external, and machine-check interruptions may or may not be taken before entering the stopped state.

A third control may specify the type of storage reference for which the address comparison is to be made. A model may provide one or more of the following settings, as well as others:

1. The any setting causes the address comparison to be performed on all storage references.
2. The data-store setting causes address comparison to be performed when storage is addressed to store data.
3. The I/O setting causes address comparison to be performed when storage is addressed by a channel to transfer data or to fetch a channel-command word. Whether references to the channel-address word or the channel-status word cause a match to be indicated depends on the model.
4. The instruction-address setting causes address comparison to be performed when storage is addressed to fetch an instruction. The rightmost bit of the address setting may or may not be ignored. The match is indicated only when the first byte of the instruction is fetched from the selected location. It depends on the model whether a match is indicated when fetching the target instruction of EXECUTE.

Alter-and-Display Controls

The operator facilities provide controls and procedures to permit the operator to alter and display the contents of addressable locations in storage, the storage keys, the page bits, the general, floating-point, and control registers, and the PSW. Information in storage can only be altered or displayed if the storage pages containing the information are in the connected or addressable state.

Before alter-and-display operations may be performed, the CPU must first be placed in the stopped state. During alter-and-display operations, the manual indicator may be turned off temporarily, and the start and restart keys may be inoperative.

Check Control

The check control has at least two settings, stop and normal. If the control is set to stop, the CPU enters the check-stop state when either:

1. A machine-check condition is detected and not corrected
2. A channel check occurs which would cause information to be stored in a channel-logout area at locations 176-179

Whether information is actually stored in assigned storage locations as a result of the machine check or channel check, the indications given for the cause of the stop, and the manner of resuming CPU operation depend on the model.

If the check control is set to normal, the action resulting from the detection of a machine check or channel check is the same as described in Chapter 11, "Machine-Check Handling," or in Chapter 12, "Input/Output Operations," respectively.

The test indicator is on while the check control is set to stop.

Programming Note

Except that recovery from a machine check or a channel check with logout is not possible, the check control permits a System/360 program, which uses assigned storage locations above 128 as ordinary storage, to be run in the BC mode.

Check-Stop Indicator

The check-stop indicator is on when the CPU is in the check-stop state. Reset operations normally cause the CPU to leave the check-stop state and thus turn off the indicator. The manual indicator may also be on in the check-stop state.

IML Controls

The IML controls perform initial microprogram loading (IML). The IML operation selects the ECPS:VSE architectural mode or the System/370 architectural mode of operation.

When the IML operation is completed, the state of the affected CPU, channels, storage, and operator facilities is the same as if a power-on reset had been performed, except that the value and state of the time-of-day clock are not reset.

The IML controls are effective while the power is on.

Interrupt Key

When the interrupt key is activated, an external-interruption condition indicating the interrupt key is generated. (See the section "Interrupt Key" in Chapter 6, "Interruptions.")

The interrupt key is effective when the CPU is in the operating or stopped state. It depends on the model whether the interrupt key is effective when the CPU is in the load state.

Interval-Timer Control

The interval-timer control disables or enables operation of the interval timer. Disabling the interval timer does not affect any other facility.

When the control is set to disable the interval timer, updating of assigned storage locations 80-83 ceases. The contents of locations 80-83 remain at the last value to which they were updated, unless changed by a subsequent store operation. Any already pending interval-timer-interruption condition is kept pending without regard to the state of the external mask, PSW bit 7, and the interval-timer mask, bit 24 of control register 0.

When the control is set to enable the interval timer, updating of locations 80-83 is resumed using the current contents. If an interval-timer-interruption request existed and was kept pending when the interval-timer control was last set to disable, that condition remains pending until the CPU is enabled for the interruption.

The setting to enable the interval timer is considered the normal setting. The test indicator may or may not be turned on when the interval-timer control is set to disable.

Programming Note

Disabling the interval timer allows execution of a program which uses locations 80-83 as ordinary storage. A program which does not use the interval timer will function correctly with the interval timer disabled, even when the interval timer fails.

Load Indicator

The load indicator is on during initial program loading, indicating that the CPU is in the load state. The indicator goes on when the load-clear or load-normal key is activated and the corresponding operation is started. It goes off after the new PSW is loaded successfully.

Load-Clear Key

Activating the load-clear key causes a clear-reset operation to be performed and initial program loading to be started using the I/O device specified by the load-unit-address controls. For details, see the sections "Resets" and "Initial Program Loading" in Chapter 4, "Control."

The load-clear key is effective when the CPU is in the operating, stopped, load, or check-stop state.

Load-Normal Key

Activating the load-normal key causes an initial-program-reset operation to be performed and initial program loading to be started using the I/O device specified by the load-unit-address controls. For details, see the sections "Resets" and "Initial Program Loading" in Chapter 4, "Control."

The load-normal key is effective when the CPU is in the operating, stopped, load, or check-stop state.

Load-Unit-Address Controls

The load-unit-address controls select three hexadecimal digits, which provide the 12 rightmost I/O address bits used for initial program loading.

Machine-Save Key

Activating the machine-save key initiates a machine-save operation. (See the section "Machine Save" in Chapter 4, "Control.") The save indicator is turned on when the operation is completed successfully.

The machine-save key is effective only when the CPU is in the stopped state.

Operation Note

The machine-save operation may be used in conjunction with a standalone dump program for the analysis of major program malfunctions. For such an operation, the following sequence would be called for:

1. Activation of the stop or system-reset-normal key
2. Activation of the machine-save key
3. Activation of the load-normal key to enter a standalone dump program

The system-reset-normal key must be activated in step 1 when the stop key is not effective because a continuous string of interruptions occurs or the CPU is in the check-stop state.

Manual Indicator

The manual indicator is on when the CPU is in the stopped state. Some functions and several manual controls are effective only when the CPU is in the stopped state.

Mode Indicator

The mode indicator shows the architectural mode of operation selected by the last IML operation.

Power Controls

The power controls are used to turn the power on and off.

The CPU, storage, channels, operator facilities, and I/O devices may all have their power turned on and off by common controls, or they may have separate power controls. When a particular unit has its power turned on, that unit is reset. The sequence is performed so that no instructions or I/O operations are performed until explicitly specified. The controls may also permit power to be turned on in stages, but the machine does not become operational until power-on is complete.

When the power is completely turned on, an IML operation is performed. A power-on reset is then initiated (see the section "Resets" in Chapter 4, "Control"). It depends on the model whether the architectural mode of operation can be selected when the power is turned on, or whether the IML controls have to be used to change mode after the power is on.

Rate Control

The setting of the rate control determines the effect of the start function and the manner in which instructions are executed.

The rate control has at least two settings. The normal setting is process. When the rate control is set to process and the start function is performed, the CPU starts operating at normal speed. When the rate control is set to instruction step, one instruction or, for interruptible instructions, one unit of operation is executed each time the start function is performed. For details, see the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control."

The test indicator is on while the rate control is not set to process.

If the setting of the rate control is changed while the CPU is in the operating or load state, the results are unpredictable.

Restart Key

Activating the restart key initiates a restart interruption. (See the section "Restart Interruption" in Chapter 6, "Interruptions.")

The restart key is effective when the CPU is in the operating or stopped state. The key is not effective when the CPU is in the check-stop state. It depends on the model whether the restart key is effective when the CPU is in the load state.

Save Indicator

The save indicator is turned on when a machine-save operation has been successfully completed. It is turned off when the load-clear, load-normal, restart, start, system-reset-clear, or system-reset-normal key is activated. It may also be turned off when other controls are activated. The indicator is off after a power-on reset. If an error is encountered during the machine-save operation, the indicator remains off.

Start Key

Activating the start key causes the CPU to perform the start function. (See the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control.")

The start key is effective only when the CPU is in the stopped state. The effect is unpredictable when the stopped state has been entered by a reset.

Stop Key

Activating the stop key causes the CPU to perform the stop function. (See the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control.")

The stop key is effective only when the CPU is in the operating state.

Operation Note

Activating the stop key has no effect when:

- An unending string of certain program or external interruptions occurs.
- The CPU is in the load or check-stop state.

Storage-Size Control

The storage-size control is provided when a model permits more than one size of virtual storage. The control determines the storage size and, hence, the value of the page-capacity count. The number of storage-size settings of the control depends on the model. (See the section "Storage Size" in Chapter 3, "Storage.")

A new setting of the storage-size control becomes effective only as part of the IML operation performed when turning the power on or when activating the IML controls.

System-Reset-Clear Key

Activating the system-reset-clear key causes a clear-reset operation to be performed. For details, see the section "Resets" in Chapter 4, "Control."

The system-reset-clear key is effective when the CPU is in the operating, stopped, load, or check-stop state.

System-Reset-Normal Key

Activating the system-reset-normal key causes a program-reset operation to be performed. For details, see the section "Resets" in Chapter 4, "Control."

The system-reset-normal key is effective when the CPU is in the operating, stopped, load, or check-stop state.

Test Indicator

The test indicator is on when a manual control for operation or maintenance is in an abnormal position that can affect the normal operation of a program.

Setting the address-compare controls or the check control to stop or setting the rate control to instruction step turns on the test indicator. Setting the interval-timer control to disable may or may not turn on the test indicator.

The test indicator may be on when one or more diagnostic functions under the control of DIAGNOSE are activated, or when other abnormal conditions occur.

Operation Note

If a manual control is left in a setting intended for maintenance purposes, such an abnormal setting may, among other things, result in false machine-check indications or cause actual machine malfunctions to be ignored. It may also alter other aspects of machine operation, including instruction execution, channel operation, and the functioning of operator controls and indicators, to the extent that operation of the machine does not comply with that described in this publication.

TOD-Clock Control

When the TOD-clock control is not activated, that is, the control is set to secure, the value of the time-of-day (TOD) clock is protected against unauthorized or inadvertent change by not permitting the instruction SET CLOCK to change the value.

When the TOD-clock control is activated, that is, the control is set to enable set, alteration of the clock value by means of SET CLOCK is permitted. This setting is temporary, and the control automatically returns to secure.

Wait Indicator

The wait indicator is on when the wait-state bit in the current PSW is one.

Appendix A. Number Representation and Instruction-Use Examples

Contents

Number Representation

Binary Integers

Signed Binary Integers

Signed binary integers are most commonly represented as halfwords (16 bits) or words (32 bits). In both lengths, the leftmost bit (bit 0) is the sign of the number. The remaining bits (bits 1-15 for halfwords and 1-31 for words) are used to designate the magnitude of the number. Binary integers are also referred to as fixed-point numbers, because the radix point is considered to be fixed at the right, and any scaling is done by the programmer.

Positive binary integers are in true binary notation with a zero sign bit. Negative binary integers are in two's-complement notation with a one bit in the sign position. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit (that is, all zeros for positive numbers, all ones for negative numbers).

Negative binary integers are formed in two's-complement notation by inverting each bit of the positive binary integer and adding one. As an example using the halfword format, the binary number with the decimal value +26 is made negative (–26) in the following manner:

```
+26     0 000 0000 0001 1010
Invert  1 111 1111 1110 0101
Add 1                    1
        ---------------------
-26     1 111 1111 1110 0110 (Two's comple-
                              ment form)
(S is the sign bit.)
```

This is equivalent to subtracting the number:

```
          00000000 00011010
from
        1 00000000 00000000
```

Negative binary integers are changed to positive in the same manner.

The following addition examples illustrate two's-complement arithmetic and overflow conditions. Only eight bit positions are used.

```
1.  +57  =  0011 1001
    +35  =  0010 0011
    ------------------
    +92  =  0101 1100
```

```
2.  +57  =  0011 1001
    -35  =  1101 1101
    ------------------
    +22  =  0001 0110   No overflow-carry
                        into leftmost posi-
                        tion and carry out.
```

```
3.  +35  =  0010 0011
    -57  =  1100 0111
    ------------------
    -22  =  1110 1010   Sign change only-no
                        carry into leftmost
                        position and no
                        carry out.
```

```
4.  -57  =  1100 0111
    -35  =  1101 1101
    ------------------
    -92  =  1010 0100   No overflow-carry
                        into leftmost posi-
                        tion and carry out.
```

```
5.  +57  =  0011 1001
    +92  =  0101 1100
    ------------------
   +149  = *1001 0101   *Overflow-carry
                        into leftmost posi-
                        tion, no carry out.
```

```
6.  -57  =  1100 0111
    -92  =  1010 0100
    ------------------
   -149  = *0110 1011   *Overflow-no carry
                        into leftmost posi-
                        tion but carry out.
```

The presence or absence of an overflow condition may be recognized from the carries:

- There is no overflow:
  a. If there is no carry into the leftmost bit position and no carry out (examples 1 and 3).
  b. If there is a carry into the leftmost position and also a carry out (examples 2 and 4).
- There is an overflow:
  a. If there is a carry into the leftmost position but no carry out (example 5).
  b. If there is no carry into the leftmost position but there is a carry out (example 6).

The following are 16-bit signed binary integers. The first is the maximum positive 16-bit binary integer. The last is the maximum negative 16-bit binary integer (the negative 16-bit binary integer with the greatest absolute value).

```
2^15 - 1 =   32,767  =  0 111 1111 1111 1111
2^0      =        1  =  0 000 0000 0000 0001
0        =        0  =  0 000 0000 0000 0000
-2^0     =       -1  =  1 111 1111 1111 1111
-2^15    =  -32,768  =  1 000 0000 0000 0000
```

The following are several 32-bit signed binary integers arranged in descending order. The first is the maximum positive binary integer that can be represented by 32 bits, and the last is the maximum negative binary integer that can be represented by 32 bits.

```
2^31 - 1 =  2 147 483 647 = 0 111 1111 1111 1111 1111 1111 1111 1111
2^16     =         65 536 = 0 000 0000 0000 0001 0000 0000 0000 0000
2^0      =              1 = 0 000 0000 0000 0000 0000 0000 0000 0001
0        =              0 = 0 000 0000 0000 0000 0000 0000 0000 0000
-2^0     =             -1 = 1 111 1111 1111 1111 1111 1111 1111 1111
-2^1     =             -2 = 1 111 1111 1111 1111 1111 1111 1111 1110
-2^16    =        -65 536 = 1 111 1111 1111 1111 0000 0000 0000 0000
-2^31 + 1 = -2 147 483 647 = 1 000 0000 0000 0000 0000 0000 0000 0001
-2^31     = -2 147 483 648 = 1 000 0000 0000 0000 0000 0000 0000 0000
```

Unsigned Binary Integers

Certain instructions, such as ADD LOGICAL, treat binary integers as unsigned rather than signed. Unsigned binary integers have the same format as signed binary integers, except that the leftmost bit is interpreted as another numeric bit rather than a sign bit. There is no complement notation because all unsigned binary integers are considered positive.

The following examples illustrate the addition of unsigned binary integers. Only eight bit positions are used. The examples are numbered the same as the corresponding examples for signed binary integers.

```
1.  57  =  0011 1001
    35  =  0010 0011
   ----------------
    92  =  0101 1100
```

```
2.  57  =  0011 1001
   221  =  1101 1101
   ----------------
   278  = *0001 0110   *Carry out of
                       leftmost position
```

```
3.  35  =  0010 0011
   199  =  1100 0111
   ----------------
   234  =  1110 1010
```

```
4. 199  =  1100 0111
   221  =  1101 1101
   ----------------
   420  = *1010 0100   *Carry out of
                       leftmost position
```

```
5.  57  =  0011 1001
    92  =  0101 1100
   ----------------
   149  =  1001 0101
```

```
6. 199  =  1100 0111
   164  =  1010 0100
   ----------------
   363  = *0110 1011   *Carry out of
                       leftmost position
```

A carry out of the leftmost bit position may or may not imply an overflow, depending on the application.

The following are several 32-bit unsigned binary integers arranged in descending order.

```
2^32 - 1 =  4 294 967 296 = 1111 1111 1111 1111 1111 1111 1111 1111
2^31     =  2 147 483 648 = 1000 0000 0000 0000 0000 0000 0000 0000
2^31 - 1 =  2 147 483 647 = 0111 1111 1111 1111 1111 1111 1111 1111
2^16     =         65 536 = 0000 0000 0000 0001 0000 0000 0000 0000
2^0      =              1 = 0000 0000 0000 0000 0000 0000 0000 0001
0        =              0 = 0000 0000 0000 0000 0000 0000 0000 0000
```

Decimal Integers

Decimal integers are represented as one or more decimal digits and a sign digit. Each digit is a 4-bit code. The decimal digits are in binary-coded decimal (BCD) form, with the values 0-9 encoded as 0000-1001. The sign is usually represented as 1100 (C hex) for plus and 1101 (D hex) for minus. These are the preferred sign codes, which are generated by the machine for the results of decimal operations. There are also several alternate sign codes (1010, 1110, and 1111 for plus; 1011 for minus). The alternate sign codes are accepted by the machine as valid but are not generated for results.

Decimal integers may have different lengths, from one to 16 bytes. There are two decimal formats: packed and zoned. In the packed format, each byte contains two decimal digits, except for the rightmost byte which contains the sign in its right digit. The number of decimal digits in the packed format can vary from one to 31. Because decimal integers must consist of whole bytes and there must be a sign digit on the right, the number of decimal digits is always odd. If an even number of significant digits is desired, a leading zero must be inserted on the left.

In the zoned format, each byte consists of a decimal digit on the right and the zone code 1111 (F hex) on the left, except for the rightmost byte where the sign code replaces the zone code. Thus, decimal integers in the zoned format can have anywhere from one to 16 digits. The zoned format

may be used directly for input and output in the extended binary-coded-decimal interchange code (EBCDIC), except that the sign must be separated from the rightmost digit and handled as a separate character. For positive (unsigned) numbers, however, the sign code of the rightmost digit can simply be replaced by the zone code, which is one of the acceptable alternate codes for plus.

In either format, negative decimal integers are represented in true notation with a separate sign. As for binary integers, the radix point (decimal point) of decimal integers is considered to be fixed at the right, and any scaling is done by the programmer.

The following are some examples of decimal integers shown in hexadecimal notation:

Value	Packed Format	Zoned Format
+123	12 3C	F1 F2 C3 or F1 F2 F3
−4321	04 32 1D	F4 F3 F2 D1
+000050	00 00 05 0C	F0 F0 F0 F0 F5 C0 or F0 F0 F0 F0 F5 F0
−7	7D	D7
00000	00 00 0C	F0 F0 F0 F0 C0 or F0 F0 F0 F0 F0

Under some circumstances, a zero with a minus sign (negative zero) is produced. For example, the multiplicand:

00 12 3D (−123)

times the multiplier:

0C (+0)

generates the product:

00 00 0D (−0)

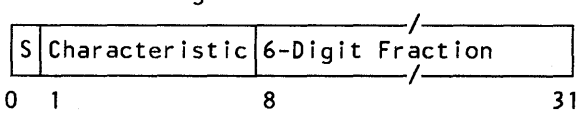
because the product sign follows the algebraic rule of signs even when the value is zero. A negative zero, however, is entirely equivalent to a positive zero; they compare equal in a decimal comparison.

Floating-Point Numbers

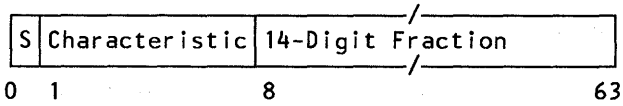
A floating-point number is expressed as a fraction multiplied by a separate power of 16. The term floating point indicates that the radix-point placement, or scaling, is automatically maintained by the machine.

The part of a floating-point number which represents the significant digits of the number is called the fraction. A second part specifies the power (exponent) to which 16 is raised and indicates the location of the radix point of the number. The fraction and exponent may be represented by 32 bits (short format), 64 bits (long format), or 128 bits (extended format).

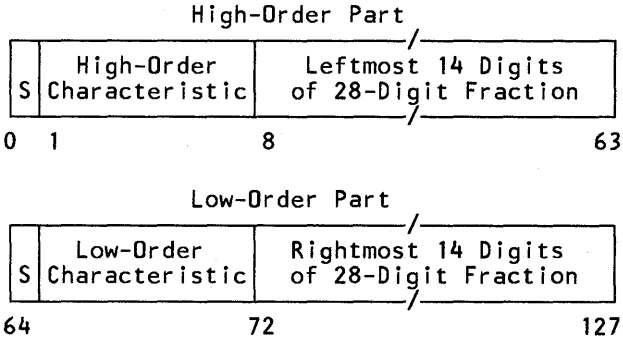
Short Floating-Point Number



Long Floating-Point Number



Extended Floating-Point Number



A floating-point number has two signs: one for the fraction and one for the exponent. The fraction sign, which is also the sign of the entire number, is the leftmost bit of each format (0 for plus, 1 for minus). The numeric part of the fraction is in true notation regardless of the sign. The numeric part is contained in bits 8-31 for the short format, in bits 8-63 for the long format, and in bits 8-63 followed by bits 72-127 for the extended format.

The exponent sign is obtained by expressing the exponent in excess-64 notation; that is, the exponent is added as a signed number to 64. The resulting number is called the characteristic. It is located in bits 1-7 for all formats. The characteristic can vary from 0 to 127, permitting the exponent to vary from −64 through 0 to +63. This provides a scale multiplier in the range of 16^{-64} to 16^{+63}. A nonzero fraction, if normalized, must be less than one and greater than or equal to 1/16, so that the range covered by the magnitude M of a floating-point number is:

$$16^{-65} \leq M < 16^{63}$$

In decimal terms:

16^{-65} is approximately equal to 5.4×10^{-79}
16^{63} is approximately equal to 7.2×10^{75}

More precisely,

In the short format:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \leq M \leq (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \leq M \leq (1 - 16^{-28}) \times 16^{63}$$

Within a given fraction length (6, 14, or 28 digits), a floating-point operation will provide the greatest precision if the fraction is normalized. A fraction is normalized when the leftmost digit (bit positions 8, 9, 10, and 11) is nonzero. It is unnormalized if the leftmost digit contains all zeros.

If normalization of the operand is desired, the floating-point instructions that provide automatic normalization are used. This automatic normalization is accomplished by left-shifting the fraction (four bits per shift) until a nonzero digit occupies the leftmost digit position. The characteristic is reduced by one for each digit shifted.

The following 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	$= +1/16 \times 16^{1}$	= 0 100 0001 0001 0000 0000 0000 0000 0000
0.5	$= +8/16 \times 16^{0}$	= 0 100 0000 1000 0000 0000 0000 0000 0000
1/64	$= +4/16 \times 16^{-1}$	= 0 011 1111 0100 0000 0000 0000 0000 0000
0.0	$= +0 \times 16^{-64}$	= 0 000 0000 0000 0000 0000 0000 0000 0000
-15.0	$= -15/16 \times 16^{1}$	= 1 100 0001 1111 0000 0000 0000 0000 0000
5.4×10^{-79}	$\neq +1/16 \times 16^{-64}$	= 0 000 0000 0001 0000 0000 0000 0000 0000
7.2×10^{75}	$\neq (1-16^{-6}) \times 16^{63}$	= 0 111 1111 1111 1111 1111 1111 1111 1111

[The symbol ≠ means "approximately equal."]

Conversion Example

Convert the decimal number 59.25 to a short floating-point number. (In another appendix are tables for the conversion of hexadecimal and decimal integers and fractions.)

1. The number is decomposed into a decimal integer and a decimal fraction.

   59.25 = 59 plus 0.25

2. The decimal integer is converted to its hexadecimal representation.

   $59_{10} = 3B_{16}$

3. The decimal fraction is converted to its hexadecimal representation.

   $0.25_{10} = 0.4_{16}$

4. The integral and fractional parts are combined and expressed as a fraction times a power of 16 (exponent).

   $3B.4_{16} = 0.3B4_{16} \times 16^{2}$

5. The characteristic is developed from the exponent and converted to binary.

   base + exponent = characteristic
   64 + 2 = 66 = 1000010

6. The fraction is converted to binary and grouped hexadecimally.

   $.3B4_{16} = .0011\ 1011\ 0100$

7. The characteristic and the fraction are stored in the short format. The sign position contains the sign of the fraction.

   S Char Fraction

   0 10000 000010 0011 1011 0100 0000 0000

Examples of instruction sequences that may be used to convert between signed binary integers and floating-point numbers are shown in the section "Floating-Point-Number Conversion" later in this appendix.

Instruction-Use Examples

The following examples illustrate the use of many of the unprivileged instructions. Before studying one of these examples, the reader should consult the instruction description in this manual for the particular instruction of interest to him.

The instruction-use examples are written principally for assembler-language programmers, to be used in conjunction with the appropriate assembler-language manuals.

Most examples present one particular instruction, both as it is written in an assembler-language statement and as it appears when assembled in storage (machine format).

Machine Format

All machine-format numerical operands are written in hexadecimal notation unless otherwise specified. Hexadecimal operands are shown converted into binary, decimal, or both if such conversion helps to clarify the example for the reader. Storage addresses are also given in hexadecimal.

Assembler-Language Format

In assembler-language statements, registers and lengths are presented in decimal. Displacements, immediate operands, and masks may be shown in decimal, hexadecimal, or binary notation; for example, 12, X'C', or B'1100' represent the same value. Whenever the value in a register or storage

location is referred to as "not significant," this value is replaced during the execution of the instruction.

When SS-format instructions are written in the assembler language, lengths are given as the total number of bytes in the field. This differs from the machine definition, in which the length field specifies the number of bytes to be added to the field address to obtain the address of the last byte of the field. Thus, the machine length is one less than the assembler-language length. The assembler program automatically subtracts one from the length specified when the instruction is assembled.

In some of the examples, symbolic addresses are used in order to simplify the examples. In assembler-language statements, a symbolic address is represented as a mnemonic term written in all capitals, such as FLAGS which may denote the address of a storage location containing data or program-control information. When symbolic addresses are used, the assembler supplies actual base and displacement values according to the programmer's specifications. Therefore, the actual values for base and displacement are not shown in the assembler-language format or in the machine-language format. For assembler-language formats, in the labels that designate instruction fields, the letter "S" is used to indicate the combination of base and displacement fields for an operand address. (For example, S1 represents the combination of B1 and D1.) In the machine-language format, the base and displacement address components are shown as asterisks (***).

General Instructions

(See Chapter 7.)

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 in propagating the leftmost (sign) bit 16 positions to the left. For example, assume that the contents of storage locations 2000-2001 are to be added to register 5. Initially:

Register 5 contains 00 00 00 19 = 25_{10}.
Storage locations 2000-2001 contain FF FE = -2_{10}.
Register 12 contains 00 00 18 00.
Register 13 contains 00 00 01 50.

The format of the required instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4A	5	D	C	6B0

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
AH	5,X'6B0'(13,12)

After the instruction is executed, register 5 contains 00 00 00 17 = 23_{10}.

AND (N, NR, NI, NC)

When the Boolean operator AND is applied to two bits, the result is one when both bits are one; otherwise, the result is zero. When two bytes are ANDed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of ANDing two bytes:

First-operand byte:	0011 0101_2
Second-operand byte:	0101 1100_2
Result byte:	0001 0100_2

And (NI)

A frequent use of the AND instruction is to set a particular bit to zero. For example, assume that storage location 4891 contains 0100 0011_2. To set the rightmost bit of this byte to zero without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

Op Code	I_2	B_1	D_1
94	FE	8	001

Assembler Format

Op Code	$D_1(B_1),I_2$
NI	1(8),X'FE'

When this instruction is executed, the byte in storage is ANDed with the immediate byte (the I_2 field of the instructions):

Location 4891	$0100\ 0011_2$
Immediate byte	$1111\ 1110_2$
Result:	$0100\ 0010_2$

The resulting byte, with bit 7 set to zero, is stored back in location 4891. Condition code 1 is set.

BRANCH AND LINK (BAL, BALR)

The BRANCH AND LINK instructions are commonly used to branch to a subroutine with the option of later returning to the main instruction sequence. For example, assume that 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.
Address 00 00 C6 contains the BAL instruction, so that 00 00 CA is the address of the next sequential instruction.

The format of the BAL instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
45	2	0	5	010

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
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 set to zero may be used to load a register for use as a base register. For example, in the assembler language, the sequence of statements:

```
BALR   15,0
USING  *,15
```

tells the assembler program that register 15 is to be used as the base register in assembling this program 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.)

As another example, BALR 6,0 may be used to preserve the current condition code in bits 2 and 3 of register 6 for future inspection.

Note that, in all three examples, a value of zero in the X_2 or R_2 field indicates that the corresponding function is not performed; it does not refer to register 0. Register 0 can be designated by the R_1 field, however.

BRANCH ON CONDITION (BC, BCR)

The BRANCH ON CONDITION instructions test the condition code to see whether a branch should or should not be taken. The branch is taken only if the 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 or 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	M_1	X_2	B_2	D_2
47	C	B	A	050

Assembler Format

Op Code	$M_1,D_2(X_2,B_2)$
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 ON COUNT (BCT, BCTR)

The BRANCH ON COUNT instructions are often used to execute a program loop for a specified number of times. For example, assume that the following represents some lines of coding in an assembler-language program:

```
      .
      .
      .
LUPE  AR   8,1
      .
      .
BACK  BCT  6,LUPE
      .
      .
      .
```

where register 6 contains 00 00 00 03 and the address of LUPE is 6826. Assume that, in order to address this location, register 10 is used as a base register and contains 00 00 68 00.

The format of the BCT instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
46	6	0	A	026

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
BCT	6,X'26'(0,10)

The effect of the coding is to execute three times the loop defined by locations LUPE through 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 compare value. For example, assume that:

Register 4 contains 00 00 00 8A = 138_{10} = the index.
Register 6 contains 00 00 00 02 = 2_{10} = the increment.
Register 7 contains 00 00 00 AA = 170_{10} = the compare value.
Register 10 contains 00 00 71 30 = the branch address.

The format of the instruction is:

Machine Format

Op Code	R_1	R_3	B_2	D_2
86	4	6	A	000

Assembler Format

Op Code	$R_1,R_3,D_2(B_2)$
BXH	4,6,0(10)

When the instruction is executed, first the contents of register 6 are added to register 4, second the sum is compared with the contents of register 7, and third the decision whether to branch is made. After execution:

Register 4 contains 00 00 00 8C = 140_{10}
Registers 6 and 7 are unchanged.

Since the new value in register 4 is not yet greater than the value in register 7, the branch to address 7130 is not taken. Repeated use of the instruction will eventually cause the branch to be taken when the value in register 4 reaches 172.

When the register used to contain the increment is odd, that register also becomes the compare-value register. The following assembler-language subroutine illustrates how this feature may be used to search a table.

Table	
2 Bytes	2 Bytes
ARG1	FUNCT1
ARG2	FUNCT2
ARG3	FUNCT3
ARG4	FUNCT4
ARG5	FUNCT5
ARG6	FUNCT6

Assume that:

Register 8 contains the search argument.
Register 9 contains the width of the table in bytes (00 00 00 04).
Register 10 contains the length of the table in bytes (00 00 00 18).
Register 11 contains the starting address of the table.
Register 14 contains the return address to the main program.

As the following subroutine is executed, the argument in register 8 is successively compared with the arguments in the table, starting with argument 6 and working backward to argument 1. If an equality is found, the corresponding function replaces the argument in register 8. If an equality is not found, $FFFF_{16}$ replaces the argument in register 8.

```
SEARCH     LNR   9,9
NOTEQUAL   BXH   10,9,LOOP
NOTFOUND   LA    8,X'FFFF'
           BCR   15,14
LOOP       CH    8,0(2,3)
           BC    7,NOTEQUAL
           LH    8,2(10,11)
           BCR   15,14
```

The first instruction (LNR) causes the value in register 9 to be made negative. After execution of

this instruction, register 9 contains FFFFFFFC = -4_{10}. Considering the case when no equality is found, the BXH instruction will be executed seven times. Each time BXH is executed, a value of –4 is added to register 10, thus reducing the value in register 10 by 4. The new value in register 10 is compared with the –4 value in register 9. The branch is taken each time until the value in register 10 is –4.

BRANCH ON INDEX LOW OR EQUAL (BXLE)

This instruction is similar to BRANCH ON INDEX HIGH except that the branch is successful when the sum is low or equal compared to the compare value.

COMPARE HALFWORD (CH)

The COMPARE HALFWORD instruction compares a 16-bit signed binary integer in storage with the contents of a register. For example, assume that:

Register 4 contains FF FF 80 00 = $-32{,}768_{10}$.
Register 13 contains 00 01 60 50.
Storage locations 16080-16081 contain 8000 = $-32{,}768_{10}$.

When the instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
49	4	0	D	030

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
CH	4,X'30'(0,13)

is executed, the contents of locations 16080-16081 are fetched, expanded to 32 bits (the sign bit is propagated to the left), and compared with the contents of register 4. Because the two numbers are equal, condition code 0 is set.

COMPARE LOGICAL (CL, CLC, CLI, CLR)

The COMPARE LOGICAL instructions differ from the signed-binary comparison instructions (C, CH, CR) in that all quantities are handled as unsigned binary integers or as unstructured data.

Compare Logical (CLC)

The COMPARE LOGICAL (CLC) instruction can be used to perform the byte-by-byte comparison of storage fields up to 256 bytes in length. For example, assume that the following two fields of data are in storage:

Field 1
1886 1891

D1	D6	C8	D5	E2	D6	D5	6B	C1	4B	C2	4B

Field 2
1900 190B

D1	D6	C8	D5	E2	D6	D5	6B	C1	4B	C3	4B

Also assume:

Register 9 contains 00 00 18 80.
Register 7 contains 00 00 19 00.

Execution of the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D5	0B	9	006	7	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
CLC	6(12,9),0(7)

sets condition code 1, indicating that the contents of field 1 are lower in value than the contents of field 2.

Because the collating sequence of the EBCDIC code is determined simply by a logical comparison of the bits in the code, the CLC instruction can be used to collate EBCDIC-coded fields. For example, in EBCDIC, the above two data fields are:

Field 1 JOHNSON,A.B.
Field 2 JOHNSON,A.C.

Condition code 1 tells us that A.B.JOHNSON precedes A.C.JOHNSON, thus placing the names in the correct alphabetic sequence.

Compare Logical (CLI)

The COMPARE LOGICAL (CLI) instruction compares a byte from the instruction stream with a byte from storage. For example, assume that:

Register 10 contains 00 00 17 00.
Storage location 1703 contains 7E.

Execution of the instruction:

Machine Format

Op Code	I_2	B_1	D_1
95	AF	A	003

Assembler Format

Op Code	$D_1(B_1),I_2$
CLI	3(10),X'AF'

sets condition code 1, indicating that the first operand (the quantity in main storage) is lower than the second (immediate) operand.

Compare Logical (CLR)

Assume that:

Register 4 contains 00 00 00 01 = 1.
Register 7 contains FF FF FF FF = $2^{32}-1$.

Execution of the instruction:

Machine Format

Op Code	R_1	R_2
15	4	7

Assembler Format

Op Code	R_1,R_2
CLR	4,7

sets condition code 1. Condition code 1 indicates that the first operand is lower than the second.

If, instead, the signed-binary comparison instruction COMPARE (CR) had been executed, the contents of register 4 would have been interpreted as +1 and the contents of register 7 as −1. Thus, the first operand would have been higher, so that condition code 2 would have been set.

COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)

The COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) instruction provides a means of comparing bytes selected from a general register to a contiguous field of bytes in storage. The M_3 field of the CLM instruction is a four-bit mask that selects zero to four bytes from a general register, each mask bit corresponding, left to right, to a register byte. In the comparison, the register bytes corresponding to ones in the mask are treated as a contiguous field. The operation proceeds left to right. For example, assume that:

Three bytes starting at storage location 10200 contain F0 BC 7B
Register 12 contains 10000
Register 6 contains F0 BC 5C 7B

Execution of the instruction:

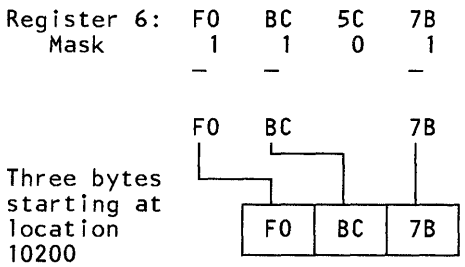
Machine Format

Op Code	R_1	M_3	B_2	D_2
BD	6	D	C	200

Assembler Format

Op Code	$R_1,M_3,D_2(B_2)$
CLM	6,B'1101',X'200'(12)

causes the following comparison:



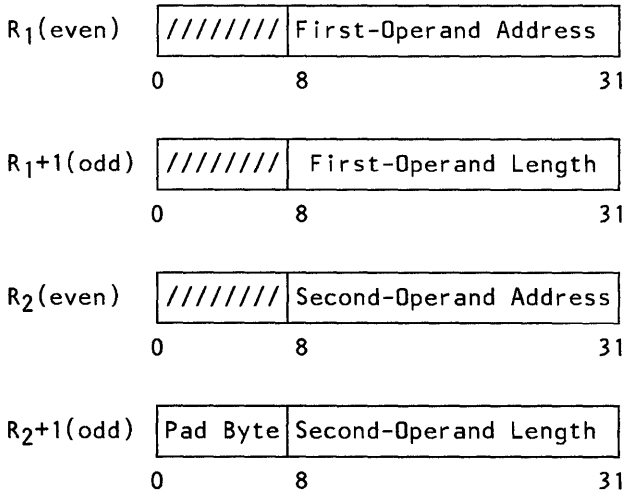
Because the selected bytes are equal, condition code 0 is set.

COMPARE LOGICAL LONG (CLCL)

The COMPARE LOGICAL LONG (CLCL) instruction is used to compare two operands in storage, byte by byte. Each operand can be of any length. Two even-odd pairs of general registers (four registers in all) are used to locate the operands and to control the execution of the CLCL instruction, as illustrated in the following diagram. The first register of each pair must be an even register, and it contains the storage address of the byte currently being compared in each operand. The odd register of each pair contains the length of the operand it covers, and the leftmost byte of the second-operand odd register contains a padding

byte which is used to extend the shorter operand, if any, to the same length as the longer operand.

The following illustrates the assignment of registers:



Since the CLCL instruction may be interrupted during execution, the interrupting program must preserve the contents of the four registers for use when the instruction is resumed.

The following instructions set up two register pairs to control a text-string comparison. For example, assume:

```
Operand 1
Address: 20800 (hex)
Length:    100 (dec)

Operand 2
Address: 20A00 (hex)
Length:    132 (dec)

Padding Byte
Address: 20003 (hex)
Length:      1
Value:      40 (hex)

Register 12 contains 00 02 00 00
```

The setup instructions are:

LA	4,X'800'(12)	Point register 4 to start of first operand
LA	5,100	Set register 5 to length of first operand
LA	8,X'A00'(12)	Point register 8 to start of second operand
LA	9,132	Set register 9 to length of second operand
ICM	9,B'1000',3(12)	Insert padding byte in leftmost byte position of register 9.

Register pair 4,5 defines the first operand. Bits 8-31 of register 4 contain the storage address of the start of an EBCDIC text string, and bits 8-31 of register 5 contain the length of the string, in this case 100 bytes.

Register pair 8,9 defines the second operand, with bits 8-31 of register 8 containing the starting location of the second operand and bits 8-31 of register 9 containing the length of the second operand, in this case 132 bytes. Bits 0-7 of register 9 contain an EBCDIC blank character (X'40') to pad the shorter operand. In this example, the padding byte is used in the first operand, after the 100th byte, to compare with the remaining bytes in the second operand.

With the register pairs thus set up, the format of the CLCL instruction is:

Machine Format

Op Code	R_1	R_2
0F	4	8

Assembler Format

Op Code	R_1,R_2
CLCL	4,8

When this instruction is executed, the comparison starts at the left end of each operand and proceeds to the right. The operation ends as soon as an inequality is detected or the end of the longer operand is reached.

If this CLCL instruction is interrupted after 60 bytes have compared equal, the operand lengths in registers 5 and 9 will have been decremented to X'28' and X'48', respectively, and the operand addresses in registers 4 and 8 will have been incremented to X'2083C' and X'20A3C'. The padding byte X'40' remains in register 9. When the CLCL instruction is reissued with these register contents, the comparison resumes at the point of interruption.

Now, assume that the instruction is interrupted after 110 bytes. That is, the first 100 bytes of the second operand have compared equal to the first operand, and the next 10 bytes of the second operand have compared equal to the padding byte (blank). The residual operand lengths in registers 5 and 9 are 0 and X'16', respectively, and the operand addresses in registers 4 and 8 are X'20864'

(the value when the first operand was exhausted, and X'20A6E' (the current value for the second operand).

When the comparison ends, the condition code is set to 0, 1, or 2, depending on whether the first operand is equal to, less than, or greater than the second operand, respectively.

When the operands are unequal, the addresses in registers 4 and 8 locate the bytes that caused the mismatch.

CONVERT TO BINARY (CVB)

The CONVERT TO BINARY instruction converts an eight-byte, packed-decimal number into a signed binary integer and loads the result into a general register. After the conversion operation is completed, the number is in the proper form for use as an operand in signed binary arithmetic. For example, assume:

Storage locations 7608-760F contain a decimal number in the packed format: 00 00 00 00 00 25 59 4C (+25,594).
The contents of register 7 are not significant.
Register 13 contains 00 00 76 00.

The format of the conversion instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4F	7	0	D	008

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
CVB	7,8(0,13)

After the instruction is executed, register 7 contains 00 00 63 FA.

CONVERT TO DECIMAL (CVD)

The CONVERT TO DECIMAL instruction performs functions exactly opposite to those of the CONVERT TO BINARY instruction. CVD converts a signed binary integer in a register to packed decimal and stores the eight-byte result. For example, assume:

Register 1 contains the signed binary integer: 00 00 0F 0F.
Register 13 contains 00 00 76 00.

The format of the instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4E	1	0	D	008

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
CVD	1,8(0,13)

After the instruction is executed, storage locations 7608-760F contain 00 00 00 00 00 03 85 5C (+3855).

The plus sign generated is the preferred plus sign, 1100_2.

DIVIDE (D, DR)

The DIVIDE instruction divides the dividend in an even-odd register pair by the divisor in a register or in storage. Since the dividend is assumed to be 64 bits long, it is important that the proper sign be first affixed. For example, assume that:

Storage locations 3550-3553 contain 00 00 08 DE = 2270_{10} = the dividend.
Storage locations 3554-3557 contain 00 00 00 32 = 50_{10} = the divisor.
The initial contents of registers 6 and 7 are not significant.
Register 8 contains 00 00 35 50.

The following assembler language statements load the registers properly and perform the divide operation:

Statement	Comments
L 6,0(0,8)	Places 00 00 08 DE into register 6.
SRDA 6,32(0)	Shifts 00 00 08 DE into register 7. Register 6 is filled with zeros (sign bits).
D 6,4(0,8)	Performs the division.

The machine format of the above DIVIDE instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
5D	6	0	8	004

After all the foregoing 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 might not have been filled with the proper sign bits (zeros in this example), and the DIVIDE instruction might not have given the expected results.

EXCLUSIVE OR (X, XC, XI, XR)

When the Boolean operator EXCLUSIVE OR is applied to two bits, the result is one when either, but not both, of the two bits is one; otherwise, the result is zero. When two bytes are EXCLUSIVE ORed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of the EXCLUSIVE OR of two bytes:

```
First-operand byte:    0011 0101₂
Second-operand byte:   0101 1100₂
-------------------------------------
Result byte:           0110 1001₂
```

Exclusive Or (XC)

The EXCLUSIVE OR (XC) instruction can be used to exchange the contents of two areas in storage without the use of an intermediate storage area. For example, assume two three-byte fields in storage:

```
              359    35B
Field 1      | 00 | 17 | 90 |

              360    362
Field 2      | 00 | 14 | 01 |
```

Execution of the instruction (assume that register 7 contains 00 00 03 58):

Machine Format

Op Code	L	B_1	D_1	B_1	D_2
D7	02	7	001	7	008

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
XC	1(3,7),8(7)

Field 1 is EXCLUSIVE ORed with field 2 as follows:

```
Field 1:  0000 0000 0001 0111 1001 0000₂
        = 00 17 90
Field 2:  0000 0000 0001 0100 0000 0001₂
        = 00 14 01
-------------------------------------------
Result:   0000 0000 0000 0011 1001 0001₂
        = 00 03 91
```

The result replaces the former contents of field 1.

Now, execution of the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D7	02	7	008	7	001

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
XC	8(3,7),1(7)

produces the following result:

```
Field 1: 0000 0000 0000 0011 1001 0001₂
       = 00 03 91
Field 2: 0000 0000 0001 0100 0000 0001₂
       = 00 14 01
-------------------------------------------
Result:  0000 0000 0001 0111 1001 0000₂
       = 00 17 90
```

The result of this operation replaces the former contents of field 2. Field 2 now contains the original value of field 1.

Lastly, execution of the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D7	02	7	001	7	008

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
XC	1(3,7),8(7)

produces the following result:

```
Field 1: 0000 0000 0000 0011 1001 0001₂
       = 00 03 91
Field 2: 0000 0000 0001 0111 1001 0000₂
       = 00 17 90
-------------------------------------------
Result:  0000 0000 0001 0100 0000 0001₂
       = 00 14 01
```

The result of this operation replaces the former contents of field 1. Field 1 now contains the original value of field 2.

Exclusive Or (XI)

A frequent use of the EXCLUSIVE OR (XI) instruction is to invert a bit (change a zero bit to a one or a one bit to a zero). For example, assume that storage location 8082 contains 0110 1001_2. To invert the leftmost and rightmost bits without affecting any of the other bits, the following instruction can be used (assume that register 9 contains 00 00 80 80):

Machine Format

Op Code	I_2	B_1	D_1
97	81	9	002

Assembler Format

Op Code	$D_1(B_1),I_2$
XI	2(9),X'81'

When the instruction is executed, the byte in storage is EXCLUSIVE ORed with the immediate byte (the I_2 field of the instruction):

```
Location 8082:    0110 1001₂
Immediate byte:   1000 0001₂
-------------------------------
Result:           1110 1000₂
```

The resulting byte is stored back in location 8082. Condition code 1 is set to indicate a nonzero result.

Notes:

1. *With the XC instruction, fields up to 256 bytes in length can be exchanged.*
2. *With the XR instruction, the contents of two registers can be exchanged.*
3. *Because the X instruction operates storage to register only, an exchange cannot be made solely by the use of X.*
4. *A field EXCLUSIVE ORed with itself is cleared to zeros.*
5. *For additional examples of the use of EXCLUSIVE OR, see the section "Floating-Point-Number Conversion" later in this appendix.*

EXECUTE (EX)

The EXECUTE instruction causes one *target instruction* in main storage to be executed out of sequence without actually branching to the target instruction. Unless the R_1 field of the EXECUTE instruction is zero, bits 8-15 of the target instruction are ORed with bits 24-31 of the R_1 register before the target instruction is executed. Thus, EXECUTE may be used to supply the length field for an SS instruction without modifying the SS instruction in storage. For example, assume that a MOVE (MVC) instruction is the target that is located at address 3820, with a format as follows:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	00	C	003	D	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
MVC	3(1,12),0(13)

where register 12 contains 00 00 89 13 and register 13 contains 00 00 90 A0.

Further assume that at storage address 5000, the following EXECUTE instruction is located:

Machine Format

Op Code	R_1	X_2	B_2	D_2
44	1	0	A	000

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
EX	1,0(0,10)

where register 10 contains 00 00 38 20 and register 1 contains 00 0F F0 03.

When the instruction at 5000 is executed, the rightmost byte of register 1 is ORed with the second byte of the target instruction:

```
Register byte:      0000 0000₂ = 00
Instruction byte:   0000 0011₂ = 03
-------------------------------------
Result:             0000 0011₂ = 03
```

causing the instruction at 3820 to be executed as if it originally were:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	03	C	003	D	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
MVC	3(4,12),0(13)

However, after execution:

Register 1 is unchanged.
The instruction at 3820 is unchanged.
The contents of the four bytes starting at location 90A0 have been moved to the four bytes starting at location 8916.
The CPU next executes the instruction at address 5004 (PSW bits 40-63 contain 00 50 04).

INSERT CHARACTERS UNDER MASK (ICM)

The INSERT CHARACTERS UNDER MASK (ICM) instruction may be used to replace all or selected bytes in a general register with bytes from storage and to set the condition code to indicate the value of the inserted field.

For example, if it is desired to insert a three-byte address from FIELDA into register 5 and leave the leftmost byte of the register unchanged, assume:

Machine Format

Op Code	R_1	M_3	S_2
BF	5	7	* * * *

Assembler Format

Op Code	R_1,M_3,S_2
ICM	5,B'0111',FIELDA

```
FIELDA:                    FE DC BA
Register 5 (before):       12 34 56 78
Register 5 (after):        12 FE DC BA
Condition code (after):    1   (leftmost bit
                                of inserted
                                field is one)
```

As another example:

Machine Format

Op Code	R_1	M_3	S_2
BF	6	9	* * * *

Assembler Format

Op Code	R_1,M_3,S_2
ICM	6,B'1001',FIELDB

```
FIELDB:                    12 34
Register 6 (before):       00 00 00 00
Register 6 (after):        12 00 00 34
Condition code (after):    2  (inserted field
                               is nonzero
                               with leftmost
                               zero bit)
```

When the mask field contains 1111, the ICM instruction produces the same result as LOAD (L) (provided that the indexing capability of the RX format is not needed), except that ICM also sets the condition code. The condition-code setting is useful when an all-zero field (condition code 0) or a leftmost one bit (condition code 1) is used as a flag.

LOAD (L, LR)

The LOAD instructions take four bytes from storage or from a general register and place them unchanged into a general register. For example, assume that the four bytes starting with location 21003 are to be loaded into register 10. Initially:

Register 5 contains 00 02 00 00.
Register 6 contains 00 00 10 03.
The contents of register 10 are not significant.
Storage locations 21003-21006 contain 00 00 AB CD.

To load register 10, the RX form of the instruction can be used:

Machine Format

Op Code	R_1	X_2	B_2	D_2
58	A	5	6	000

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
L	10,0(5,6)

After the instruction is executed, register 10 contains 00 00 AB CD.

LOAD ADDRESS (LA)

The LOAD ADDRESS instruction provides a convenient way to place a nonnegative binary integer up to 4095_{10} in a register without first defining a constant and then using it as an operand. For example, the following instruction places the number 2048_{10} in register 1:

Machine Format

Op Code	R_1	X_2	B_2	D_2
41	1	0	0	800

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
LA	1,2048(0,0)

The LOAD ADDRESS instruction can also be used to increment a register by an amount up to 4095_{10} specified in the D_2 field. Only the rightmost 24 bits of the result are retained, however. For example, assume that register 5 contains 00 12 34 56.

The instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
41	5	0	5	00A

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
LA	5,10(0,5)

adds 10 (decimal) to the contents of register 5 as follows:

```
Register 5 (old): 00 12 34 56
D2 field:         00 00 00 0A
Register 5 (new): 00 12 34 60
```

The register may be specified as either B_2 or X_2. Thus, the instruction LA 5,10(5,0) produces the same result.

As the most general example, the instruction LA 6,10(5,4) adds the displacement, in this case 10, to the contents of register 4 and to the contents of register 5 and places the 24-bit sum of these three values in register 6.

LOAD HALFWORD (LH)

The LOAD HALFWORD instruction places unchanged a halfword from storage into the right half of a register. The left half of the register is loaded with zeros or ones according to the sign (leftmost bit) of the halfword.

For example, assume that the two bytes in storage locations 1803-1804 are to be loaded into register 6. Also assume:

The contents of register 6 are not significant.
Register 14 contains 00 00 18 03.
Locations 1803-1804 contain 00 20.

The instruction required to load the register is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
48	6	0	E	000

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
LH	6,0(0,14)

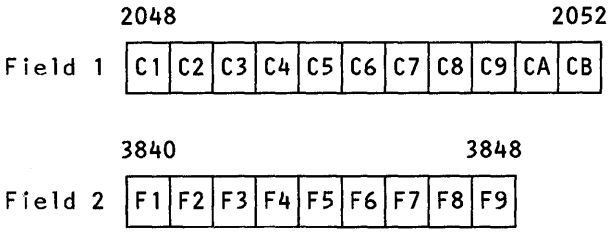
After the instruction is executed, register 6 contains 00 00 00 20. If locations 1803-1804 had contained a negative number, for example, A7 B6, a minus sign would have been propagated to the left, giving FF FF A7 B6 as the final result in register 6.

MOVE (MVC, MVI)

Move (MVC)

The MOVE (MVC) instruction can be used to move data from one storage location to another.

For example, assume that the following two fields are in storage:



Also assume:

Register 1 contains 00 00 20 48.
Register 2 contains 00 00 38 40.

With the following instruction, the first eight bytes of field 2 replace the first eight bytes of field 1:

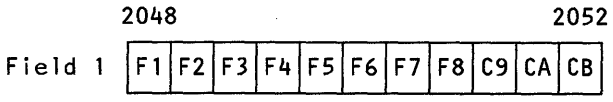
Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	07	1	000	2	000

Assembler Format

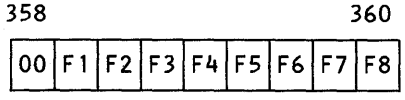
Op Code	$D_1(L,B_1),D_2(B_2)$
MVC	0(8,1),0(2)

After the instruction is executed, field 1 becomes:



Field 2 is unchanged.

MVC can also be used to propagate a byte through a field by starting the first-operand field one byte location to the right of the second-operand field. For example, suppose that an area in storage starting with address 358 contains the following data:



With the following MVC instruction, the zeros in location 358 can be propagated throughout the entire field (assume that register 11 contains 00 00 03 58):

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	07	B	001	B	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
MVC	1(8,11),0(11)

Because 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, and so on, until the entire field is filled with zeros. Note that an MVI instruction could have been used originally to place the byte of zeros in location 358.

Notes:

1. *Although the field occupying locations 358-360 contains nine bytes, the length coded in the assembler format is equal to the number of moves (one less than the field length).*
2. *The order of operands is important even though only one field is involved.*

Move (MVI)

The MOVE (MVI) instruction places one byte of information from the instruction stream into storage. For example, the instruction:

Machine Format

Op Code	I_2	B_1	D_1
92	5B	1	000

Assembler Format

Op Code	$D_1(B_1),I_2$
MVI	0(1),C'$'

may be used, in conjunction with the instruction EDIT AND MARK, to insert a dollar symbol at the storage address contained in general register 1 (see also the example for EDIT AND MARK).

MOVE LONG (MVCL)

The MOVE LONG (MVCL) instruction can be used for moving data in storage as in the first example of the MVC instruction, provided that the two operands do not overlap. MVCL differs from MVC in that the address and length of each

operand are specified in an even-odd pair of general registers. Consequently, MVCL can be used to move more than 256 bytes of data with one instruction. As an example, assume:

Register 2 contains 00 0A 00 00.
Register 3 contains 00 00 08 00.
Register 8 contains 00 06 00 00.
Register 9 contains 00 00 08 00.

Execution of the instruction:

Machine Format

Op Code	R_1	R_2
0E	8	2

Assembler Format

Op Code	R_1,R_2
MVCL	8,2

moves $2{,}048_{10}$ bytes from locations A0000-A07FF to location 60000-607FF. Condition code 0 is set to indicate that the operand lengths are equal.

If register 3 had contained F0 00 04 00, only the $1{,}024_{10}$ bytes from locations A0000-A03FF would have been moved to locations 60000-603FF. The remaining locations 60400-607FF of the first operand would have been filled with 1,024 copies of the padding byte X'F0', as specified by the leftmost byte of register 3. Condition code 2 is set to indicate that the first operand is longer than the second.

The technique for setting a field to zeros that is illustrated in the second example of MVC cannot be used with MVCL. If the registers were set up to attempt such an operation with MVCL, no data movement would take place and condition code 3 would indicate destructive overlap.

Instead, MVCL may be used to clear a storage area to zeros as follows. Assume register 8 and 9 are set up as before. Register 3 contains only zeros, specifying zero length for the second operand and a zero padding byte. The contents of register 2 are not significant. Executing the instruction MVCL 8,2 causes locations 60000-607FF to be filled with zeros. Condition code 2 is set.

MOVE NUMERICS (MVN)

Two related instructions, MOVE NUMERICS and MOVE ZONES, may be used with decimal data in the zoned format to operate separately on the rightmost four bits (the numeric bits) and the leftmost four bits (the zone bits) of each byte. Both are similar to MOVE (MVC), except that MOVE NUMERICS moves only the numeric bits and MOVE ZONES moves only the zone bits.

To illustrate the operation of the MOVE NUMERICS instruction, assume that the following two fields are in storage:

	7090			7093
Field A	C6	C7	C8	C9

	7041					7046
Field B	F0	F1	F2	F3	F4	F5

Also assume:

Register 14 contains 00 00 70 90.
Register 15 contains 00 00 70 40.

After the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D1	03	F	001	E	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
MVN	1(4,15),0(14)

is executed, field B becomes:

7041					7046
F6	F7	F8	F9	F4	F5

The numeric bits of the bytes at locations 7090-7093 have been stored in the numeric bits of the bytes at locations 7041-7044. The contents of locations 7090-7093 and 7045-7046 are unchanged.

MOVE WITH OFFSET (MVO)

MOVE WITH OFFSET may be used to shift a packed-decimal number an odd number of digit positions or to concatenate a sign to an unsigned packed-decimal number.

Assume that the three-byte unsigned packed-decimal number in storage locations 4500-4502 is to be moved to locations 5600-5603

and given the sign of the packed-decimal number ending at location 5603. Also assume:

Register 12 contains 00 00 56 00.
Register 15 contains 00 00 45 00.
Storage locations 5600-5603 contain 77 88 99 0C.
Storage locations 4500-4502 contain 12 34 56.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F1	3	2	C	000	F	000

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
MVO	0(4,12),0(3,15)

is executed, the storage locations 5600-5603 contain 01 23 45 6C. Note that the second operand is extended on the left with one zero to fill out the first-operand field.

MOVE ZONES (MVZ)

The MOVE ZONES instruction can, similarly to MOVE (MVC) and MOVE NUMERICS, operate on overlapping or nonoverlapping fields. When operating on nonoverlapping fields, MOVE ZONES works like the MOVE NUMERICS instruction (see the example for MOVE NUMERICS), except that MOVE ZONES moves only the zone bits of each byte. To illustrate the use of MOVE ZONES with overlapping fields, assume that the following data field is in storage:

800					805
F1	C2	F3	C4	F5	C6

Also assume that register 15 contains 00 00 08 00. The instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D3	04	F	001	F	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
MVZ	1(5,15),0(15)

propagates the zone bits from the byte at address 800 through the entire field, so that the field becomes:

800					805
F1	F2	F3	F4	F5	F6

MULTIPLY (M, MR)

Assume that a number in register 5 is to be multiplied by the contents of a 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 06 00.
Register 12 contains 00 00 30 00.
Storage locations 3750-3753 contain 00 00 00 83 = 131_{10} = the multiplier.

The instruction required for performing the multiplication is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
5C	4	B	C	150

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
M	4,X'150'(11,12)

After the instruction is executed, the product is in the register pair 4 and 5:

Register 4 contains 00 00 00 00.
Register 5 contains 00 00 4E CE = $20,174_{10}$.
Storage locations 3750-3753 are unchanged.

The RR format of the instruction can be used to square the number in a register. Assume that register 7 contains 00 01 00 05. The instruction:

Machine Format

Op Code	R_1	R_2
1C	6	7

Assembler Format

Op Code	R_1,R_2
MR	6,7

multiplies the number in register 7 by itself and places the result in the pair of registers 6 and 7:

Register 6 contains 00 00 00 01.
Register 7 contains 00 0A 00 19.

MULTIPLY HALFWORD (MH)

The MULTIPLY HALFWORD instruction is used to multiply the contents of a register by a 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 = the multiplier.

The instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4C	B	E	F	002

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
MH	11,2(14,15)

multiplies the two numbers. The product, FF FF FC CD = -819_{10}, replaces the original contents of register 11.

Only the rightmost 32 bits of a product are stored in a register; any significant bits on the left are lost. No program interruption occurs on overflow.

OR (O, OR, OI, OC)

When the Boolean operator OR is applied to two bits, the result is one when either bit is one; otherwise, the result is zero. When two bytes are ORed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of ORing two bytes:

First-operand byte:	$0011\ 0101_2$
Second-operand byte:	$0101\ 1100_2$
Result byte:	$0111\ 1101_2$

Or (OI)

A frequent use of the OR instruction is to set a particular bit to one. For example, assume that storage location 4891 contains $0100\ 0010_2$. To set the rightmost bit of this byte to one without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

Op Code	I_2	B_1	D_1
96	01	8	001

Assembler Format

Op Code	$D_1(B_1),I_2$
OI	1(8),X'01'

When this instruction is executed, the byte in storage is ORed with the immediate byte (the I_2 field of the instruction):

Location 4891:	$0100\ 0010_2$
Immediate byte:	$0000\ 0001_2$
Result:	$0100\ 0011_2$

The resulting byte with bit 7 set to one is stored back in location 4891. Condition code 1 is set.

PACK (PACK)

Assume that storage locations 1000-1003 contain the following zoned-decimal number that is to be converted to a packed-decimal number and left in the same location:

	1000			1003
Zoned number	F1	F2	F3	C4

Also assume that register 12 contains 00 00 10 00. After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F2	3	3	C	000	C	000

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
PACK	0(4,12),0(4,12)

is executed, the result in locations 1000-1003 is in the packed-decimal format:

	1000			1003
Packed number	00	01	23	4C

Notes:

1. *This example illustrates the operation of PACK when the first- and second-operand fields overlap completely.*
2. *During the operation, the second operand was extended on the left with zeros.*

SHIFT LEFT DOUBLE (SLDA)

The SHIFT LEFT DOUBLE instruction 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 1000₂
```

The instruction:

Machine Format

Op Code	R_1	R_3	B_2	D_2
8F	2	////	0	01F

Assembler Format

Op Code	R_1,D_2(B_2)
SLDA	2,31(0)

results in registers 2 and 3 both being left-shifted 31 bit positions, so that their new contents are:

```
7F 6E 5D 4C   00 00 00 00 =
0111 1111 0110 1110 0101 1101 0100 1100
0000 0000 0000 0000 0000 0000 0000 0000₂
```

In this case, a significant bit is shifted out of bit position 1 of register 2. Condition code 3 is set to indicate this overflow and, if the fixed-point-overflow mask bit in the PSW is one, a fixed-point overflow interruption occurs.

SHIFT LEFT SINGLE (SLA)

Because SHIFT LEFT SINGLE leaves the sign bit unchanged, 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 0010₂
```

The instruction:

Machine Format

Op Code	R_1	R_3	B_2	D_2
8B	2	////	0	008

Assembler Format

Op Code	R_1,D_2(B_2)
SLA	2,8(0)

results in register 2 being shifted left eight bit positions so that its new contents are:

```
7F 0A 72 00 =
0111 1111 0000 1010 0111 0010 0000 0000₂
```

Condition code 2 is set to indicate that the result is nonzero and positive.

If a left shift of nine places had been specified, a significant bit would have been shifted out of bit position 1. Condition code 3 would have been set to indicate this overflow and, if the fixed-point-overflow mask bit in the PSW is one, a fixed-point overflow interruption would have occurred.

STORE CHARACTERS UNDER MASK (STCM)

STORE CHARACTERS UNDER MASK (STCM) may be used to place selected bytes from a register into storage. For example, if it is desired to store a three-byte address from general register 8 into location FIELD3, assume:

Machine Format

Op Code	R_1	M_3	S_2
BE	8	7	* * * *

Register Format

Op Code	R_1,M_3,S_2
STCM	8,B'0111',FIELD3

```
Register 8:        12 34 56 78
FIELD3 (before):   not significant
FIELD3 (after):    34 56 78
```

As another example:

Machine Format

Op Code	R_1	M_3	S_2
BE	9	5	* * * *

Register Format

Op Code	R_1,M_3,S_2
STCM	9,B'0101',FIELD2

```
Register 9:        01 23 45 67
FIELD2 (before):   not significant
FIELD2 (after):    23 67
```

STORE MULTIPLE (STM)

Assume that the contents of general registers 14, 15, 0, and 1 are to be stored in consecutive words starting with location 4050 and that:

Register 14 contains 00 00 25 63.
Register 15 contains 00 01 27 36.
Register 0 contains 12 43 00 62.
Register 1 contains 73 26 12 57.
Register 6 contains 00 00 40 00.
The initial contents of locations 4050-405F are not significant.

The STORE MULTIPLE instruction allows the use of just one instruction to store the contents of the four registers:

Machine Format

Op Code	R_1	R_3	B_2	D_2
90	E	1	6	050

Assembler Format

Op Code	R_1,R_3,D_2(B_2)
STM	14,1,X'50'(6)

After the instruction is executed:

Locations 4050-4053 contain 00 00 25 63.
Locations 4054-4057 contain 00 01 27 36.
Locations 4058-405B contain 12 43 00 62.
Locations 405C-405F contain 73 26 12 57.

TEST UNDER MASK (TM)

The TEST UNDER MASK instruction examines selected bits of a byte and sets the condition code accordingly. For example, assume that:

Storage location 9999 contains FB.
Register 7 contains 00 00 99 90.

Execution of the instruction:

Machine Format

Op Code	I_2	B_1	D_1
91	C3	7	009

Assembler Format

Op Code	D_1(B_1),I_2
TM	9(7),B'11000011'

produces the following result:

```
FB      = 1111 1011₂
Mask    = 1100 0011₂
-------------------
Result  = 11xx xx11₂
```

Condition code 3 is set: all selected bits are ones.

If location 9999 had contained B9, the result would have been:

```
B9      = 1011 1001₂
Mask    = 1100 0011₂
-------------------
Result  = 10xx xx01₂
```

Condition code 1 is set: the selected bits are both zeros and ones.

If location 9999 had contained 3C, the result would have been:

```
3C      = 0011 1100₂
Mask    = 1100 0011₂
-------------------
Result  = 00xx xx00₂
```

Condition code 0 is set: all selected bits are zeros.

Note: *Storage location 9999 remains unchanged.*

TRANSLATE (TR)

The TRANSLATE instruction can be used to translate data from any character code to any other desired code, provided that each coded character

consists of eight bits or fewer. In the following example, EBCDIC is translated to ASCII. The first step is to create a 256-byte table in storage locations 1000-10FF. This table contains the characters of the target code in the sequence of the binary representation of the source code; that is, the ASCII representation of a character is placed in storage at the starting address of the table plus the binary value of the EBCDIC representation of the same character. For simplicity, the example shows only the part of the table containing the decimal digits:

Translate Table for Decimal Digits:

10F0									10F9
30	31	32	33	34	35	36	37	38	39

Assume that the four-byte field at storage location 2100 contains the EBCDIC code for the digits 1984:

Locations 2100-2103 contain F1 F9 F8 F4.
Register 12 contains 00 00 21 00.
Register 15 contains 00 00 10 00.

As the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DC	03	C	000	F	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
TR	0(4,12),0(15)

is executed, the binary value of each source byte is added to the starting address of the table, and the resulting address is used to fetch a target byte:

Table starting address:	1000
First source byte:	F1
Address of target byte:	10F1

After execution of the instruction:

Locations 2100-2103 contain 31 39 38 34.

Thus, the ASCII code for the digits 1984 has replaced the EBCDIC code in the four-byte field at storage location 2100.

TRANSLATE AND TEST (TRT)

The TRANSLATE AND TEST instruction can be used to scan a data field for characters with a special meaning. To indicate which characters have a special meaning, a table similar to the one used for the TRANSLATE instruction is set up, except that zeros in the table indicate characters without any special meaning and nonzero values indicate characters with a special meaning.

The figure "Translate-and-Test Table" that follows has been set up to distinguish alphameric characters (A to Z and 0 to 9) from blanks, certain special symbols, and all other characters which are considered invalid. EBCDIC coding is assumed. The 256-byte table is assumed stored at locations 2000-20FF.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
200_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
201_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
202_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
203_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
204_	04	40	40	40	40	40	40	40	40	40	40	08	40	0C	10	40
205_	14	40	40	40	40	40	40	40	40	40	40	18	1C	20	40	40
206_	24	28	40	40	40	40	40	40	40	40	40	2C	40	40	40	40
207_	40	40	40	40	40	40	40	40	40	40	40	30	34	38	3C	40
208_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
209_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20A_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20B_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20C_	40	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20D_	40	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20E_	40	40	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20F_	00	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40

Note: If the character codes in the statement being translated occupy a range smaller than 00 through FF_{16}, a table of fewer than 256 bytes can be used.

Translate and Test Table

The table entries for the alphameric characters in EBCDIC are 00; thus, the letter A (code C1) corresponds to byte location 20C1, which contains 00.

The 15 special symbols have nonzero entries from 04_{16} to $3C_{16}$ in increments of 4. Thus, the blank (code 40) has the entry 04_{16}, the period (code 4B) has the entry 08_{16}, and so on.

All other table positions have the entry 40_{16} to indicate an invalid character.

The table entries are chosen so that they may be used to select one of a list of 16 words containing addresses of different routines to be entered for each special symbol or invalid character encountered during the scan.

Assume that this list of 16 branch addresses is stored at locations 3004-3043.

Starting at storage location CA80, there is the following sequence of 21_{10} EBCDIC characters:

Locations CA80-CA94: UNPKbPROUT(9),WORD(5)

Also assume:

Register 1 contains 00 00 2F FF.
Register 2 contains 00 00 30 00.
Register 15 contains 00 00 20 00.

As the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DD	14	1	001	F	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
TRT	1(21,1),0(15)

is executed, the value of the first argument byte, the letter U, is added to the starting address of the table to produce the address of the table entry to be examined:

Table starting address	2000
First argument byte (U)	E4
Address of table entry	20E4

Because zeros were placed in storage location 20E4, no special action occurs. The operation continues with the second and subsequent argument bytes until it reaches the blank in location CA84. When this symbol is reached, its value is added to the starting address of the table, as usual:

Table starting address	2000
Argument byte (blank)	40
Address of table entry	2040

Because location 2040 contains a nonzero value, the following actions occur:

1. The address of the argument byte, 00CA84, is placed in the rightmost 24 bits of register 1.
2. The table entry, 04, is placed in the rightmost eight bits of register 2, which now contains 00 00 30 04.
3. Condition code 1 is set (scan not completed).

The TRANSLATE AND TEST instruction may be followed by instructions to branch to the routine at the address found at location 3004, which corresponds to the blank character encountered in the scan. When this routine is completed, program control may return to the TRANSLATE AND TEST instruction to continue the scan, except that the length must first be adjusted for the characters already scanned.

For this purpose, the TRANSLATE AND TEST may be executed by the use of an EXECUTE instruction, which supplies the length specification from a general register. In this way, a complete statement scan can be performed with a single TRANSLATE AND TEST instruction repeated over and over by means of EXECUTE, and without modifying any instructions in storage. In the example, after the first execution of TRANSLATE AND TEST register 1 contains the address of the last argument byte translated. It is then a simple matter to subtract this address from the address of the last argument byte (CA94) to produce a length specification. This length minus one is placed in the register that is referenced as the R1 field of the EXECUTE instruction. (Note that the length code in the machine format is one less than the total number of bytes in the field.) The second-operand address of the EXECUTE instruction points to the TRANSLATE AND TEST instruction, which is the same as illustrated above, except for the length (L) which is set to zero.

UNPACK (UNPK)

Assume that storage locations 2501-2502 contain a signed, packed-decimal number that is to be unpacked and placed in storage locations 1000-1004. Also assume:

Register 12 contains 00 00 10 00.
Register 13 contains 00 00 25 00.
Storage locations 2501-2502 contain 12 3D.
The initial contents of storage locations 1000-1004 are not significant.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F3	4	1	C	000	D	001

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
UNPK	0(5,12),1(2,13)

is executed, the storage locations 1000-1004 contain F0 F0 F1 F2 D3.

Decimal Instructions

(See Chapter 8.)

ADD DECIMAL (AP)

Assume that the signed, packed-decimal number at storage locations 500-503 is to be added to the signed, packed-decimal number at locations 2000-2002. Also assume:

Register 12 contains 00 00 20 00.
Register 13 contains 00 00 05 00.
Storage locations 2000-2002 contain 38 46 0D (a negative number).
Storage locations 500-503 contain 01 12 34 5C (a positive number).

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
FA	2	3	C	000	D	000

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
AP	0(3,12),0(4,13)

is executed, the storage locations 2000-2002 contain 73 88 5C; condition code 2 is set to indicate that the sum is positive. Note that:

1. Because the two numbers had different signs, they were in effect subtracted.
2. Although the second operand is longer than the first operand, no overflow interruption occurs because the result can be entirely contained within the first operand.

COMPARE DECIMAL (CP)

Assume that the signed, packed-decimal contents of storage locations 700-703 are to be algebraically compared with the signed, packed-decimal contents of locations 500-502. Also assume:

Register 12 contains 00 00 06 00.
Register 13 contains 00 00 03 00.
Storage locations 700-703 contain 17 25 35 6D.
Storage locations 500-502 contain 72 14 2D.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F9	3	2	C	100	D	200

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
CP	X'100'(4,12),X'200'(3,13)

is executed, condition code 1 is set, indicating that the first operand (the contents of locations 700-703) is less than the second.

DIVIDE DECIMAL (DP)

Assume that the signed, packed-decimal number at storage locations 2000-2004 (the dividend) is to be divided by the signed, packed-decimal number at locations 3000-3001 (the divisor). Also assume:

Register 12 contains 00 00 20 00.
Register 13 contains 00 00 30 00.
Storage locations 2000-2004 contain 01 23 45 67 8C.
Storage locations 3000-3001 contain 32 1D.

After the instruction:

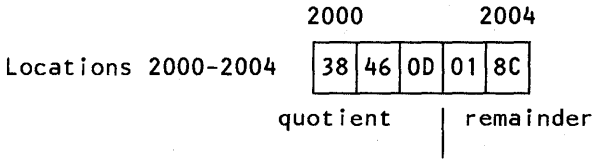
Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
FD	4	1	C	000	D	000

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
DP	0(5,12),0(2,13)

is executed, the dividend is entirely replaced by the signed quotient and remainder, as follows:



Notes:

1. *Because the dividend and divisor have different signs, the quotient receives a negative sign.*
2. *The remainder receives the sign of the dividend and the length of the divisor.*
3. *If an attempt were made to divide the dividend by the one-byte field at location 3001, the quotient would be too long to fit within the four bytes allotted to it. A decimal-divide exception would exist, causing a program interruption.*

EDIT (ED)

Before decimal data in the packed format can be used in a printed report, digits and signs must be converted to printable characters. Moreover, punctuation marks, such as commas and decimal points, may have to be inserted in appropriate places. The highly flexible EDIT instruction performs these functions in a single instruction execution.

This example shows step-by-step one way that the EDIT instruction can be used. The field to be edited (the source) is four bytes long; it is edited against a pattern 13 bytes long. The following symbols are used:

Symbol	Meaning
b (Hexadecimal 40)	Blank character
((Hexadecimal 21)	Significance starter
d (Hexadecimal 20)	Digit selector

Assume that the source and pattern fields are:

Source

1200 1203

02	57	42	6C

+

Pattern

1000 100C

40	20	20	6B	20	21	20	4B	20	20	40	C3	D9
b	d	d	,	d	(	d	.	d	d	b	C	R

Execution of the instruction (assume that register 12 contains 00 00 10 00):

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DE	0C	C	000	C	200

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
ED	0(13,12),X'200'(12)

alters the pattern field as follows:

Pattern	Digit	Significance Indicator (Before/After)	Rule	Location 1000-100C
b		off/off	leave(1)	bdd,d(d.ddbCR
d	0	off/off	fill	bbd,d(d.ddbCR
d	2	off/on(2)	digit	bb2,d(d.ddbCR
,		on/on	leave	same
d	5	on/on	digit	bb2,5(d.ddbCR
(	7	on/on	digit	bb2,57d.ddbCR
d	4	on/on	digit	bb2,574.ddbCR
.		on/on	leave	same
d	2	on/on	digit	bb2,574.2dbCR
d	6+	on/off(3)	digit	bb2,574.26bCR
b		off/off	fill	same
C		off/off	fill	bb2,574.26bbR
R		off/off	fill	bb2,574.26bbb

Notes:

1. This character is the fill byte.
2. First nonzero decimal source digit turns on significance indicator.
3. Plus sign in the four rightmost bits of the byte turns off significance indicator.

Thus, after the instruction is executed, the pattern field contains the result as follows:

Pattern

1000 100C

40	40	F2	6B	F5	F7	F4	4B	F2	F6	40	40	40
b	b	2	,	5	7	4	.	2	6	b	b	b

When printed, the new pattern field appears as:

2,574.26

The source field remains unchanged. Condition code 2 is set because the number was greater than zero.

If the number in the source field is changed to 00 00 02 6D, a negative number, and the original pattern is used, the edited result this time is:

Pattern

1000 100C

40	40	40	40	40	40	F0	4B	F2	F6	40	C3	D9
b	b	b	b	b	b	0	.	2	6	b	C	R

This pattern field prints as:

0.26 CR

The significance starter forces the significance indicator to the on state and hence causes the decimal point to be preserved. Because the minus-sign code has no effect on the significance indicator, the characters CR are printed to show a negative (credit) amount.

Condition code 1 is set (number less than zero).

EDIT AND MARK (EDMK)

The EDIT AND MARK instruction may be used, in addition to the functions of EDIT, to insert a currency symbol, such as a dollar sign, at the appropriate position in the edited result. Assume the same source in storage locations 1200-1203, the same pattern in locations 1000-100C, and the same contents of general register 12 as for the EDIT instruction above. The previous contents of general register 1 are immaterial; a LOAD ADDRESS instruction is used to set up the first digit position that is forced to print if no significant digits occur to the left.

The instructions:

LA 1,6(0,12)	Load address of forced significant digit into GR1.
EDMK0(13,12),X'200'(12)	Leave address of first significant digit in GR1.
BCTR1,0	Subtract 1 from address in GR1.
MVI 0(1),C'$'	Store dollar sign and address in GR1.

produce the following results for the two examples under EDIT:

Pattern

1000											100C
40	5B	F2	6B	F5	F7	F4	4B	F2	F6	40	40	40
b	$	2	,	5	7	4	.	2	6	b	b	b

This pattern field prints as:

$2,574.26

Condition code 2 is set to indicate that the number edited was greater than zero.

Pattern

1000											100C
40	40	40	40	40	5B	F0	4B	F2	F6	40	C3	D9
b	b	b	b	b	$	0	.	2	6	b	C	R

This pattern field prints as:

$0.26 CR

Condition code 1 is set because the number is less than zero.

MULTIPLY DECIMAL (MP)

Assume that the signed, packed-decimal number in storage locations 1202-1204 (the multiplicand) is to be multiplied by the signed, packed-decimal number in locations 500-501 (the multiplier).

	1202		1204
Multiplicand	38	46	0D

	500	501
Multiplier	32	1D

Because the multiplier and multiplicand have a total of eight significant digits, at least five bytes must be reserved for the signed result. ZERO AND ADD can be used to move the multiplicand into a longer field. Assume:

Register 4 contains 00 00 12 00.
Register 6 contains 00 00 05 00.

Then execution of the instruction:

ZAP X'100'(5,4),2(3,4)

sets up a new multiplicand in storage locations 1300-1304:

	1300				1304
Multiplicand (new)	00	00	38	46	0D

Now, after the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
FC	4	1	4	100	6	000

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
MP	X'100'(5,4),0(2,6)

is executed, storage locations 1300-1304 contain the product:

01 23 45 66 0C.

SHIFT AND ROUND DECIMAL (SRP)

The SHIFT AND ROUND DECIMAL (SRP) instruction can be used for shifting decimal numbers in storage to the left or right. When a number is shifted right, rounding can also be done.

Decimal Left Shift

In this example, the contents of storage location FIELD1 are shifted three places to the left, effectively multiplying the contents of FIELD1 by 1000. FIELD1 is six bytes long. The following instruction performs the operation:

Machine Format

Op Code	L_1	I_3	S_1	B_2	D_2
F0	5	0	****	0	003

Assembler Format

Op Code	$S_1(L_1),S_2,I_3$
SRP	FIELD1(6),3,0

FIELD1 (before): 00 01 23 45 67 8C

FIELD1 (after): 12 34 56 78 00 0C

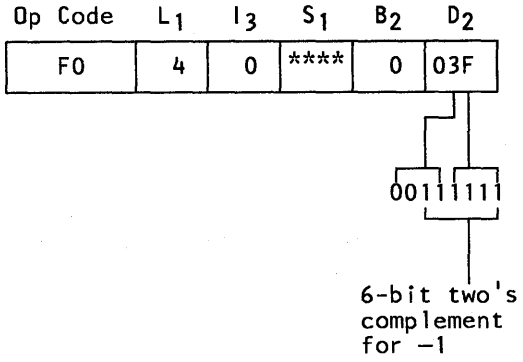
The second-operand address in this instruction specifies the shift amount (three places). The rounding factor, I3, is not used in left shift, but it

must be a valid decimal digit. After execution, condition code 2 is set to show that the result is greater than zero.

Decimal Right Shift

In this example, the contents of storage location FIELD2 are shifted one place to the right, effectively dividing the contents of FIELD2 by 10 and discarding the remainder. FIELD2 is five bytes in length. The following instruction performs this operation:

Machine Format



Assembler Format

Op Code	$S_1(L_1),S_2,I_3$
SRP	FIELD2(5),64−1,0

```
FIELD 2 (before):  01 23 45 67 8C

FIELD 2 (after):   00 12 34 56 7C
```

In the SRP instruction, shifts to the right are specified in the second-operand address by negative shift values, which are represented as a six-bit value in two's complement form.

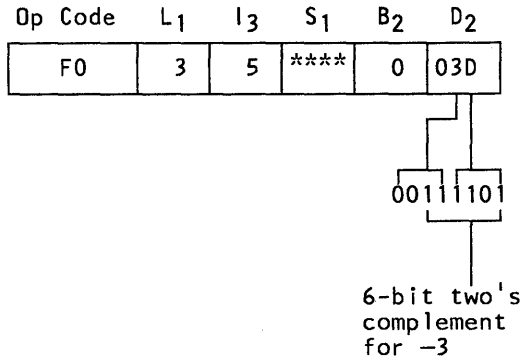
The six-bit two's complement of a number, n, can be specified as 64 − n. In this example, a right shift of one is represented as 64 − 1.

Condition code 2 is set.

Decimal Right Shift and Round

In this example, the contents of storage location FIELD3 are shifted three places to the right and rounded, effectively dividing by 1000 and rounding to the nearest whole number. FIELD3 is four bytes in length.

Machine Format



Assembler Format

Op Code	$S_1(L_1),S_2,I_3$
SRP	FIELD3(4),64−3,5

```
FIELD 3 (before):  12 39 60 0D

FIELD 3 (after):   00 01 24 0D
```

The shift amount (three places) is specified in the D_2 field. The I_3 field specifies the rounding factor of 5. The rounding factor is added to the last digit shifted out (which is a 6), and the carry is propagated to the left. The sign is ignored during the addition.

Condition code 1 is set because the result is less than zero.

Multiplying by a Variable Power of 10

Since the shift value designated by the SRP instruction specifies both the direction and amount of the shift, the operation is equivalent to multiplying the decimal first operand by 10 raised to the power specified by the shift value.

If the shift value is variable, it may be specified by the B_2 field instead of the displacement D_2 of the SRP instruction. The general register designated by B_2 should contain the shift value (power of 10) as a signed binary integer.

A fixed scale factor modifying the variable power of 10 may be specified by using both the B_2 field (variable part in a general register) and the D_2 field (fixed part in the displacement).

The SRP instruction uses only the rightmost six bits of the effective address $D_2(B_2)$ and interprets them as a six-bit signed binary integer to control the left or right shift as in the previous two examples.

ZERO AND ADD (ZAP)

Assume that the signed, packed-decimal number at storage locations 4500-4502 is to be moved to locations 4000-4004 with four leading zeros in the result field. Also assume:

Register 9 contains 00 00 40 00.
Storage locations 4000-4004 contain 12 34 56 78 90.
Storage locations 4500-4502 contain 38 46 0D.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F8	4	2	9	000	9	500

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
ZAP	0(5,9),X'500'(3,9)

is executed, the storage locations 4000-4004 contain 00 00 38 46 0D; condition code 1 is set to indicate a negative result.

Note that, because the first operand is not checked for valid sign and digit codes, it may contain any combination of hexadecimal digits before the operation.

Floating-Point Instructions

(See Chapter 9.)

In this section, the abbreviations FPR0, FPR2, FPR4, and FPR6 stand for floating-point registers 0, 2, 4, and 6 respectively.

ADD NORMALIZED (AD, ADR, AE, AER, AXR)

The ADD NORMALIZED instructions perform the addition of two floating-point numbers and place the normalized result in a floating-point register. Neither of the two numbers to be added must necessarily be normalized before addition occurs. For example, assume that:

FPR6 contains C3 08 21 00 00 00 00 00 = -82.1_{16} = -130.06_{10} approximately (unnormalized).
Storage locations 2000-2007 contain 41 12 34 56 00 00 00 00 = $+1.23456_{16}$ = $+1.14_{10}$ (normalized).
Register 13 contains 00 00 20 00.

The instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
7A	6	0	D	000

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
AE	6,0(0,13)

performs the short-precision addition of the two operands, as follows.

The characteristics of the two numbers (43 and 41) are compared. Since the number in storage has a characteristic that is smaller by 2, it is right-shifted two hexadecimal digit positions. The two numbers are then added:

```
                                          GD[1]
FPR6:                      -43 08 21 00
Shifted no. from storage:  +43 00 12 34   5
-------------------------------------------------
Intermediate sum:          -43 08 0E CB   B
[1]Guard digit
```

Because the intermediate sum is unnormalized, it is left-shifted to form the normalized floating-point number –42 80 EC BB = $-80.ECBB_{16}$ = –128.92. Combining the sign with the characteristic, the result is C2 80 EC BB, which replaces the left half of FPR6. The right half of FPR6 and the contents of storage locations 2000-2007 are unchanged. Condition code 1 is set to indicate a negative result.

If the long-precision instruction AD is used, the result in FPR6 is C2 80 EC BA A0 00 00 00. Note that the long-precision instruction avoids a loss of precision in this example.

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. For example, using the the same operands as in the example for ADD NORMALIZED, when the short-precision instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
7E	6	0	D	000

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
AU	6,0(0,13)

is executed, the two numbers are added as follows:

```
                                        GD[1]
FPR6:                        -43 08 21 00
Shifted no. from storage:    +43 00 12 34  5
-------------------------------------------
Sum:                         -43 08 0E CB  B
[1]Guard digit
```

The guard digit participates in the addition but is discarded. The unnormalized sum replaces the left half of FPR6. Condition code 1 is set because the result is negative.

The result in FPR6 (C3 08 0E CB 00 00 00 00) shows a loss of a significant digit when compared to the result of short-precision normalized addition.

COMPARE (CD, CDR, CE, CER)

Assume that FPR4 contains 43 00 00 00 00 00 00 00 (=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 using a long-precision COMPARE instruction.

Machine Format

Op Code	R_1	R_2
29	4	6

Assembler Format

```
Op Code  R1,R2
-------------
  CDR    4,6
```

The number with the smaller characteristic, which is the one in register FPR6, is right-shifted 15 hexadecimal digit positions so that the two characteristics agree. The shifted contents of FPR6 are 43 00 00 00 00 00 00 00, with a guard digit of zero. Therefore, when the two numbers are compared, condition code 0 is set, indicating an equality.

As the above example implies, when floating-point numbers are compared, more than two numbers may compare equal if one of the numbers is unnormalized. For example, the unnormalized floating-point number 41 00 12 34 56 78 9A BC compares equal to all numbers of the form 3F 12 34 56 78 9A BC 0X (X represents any hexadecimal digit). When the COMPARE instruction is executed, the two rightmost digits are shifted right two places, the 0 becomes the guard digit, and the X does not participate in the comparison.

However, when two normalized floating-point numbers are compared, the relationship between numbers that compare equal is unique: each digit in one number must be identical to the corresponding digit in the other number.

Floating-Point-Number Conversion

The following examples illustrate one method of converting between binary fixed-point numbers (32-bit signed binary integers) and normalized floating-point numbers. Conversion must provide for the different representations used with negative numbers: the two's-complement form for signed binary integers, and the signed-absolute-value form for the fractions of floating-point numbers.

Fixed Point to Floating Point

The method used here inverts the leftmost bit of the signed binary integer which, after appending additional zero bits on the left as necessary, is equivalent to adding 2^{31} to the number. This changes it from a signed integer in the range $2^{31} - 1$ through -2^{31} to an unsigned integer in the range $2^{32} - 1$ through 0. After conversion to the long floating-point format, the value 2^{31} is subtracted again.

Assume that general register 9 (GR9) contains the integer –59 in two's-complement form:

```
GR9     FF FF FF C5
```

Further, assume two eight-byte fields in storage: TEMP, for use as temporary storage, and TWO31, which contains the floating-point constant 2^{31} in the following format:

```
TWO31   4E 00 00 00 80 00 00 00
```

This is an unnormalized long floating-point number with the characteristic 4E, which corresponds to a radix point to the right of the number.

The following instruction sequence performs the conversion:

```
                          Result
X   9,TWO31+4             GR9:
                          7F FF FF C5
ST  9,TEMP+4              TEMP:
                          -  -  -  -  7F FF FF C5
MVC TEMP(4),TWO31         TEMP:
                          4E 00 00 00 7F FF FF C5
LD  2,TEMP                FPR2:
                          4E 00 00 00 7F FF FF C5
SD  2,TWO31               FPR2:
                          C2 3B 00 00 00 00 00 00
```

The EXCLUSIVE OR (X) instruction inverts the leftmost bit in general register 9, using the right half of the constant as the source for a leftmost one

bit. The next two instructions assemble the modified number in an unnormalized long floating-point format, using the left half of the constant as the plus sign, the characteristic, and the leading zeros of the fraction. LOAD (LD) places the number unchanged in floating-point register 2. The SUBTRACT NORMALIZED (SD) instruction performs the final two steps by subtracting 2^{31} in floating-point form and normalizing the result.

Floating Point to Fixed Point

The procedure described here consists basically in reversing the steps of the previous procedure. Two additional considerations must be taken into account. First: the floating-point number may not be an exact integer. Truncating the excess hexadecimal digits on the right requires shifting the number one digit position farther to the right than desired for the final result, so that the units digit occupies the position of the guard digit Second: the floating-point number may have to be tested as to whether it is outside the range of numbers representable as a signed binary integer.

Assume that floating-point register 6 contains the number $59.25_{10} = 3B.4_{16}$ in normalized form:

```
FPR6     42 3B 40 00 00 00 00 00
```

Further, assume three eight-byte fields in storage: TEMP, for use as temporary storage, and the constants 2^{32} (TWO32) and 2^{31} (TWO31R) in the following formats:

```
TWO32    4E 00 00 01 00 00 00 00
TWO31R   4F 00 00 00 08 00 00 00
```

The constant TWO31R is shifted right one more position than the constant TWO31 of the previous example, so as to force the units digit into the guard-digit position.

The following instruction sequence performs the integer truncation, range tests, and conversion to a signed binary integer in general register 8 (GR8):

```
                          Result
SD   6,TWO31R             FPR6:
                          C8 7F FF FF C5 00 00 00
BC   11,OVERFLOW          Branch to overflow
                            routine if result
                            non-negative
AW   6,TWO32              FPR6:
                          4E 00 00 00 80 00 00 3B
BC   4,OVERFLOW           Branch to overflow
                            routine if result
                            negative
STD  6,TEMP               TEMP:
                          4E 00 00 00 80 00 00 3B
XI   TEMP+4,X'80'         TEMP:
                          4E 00 00 00 00 00 00 3B
L    8,TEMP+4             GR8:
                          00 00 00 03
```

The SUBTRACT NORMALIZED (SD) instruction shifts the fraction of the number to the right until it lines up with TWO31R, which causes the fraction digit 4 to fall to the right of the guard digit and be lost; the result of subtracting 2^{31} from the remaining digits is renormalized. The result should be negative; if not, the original number was too large in the positive direction. The first BRANCH ON CONDITION (BC) performs this test.

The ADD UNNORMALIZED (AW) instruction adds 2^{32}: 2^{31} to correct for the previous subtraction and another 2^{31} to change to an all-positive range. The second BC tests for a negative result, showing that the number was too large in the negative direction. The unnormalized result is placed in temporary storage by the STORE (STD) instruction. There the leftmost bit of the binary integer is inverted by the EXCLUSIVE OR (XI) instruction before being loaded into GR8.

Multiprogramming and Multiprocessing Examples

Although the 4300 Processors make no provision for multiple-CPU systems, the references to multiprocessing may be helpful when writing problem-state programs that are to run correctly on multiprocessing configurations of System/370 as well.

When two or more programs sharing common storage locations are running concurrently in a multiprogramming or multiprocessing environment, one program may, for example, set a flag bit in the common-storage area for testing by another program. It should be noted that the instructions AND (NI or NC), EXCLUSIVE OR (XI or XC), and OR (OI or OC) could be used to set flag bits in a multiprogramming environment; but the same instructions may cause program logic errors in a multiprocessing system where two or more CPUs can fetch, modify, and store data in the same storage locations simultaneously.

Example of a Program Failure Using OR Immediate

Assume that two independent programs try to set different bits to one in a common byte in storage. The following example shows how the use of the instruction OR immediate (OI) can fail to accomplish this, if the programs are executed nearly simultaneously on two different CPUs. One of the possible error situations is depicted.

Execution of instruction OI FLAGS,X'01' on CPU A	FLAGS	Execution of instruction OI FLAGS,X'80' on CPU B
	X'00'	Fetch FLAGS X'00'
Fetch FLAGS X'00'	X'00'	
	X'00'	OR X'80' into X'00'
OR X'01' into X'00'	X'00'	
	X'80'	Store X'80' into FLAGS
Store X'01' into FLAGS	X'01'	
FLAGS should have value of X'81' following both updates.		

The problem shown here is that the value stored by the OI instruction executed on CPU A overlays the value that was stored by CPU B. The X'80' flag bit was erroneously turned off, and the date is now invalid.

The COMPARE AND SWAP instruction has been provided to overcome this and similar problems.

COMPARE AND SWAP (CS, CDS)

The COMPARE AND SWAP (CS) and COMPARE DOUBLE AND SWAP (CDS) instructions can be used in multiprogramming or multiprocessing environments to serialize access to counters, flags, control words, and other common storage areas.

The following examples of the use of the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions illustrate the applications for which the instructions are intended. It is important to note that these are examples of functions that can be performed by programs running enabled for interruption (multiprogramming) or by programs that are running on a multiprocessing configuration. That is, the routine allows a program to modify the contents of a storage location while running enabled, even though the routine may be interrupted by another program on the same CPU that will update the location, and even though the possibility exists that another CPU may simultaneously update the same location.

The CS instruction first checks the value of a storage location and then modifies it only if the value is what the program expects; normally this would be a previously fetched value. If the value in storage is not what the program expects, then the location is not modified; instead, the current value of the location is loaded into a general register, in preparation for the program to loop back and try again. During the execution of CS, no other CPU can access the specified location.

Setting a Single Bit

The following instruction sequence shows how the CS instruction can be used to set a single bit in storage to one. Assume that FLAGS is the first byte of a word in storage called "WORD."

```
        LA   6,X'80'     Put bit to be ORed
                           into GR6
        SLL  6,24        Shift left 24 places to
                           align the byte to be
                           ORed with the loca-
                           tion of FLAGS within
                           WORD
        L    7,WORD      Get original flag bit
                           values
RETRY   LR   8,7         Put flags to be modi-
                           fied into GR8
        OR   8,6         Set bit to one in new
                           copy of flags
        CS   7,8,WORD    Store new flags unless
                           original flags were
                           changed
        BC   4,RETRY     If new flags are not
                           stored, try again
```

The format of the CS instruction is:

Machine Format

Op Code	R_1	R_3	S_2
BA	7	8	****

Assembler Format

Op Code	R_1,R_3,S_2
CS	7,8,WORD

The CS instruction compares the first operand (general register 7 containing the original flag values) to the second operand (WORD) while storage access to the specified location is not permitted to any CPU other than the one executing the CS instruction.

If the comparison is successful, indicating that FLAGS still has the same value that it originally had, the modified copy in general register 8 is stored into FLAGS. If FLAGS has changed since it was loaded, the compare will not be successful, and the current value of FLAGS is loaded into general register 7.

The CS instruction sets condition code 0 to indicate a successful compare and swap, and condition code 1 to indicate an unsuccessful compare and swap.

The program executing the sample instructions tests the condition code following the CS instruction and reexecutes the flag-modifying instructions if the CS instruction indicated an unsuccessful comparison. When the CS instruction is successful, the program continues execution outside the loop and FLAGS contains valid data.

The branch to RETRY will be taken only if some other program modifies the update location. This type of a loop differs from the typical "bit-spin" loop. In a bit-spin loop, the program continues to loop until the bit changes. In this example, the program continues to loop only if the value does change during each iteration. If a number of CPUs simultaneously attempt to modify a single location by using the sample instruction sequence, one CPU will fall through on the first try, another will loop once, and so on until all CPUs have succeeded.

Updating Counters

In this example, a 32-bit counter is updated by a program using the CS instruction to ensure that the counter will be correctly updated. The original value of the counter is obtained by loading the word containing the counter into general register 7. This value is moved into general register 8 to provide a modifiable copy, and general register 6 (containing an increment to the counter) is added to the modifiable copy to provide the updated counter value. The CS instruction is used to ensure valid storing of the counter.

The program updating the counter checks the result by examining the condition code. The condition code 0 indicates a successful update, and the program can proceed. If the counter had been changed between the time that the program loaded its original value and the time that it executed the CS instruction, the CS instruction would have loaded the new counter value into general register 7 and set the condition code to 1, indicating an unsuccessful update. The program then must update the new counter value in general register 7 and retry the CS instruction, retesting the condition code, and retrying until a successful update is completed.

The following instruction sequence performs the above procedure:

```
     LA   6,1        Put increment (1) into
                       GR6
     L    7,CNTR     Put original counter
                       value into GR7
LOOP LR   8,7        Set up copy in GR8 to
                       modify
     AR   8,6        Increment copy
     CS   7,8,CNTR   Update counter in
                       storage
     BC   4,LOOP     If original value had
                       changed, update new
                       value
```

The following shows two CPUs, A and B, executing this instruction sequence simultaneously: both CPUs attempt to add one to CNTR.

CPU A			CPU B		Comments
GR7	GR8	CNTR	GR7	GR8	
		16			
16	16				CPU A loads GR7 and GR8 from CNTR
			16	16	CPU B loads GR7 and GR8 from CNTR
				17	CPU B adds one to GR8
	17				CPU A adds one to GR8
		17			CPU A executes CS; successful match, store
			17		CPU B executes CS; no match, GR7 changed to CNTR value
				18	CPU B loads GR8 from GR7, adds one to GR8
		18			CPU B executes CS; successful match, store

Appendix B. Lists of Instructions

The following three figures list instructions arranged by name, mnemonic, and operation code. Some models may offer instructions that do not appear in the figures, such as those provided for emulation or as part of special or custom features.

The operation code 00 with a two-byte instruction format is allocated for use by the program when an indication of an invalid operation is required. It is improbable that this operation code will ever be assigned to an instruction implemented in the CPU.

Explanation of Symbols in "Characteristics" and "Op Code" Columns

A	Access exceptions.
A[1]	Access exceptions; not all access exceptions may occur; see instruction description for details.
B	PER branch event.
C	Condition code is set.
D	Data exception.
DF	Decimal-overflow exception.
DK	Decimal-divide exception.
DM	DIAGNOSE may generate various program exceptions and may change the condition code.
EO	Exponent-overflow exception.
EU	Exponent-underflow exception.
EX	Execute exception.
FK	Floating-point-divide exception.
IF	Fixed-point-overflow exception.
II	Interruptible instruction.
IK	Fixed-point-divide exception.
L	New condition code loaded.
LS	Significance exception.
MO	Monitor event.
P	Privileged-operation exception.
PS	Page-state exception.
PT	Page-transition exception.
R	PER general-register-alteration event.
RR	RR instruction format.
RS	RS instruction format.
RX	RX instruction format.
S	S instruction format.
SI	SI instruction format.
SO	Special-operation exception.
SP	Specification exception.
SS	SS instruction format.
ST	PER storage-alteration event.
$	Causes serialization
$[1]	Causes serialization when the M_1 and R_2 fields contain all ones and all zeros, respectively.
*	Bits 8-14 of the operation code are ignored.
≠	Bits 8-15 of the operation code are ignored.

Name	Mnemonic	Characteristics											Op Code	Page No.
ADD	AR	RR	C					IF			R		1A	7-7
ADD	A	RX	C		A			IF			R		5A	7-7
ADD DECIMAL	AP	SS	C		A		D	DF				ST	FA	8-4
ADD HALFWORD	AH	RX	C		A			IF			R		4A	7-7
ADD LOGICAL	ALR	RR	C								R		1E	7-7
ADD LOGICAL	AL	RX	C		A						R		5E	7-7
ADD NORMALIZED (extended)	AXR	RR	C			SP	EU	EO		LS			36	9-6
ADD NORMALIZED (long)	ADR	RR	C			SP	EU	EO		LS			2A	9-6
ADD NORMALIZED (long)	AD	RX	C		A	SP	EU	EO		LS			6A	9-6
ADD NORMALIZED (short)	AER	RR	C			SP	EU	EO		LS			3A	9-6
ADD NORMALIZED (short)	AE	RX	C		A	SP	EU	EO		LS			7A	9-6
ADD UNNORMALIZED (long)	AWR	RR	C			SP		EO		LS			2E	9-7
ADD UNNORMALIZED (long)	AW	RX	C		A	SP		EO		LS			6E	9-7
ADD UNNORMALIZED (short)	AUR	RR	C			SP		EO		LS			3E	9-7
ADD UNNORMALIZED (short)	AU	RX	C		A	SP		EO		LS			7E	9-7
AND	NR	RR	C								R		14	7-7
AND	N	RX	C		A						R		54	7-7
AND (character)	NC	SS	C		A							ST	D4	7-7
AND (immediate)	NI	SI	C		A							ST	94	7-7
BRANCH AND LINK	BALR	RR									B	R	05	7-8
BRANCH AND LINK	BAL	RX									B	R	45	7-8
BRANCH ON CONDITION	BCR	RR							$[1]		B		07	7-9
BRANCH ON CONDITION	BC	RX									B		47	7-9
BRANCH ON COUNT	BCTR	RR									B	R	06	7-9
BRANCH ON COUNT	BCT	RX									B	R	46	7-9
BRANCH ON INDEX HIGH	BXH	RS									B	R	86	7-10
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS									B	R	87	7-10
CLEAR I/O	CLRIO	S	C	P					$				9D01*	12-14
CLEAR PAGE	CLRP	S		P	A[1]					PS		ST	B215	10-3
COMPARE	CR	RR	C										19	7-11
COMPARE	C	RX	C		A								59	7-11
COMPARE (long)	CDR	RR	C			SP							29	9-8
COMPARE (long)	CD	RX	C		A	SP							69	9-8
COMPARE (short)	CER	RR	C			SP							39	9-8
COMPARE (short)	CE	RX	C		A	SP							79	9-8
COMPARE AND SWAP	CS	RS	C		A	SP			$		R	ST	BA	7-11
COMPARE DECIMAL	CP	SS	C		A		D						F9	8-5
COMPARE DOUBLE AND SWAP	CDS	RS	C		A	SP			$		R	ST	BB	7-11
COMPARE HALFWORD	CH	RX	C		A								49	7-13
COMPARE LOGICAL	CLR	RR	C										15	7-13
COMPARE LOGICAL	CL	RX	C		A								55	7-13
COMPARE LOGICAL (character)	CLC	SS	C		A								D5	7-13
COMPARE LOGICAL (immediate)	CLI	SI	C		A								95	7-13
COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS	C		A								BD	7-13
COMPARE LOGICAL LONG	CLCL	RR	C		A	SP				II	R		OF	7-14
CONNECT PAGE	CTP	RS	C	P	A[1]	SP				PT	R		B0	10-3
CONVERT TO BINARY	CVB	RX			A		D		IK		R		4F	7-15
CONVERT TO DECIMAL	CVD	RX			A							ST	4E	7-16
DECONFIGURE PAGE	DEP	S		P	A[1]	SP				PT			B21B	10-3
DIAGNOSE				P	DM								83	10-3
DISCONNECT PAGE	DCTP	S	C	P	A[1]	SP				PT			B21C	10-5
DIVIDE	DR	RR				SP			IK		R		1D	7-16
DIVIDE	D	RX			A	SP			IK		R		5D	7-16
DIVIDE (long)	DDR	RR				SP	EU	EO	FK				2D	9-8
DIVIDE (long)	DD	RX			A	SP	EU	EO	FK				6D	9-8
DIVIDE (short)	DER	RR				SP	EU	EO	FK				3D	9-8
DIVIDE (short)	DE	RX			A	SP	EU	EO	FK				7D	9-8
DIVIDE DECIMAL	DP	SS			A	SP	D		DK			ST	FD	8-5
EDIT	ED	SS	C		A		D					ST	DE	8-6
EDIT AND MARK	EDMK	SS	C		A		D				R	ST	DF	8-9
EXCLUSIVE OR	XR	RR	C								R		17	7-16
EXCLUSIVE OR	X	RX	C		A						R		57	7-16
EXCLUSIVE OR (character)	XC	SS	C		A							ST	D7	7-16
EXCLUSIVE OR (immediate)	XI	SI	C		A							ST	97	7-16

Instructions Arranged by Name (Part 1 of 3)

Name	Mne-monic	Characteristics						Op Code	Page No.
EXECUTE	EX	RX		A SP		EX		44	7-17
HALT DEVICE	HDV	S C	P			$		9E01*	12-16
HALT I/O	HIO	S C	P			$		9E00*	12-19
HALVE (long)	HDR	RR		SP	EU			24	9-9
HALVE (short)	HER	RR		SP	EU			34	9-9
INSERT CHARACTER	IC	RX		A			R	43	7-18
INSERT CHARACTERS UNDER MASK	ICM	RS C		A			R	BF	7-18
INSERT PAGE BITS	IPB	RS	P	A[1]			R	B4	10-5
INSERT PSW KEY	IPK	S	P				R	B20B	10-5
INSERT STORAGE KEY	ISK	RR	P	A[1] SP			R	09	10-5
LOAD	LR	RR					R	18	7-19
LOAD	L	RX		A			R	58	7-19
LOAD (long)	LDR	RR		SP				28	9-10
LOAD (long)	LD	RX		A SP				68	9-10
LOAD (short)	LER	RR		SP				38	9-10
LOAD (short)	LE	RX		A SP				78	9-10
LOAD ADDRESS	LA	RX					R	41	7-19
LOAD AND TEST	LTR	RR C					R	12	7-19
LOAD AND TEST (long)	LTDR	RR C		SP				22	9-10
LOAD AND TEST (short)	LTER	RR C		SP				32	9-10
LOAD COMPLEMENT	LCR	RR C			IF		R	13	7-19
LOAD COMPLEMENT (long)	LCDR	RR C		SP				23	9-10
LOAD COMPLEMENT (short)	LCER	RR C		SP				33	9-10
LOAD CONTROL	LCTL	RS	P	A SP				B7	10-6
LOAD FRAME INDEX	LFI	RS C	P				R	B8	10-6
LOAD HALFWORD	LH	RX		A			R	48	7-20
LOAD MULTIPLE	LM	RS		A			R	98	7-20
LOAD NEGATIVE	LNR	RR C					R	11	7-20
LOAD NEGATIVE (long)	LNDR	RR C		SP				21	9-11
LOAD NEGATIVE (short)	LNER	RR C		SP				31	9-11
LOAD POSITIVE	LPR	RR C			IF		R	10	7-20
LOAD POSITIVE (long)	LPDR	RR C		SP				20	9-11
LOAD POSITIVE (short)	LPER	RR C		SP				30	9-11
LOAD PSW	LPSW	S L	P	A SP		$		82	10-7
LOAD ROUNDED (extended to long)	LRDR	RR		SP	EO			25	9-11
LOAD ROUNDED (long to short)	LRER	RR		SP	EO			35	9-11
MAKE ADDRESSABLE	MAD	S C	P	A[1]		PT		B21D	10-7
MAKE UNADDRESSABLE	MUN	S C	P	A[1] SP		PT		B21E	10-7
MONITOR CALL	MC	SI		SP		MO		AF	7-21
MOVE (character)	MVC	SS		A			ST	D2	7-21
MOVE (immediate)	MVI	SI		A			ST	92	7-21
MOVE INVERSE	MVCIN	SS		A			ST	E8	7-22
MOVE LONG	MVCL	RR C		A SP		II	R ST	0E	7-22
MOVE NUMERICS	MVN	SS		A			ST	D1	7-24
MOVE WITH OFFSET	MVO	SS		A			ST	F1	7-25
MOVE ZONES	MVZ	SS		A			ST	D3	7-26
MULTIPLY	MR	RR		SP			R	1C	7-26
MULTIPLY	M	RX		A SP			R	5C	7-26
MULTIPLY (extended)	MXR	RR		SP	EU EO			26	9-12
MULTIPLY (long to extended)	MXDR	RR		SP	EU EO			27	9-12
MULTIPLY (long to extended)	MXD	RX		A SP	EU EO			67	9-12
MULTIPLY (long)	MDR	RR		SP	EU EO			2C	9-12
MULTIPLY (long)	MD	RX		A SP	EU EO			6C	9-12
MULTIPLY (short to long)	MER	RR		SP	EU EO			3C	9-12
MULTIPLY (short to long)	ME	RX		A SP	EU EO			7C	9-12
MULTIPLY DECIMAL	MP	SS		A SP	D		ST	FC	8-9
MULTIPLY HALFWORD	MH	RX		A			R	4C	7-26
OR	OR	RR C					R	16	7-27
OR	O	RX C		A			R	56	7-27
OR (character)	OC	SS C		A			ST	D6	7-27
OR (immediate)	OI	SI C		A			ST	96	7-27
PACK	PACK	SS		A			ST	F2	7-28
RESET REFERENCE BIT	RRB	S C	P	A[1]				B213	10-8
RETRIEVE STATUS AND PAGE	RSP	SS C	P	A			ST	D8	10-8
SET CLOCK	SCK	S C	P	A SP				B204	10-8

Instructions Arranged by Name (Part 2 of 3)

Name	Mnemonic	Characteristics				Op Code	Page No.
SET CLOCK COMPARATOR	SCKC	S	P A SP			B206	10-9
SET CPU TIMER	SPT	S	P A SP			B208	10-9
SET PAGE BITS	SPB	RS C	P A[1]			B5	10-9
SET PROGRAM MASK	SPM	RR L				04	7-28
SET PSW KEY FROM ADDRESS	SPKA	S	P			B20A	10-10
SET STORAGE KEY	SSK	RR	P A[1] SP			08	10-10
SET SYSTEM MASK	SSM	S	P A SP		SO	80	10-11
SHIFT AND ROUND DECIMAL	SRP	SS C	A	D DF	ST	F0	8-10
SHIFT LEFT DOUBLE	SLDA	RS C	SP	IF	R	8F	7-28
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS	SP		R	8D	7-29
SHIFT LEFT SINGLE	SLA	RS C		IF	R	8B	7-29
SHIFT LEFT SINGLE LOGICAL	SLL	RS			R	89	7-30
SHIFT RIGHT DOUBLE	SRDA	RS C	SP		R	8E	7-30
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS	SP		R	8C	7-30
SHIFT RIGHT SINGLE	SRA	RS C			R	8A	7-31
SHIFT RIGHT SINGLE LOGICAL	SRL	RS			R	88	7-31
START I/O	SIO	S C	P	$		9C00*	12-21
START I/O FAST RELEASE	SIOF	S C	P	$		9C01*	12-21
STORE	ST	RX	A		ST	50	7-31
STORE (long)	STD	RX	A SP		ST	60	9-13
STORE (short)	STE	RX	A SP		ST	70	9-13
STORE CAPACITY COUNTS	STCAP	S	P A		ST	B21F	10-11
STORE CHANNEL ID	STIDC	S C	P	$		B203	12-23
STORE CHARACTER	STC	RX	A		ST	42	7-32
STORE CHARACTERS UNDER MASK	STCM	RS	A		ST	BE	7-32
STORE CLOCK	STCK	S C	A	$	ST	B205	7-32
STORE CLOCK COMPARATOR	STCKC	S	P A SP		ST	B207	10-11
STORE CONTROL	STCTL	RS	P A SP		ST	B6	10-12
STORE CPU ID	STIDP	S	P A SP		ST	B202	10-12
STORE CPU TIMER	STPT	S	P A SP		ST	B209	10-13
STORE HALFWORD	STH	RX	A		ST	40	7-33
STORE MULTIPLE	STM	RS	A		ST	90	7-33
STORE THEN AND SYSTEM MASK	STNSM	SI	P A		ST	AC	10-13
STORE THEN OR SYSTEM MASK	STOSM	SI	P A SP		ST	AD	10-13
SUBTRACT	SR	RR C		IF	R	1B	7-33
SUBTRACT	S	RX C	A	IF	R	5B	7-33
SUBTRACT DECIMAL	SP	SS C	A	D DF	ST	FB	8-11
SUBTRACT HALFWORD	SH	RX C	A	IF	R	4B	7-34
SUBTRACT LOGICAL	SLR	RR C			R	1F	7-34
SUBTRACT LOGICAL	SL	RX C	A		R	5F	7-34
SUBTRACT NORMALIZED (extended)	SXR	RR C	SP	EU EO LS		37	9-14
SUBTRACT NORMALIZED (long)	SDR	RR C	SP	EU EO LS		2B	9-14
SUBTRACT NORMALIZED (long)	SD	RX C	A SP	EU EO LS		6B	9-14
SUBTRACT NORMALIZED (short)	SER	RR C	SP	EU EO LS		3B	9-14
SUBTRACT NORMALIZED (short)	SE	RX C	A SP	EU EO LS		7B	9-14
SUBTRACT UNNORMALIZED (long)	SWR	RR C	SP	EO LS		2F	9-14
SUBTRACT UNNORMALIZED (long)	SW	RX C	A SP	EO LS		6F	9-14
SUBTRACT UNNORMALIZED (short)	SUR	RR C	SP	EO LS		3F	9-14
SUBTRACT UNNORMALIZED (short)	SU	RX C	A SP	EO LS		7F	9-14
SUPERVISOR CALL	SVC	RR		$		0A	7-34
TEST AND SET	TS	S C	A	$	ST	93	7-35
TEST CHANNEL	TCH	S C	P	$		9F00≠	12-24
TEST I/O	TIO	S C	P	$		9D00*	12-25
TEST UNDER MASK	TM	SI C	A			91	7-35
TRANSLATE	TR	SS	A		ST	DC	7-36
TRANSLATE AND TEST	TRT	SS C	A		R	DD	7-36
UNPACK	UNPK	SS	A		ST	F3	7-37
ZERO AND ADD	ZAP	SS C	A	D DF	ST	F8	8-11

Instructions Arranged by Name (Part 3 of 3)

Mne-monic	Name	Characteristics										Op Code	Page No.
	DIAGNOSE			P	DM							83	10-3
A	ADD	RX	C		A			IF			R	5A	7-7
AD	ADD NORMALIZED (long)	RX	C		A	SP	EU	EO		LS		6A	9-6
ADR	ADD NORMALIZED (long)	RR	C			SP	EU	EO		LS		2A	9-6
AE	ADD NORMALIZED (short)	RX	C		A	SP	EU	EO		LS		7A	9-6
AER	ADD NORMALIZED (short)	RR	C			SP	EU	EO		LS		3A	9-6
AH	ADD HALFWORD	RX	C		A			IF			R	4A	7-7
AL	ADD LOGICAL	RX	C		A						R	5E	7-7
ALR	ADD LOGICAL	RR	C								R	1E	7-7
AP	ADD DECIMAL	SS	C		A		D	DF			ST	FA	8-4
AR	ADD	RR	C					IF			R	1A	7-7
AU	ADD UNNORMALIZED (short)	RX	C		A	SP		EO		LS		7E	9-7
AUR	ADD UNNORMALIZED (short)	RR	C			SP		EO		LS		3E	9-7
AW	ADD UNNORMALIZED (long)	RX	C		A	SP		EO		LS		6E	9-7
AWR	ADD UNNORMALIZED (long)	RR	C			SP		EO		LS		2E	9-7
AXR	ADD NORMALIZED (extended)	RR	C			SP	EU	EO		LS		36	9-6
BAL	BRANCH AND LINK	RX									B R	45	7-8
BALR	BRANCH AND LINK	RR									B R	05	7-8
BC	BRANCH ON CONDITION	RX									B	47	7-9
BCR	BRANCH ON CONDITION	RR							$[1]		B	07	7-9
BCT	BRANCH ON COUNT	RX									B R	46	7-9
BCTR	BRANCH ON COUNT	RR									B R	06	7-9
BXH	BRANCH ON INDEX HIGH	RS									B R	86	7-10
BXLE	BRANCH ON INDEX LOW OR EQUAL	RS									B R	87	7-10
C	COMPARE	RX	C		A							59	7-11
CD	COMPARE (long)	RX	C		A	SP						69	9-8
CDR	COMPARE (long)	RR	C			SP						29	9-8
CDS	COMPARE DOUBLE AND SWAP	RS	C		A	SP		$			R ST	BB	7-11
CE	COMPARE (short)	RX	C		A	SP						79	9-8
CER	COMPARE (short)	RR	C			SP						39	9-8
CH	COMPARE HALFWORD	RX	C		A							49	7-13
CL	COMPARE LOGICAL	RX	C		A							55	7-13
CLC	COMPARE LOGICAL (character)	SS	C		A							D5	7-13
CLCL	COMPARE LOGICAL LONG	RR	C		A	SP			II		R	0F	7-14
CLI	COMPARE LOGICAL (immediate)	SI	C		A							95	7-13
CLM	COMPARE LOGICAL CHARACTERS UNDER MASK	RS	C		A							BD	7-13
CLR	COMPARE LOGICAL	RR	C									15	7-13
CLRIO	CLEAR I/O	S	C	P				$				9D01*	12-14
CLRP	CLEAR PAGE	S		P	A[1]				PS		ST	B215	10-3
CP	COMPARE DECIMAL	SS	C		A		D					F9	8-5
CR	COMPARE	RR	C									19	7-11
CS	COMPARE AND SWAP	RS	C		A	SP		$			R ST	BA	7-11
CTP	CONNECT PAGE	RS	C	P	A[1]	SP			PT		R	B0	10-3
CVB	CONVERT TO BINARY	RX			A		D	IK			R	4F	7-15
CVD	CONVERT TO DECIMAL	RX			A						ST	4E	7-16
D	DIVIDE	RX			A	SP		IK			R	5D	7-16
DCTP	DISCONNECT PAGE	S	C	P	A[1]	SP			PT			B21C	10-5
DD	DIVIDE (long)	RX			A	SP	EU	EO	FK			6D	9-8
DDR	DIVIDE (long)	RR				SP	EU	EO	FK			2D	9-8
DE	DIVIDE (short)	RX			A	SP	EU	EO	FK			7D	9-8
DEP	DECONFIGURE PAGE	S		P	A[1]	SP			PT			B21B	10-3
DER	DIVIDE (short)	RR				SP	EU	EO	FK			3D	9-8
DP	DIVIDE DECIMAL	SS			A	SP	D		DK		ST	FD	8-5
DR	DIVIDE	RR				SP		IK			R	1D	7-16
ED	EDIT	SS	C		A		D				ST	DE	8-6
EDMP	EDIT AND MARK	SS	C		A		D				R ST	DF	8-9
EX	EXECUTE	RX			A	SP			EX			44	7-17
HDR	HALVE (long)	RR				SP	EU					24	9-9
HDV	HALT DEVICE	S	C	P				$				9E01*	12-16
HER	HALVE (short)	RR				SP	EU					34	9-9
HIO	HALT I/O	S	C	P				$				9E00*	12-19
IC	INSERT CHARACTER	RX			A						R	43	7-18
ICM	INSERT CHARACTERS UNDER MASK	RS	C		A						R	BF	7-18
IPB	INSERT PAGE BITS	RS		P	A[1]						R	B4	10-5

Instructions Arranged by Mnemonic (Part 1 of 3)

Mnemonic	Name	Characteristics						Op Code	Page No.
IPK	INSERT PSW KEY	S		P			R	B20B	10-5
ISK	INSERT STORAGE KEY	RR		P	A[1]	SP	R	09	10-5
L	LOAD	RX			A		R	58	7-19
LA	LOAD ADDRESS	RX					R	41	7-19
LCDR	LOAD COMPLEMENT (long)	RR	C			SP		23	9-10
LCER	LOAD COMPLEMENT (short)	RR	C			SP		33	9-10
LCR	LOAD COMPLEMENT	RR	C				IF R	13	7-19
LCTL	LOAD CONTROL	RS		P	A	SP		B7	10-6
LD	LOAD (long)	RX			A	SP		68	9-10
LDR	LOAD (long)	RR				SP		28	9-10
LE	LOAD (short)	RX			A	SP		78	9-10
LER	LOAD (short)	RR				SP		38	9-10
LFI	LOAD FRAME INDEX	RS	C	P			R	B8	10-6
LH	LOAD HALFWORD	RX			A		R	48	7-20
LM	LOAD MULTIPLE	RS			A		R	98	7-20
LNDR	LOAD NEGATIVE (long)	RR	C			SP		21	9-11
LNER	LOAD NEGATIVE (short)	RR	C			SP		31	9-11
LNR	LOAD NEGATIVE	RR	C				R	11	7-20
LPDR	LOAD POSITIVE (long)	RR	C			SP		20	9-11
LPER	LOAD POSITIVE (short)	RR	C			SP		30	9-11
LPR	LOAD POSITIVE	RR	C				IF R	10	7-20
LPSW	LOAD PSW	S	L	P	A	SP	$	82	10-7
LR	LOAD	RR					R	18	7-19
LRDR	LOAD ROUNDED (extended to long)	RR				SP	EO	25	9-11
LRER	LOAD ROUNDED (long to short)	RR				SP	EO	35	9-11
LTDR	LOAD AND TEST (long)	RR	C			SP		22	9-10
LTER	LOAD AND TEST (short)	RR	C			SP		32	9-10
LTR	LOAD AND TEST	RR	C				R	12	7-19
M	MULTIPLY	RX			A	SP	R	5C	7-26
MAD	MAKE ADDRESSABLE	S	C	P	A[1]		PT	B21D	10-7
MC	MONITOR CALL	SI				SP	MO	AF	7-21
MD	MULTIPLY (long)	RX			A	SP	EU EO	6C	9-12
MDR	MULTIPLY (long)	RR				SP	EU EO	2C	9-12
ME	MULTIPLY (short to long)	RX			A	SP	EU EO	7C	9-12
MER	MULTIPLY (short to long)	RR				SP	EU EO	3C	9-12
MH	MULTIPLY HALFWORD	RX			A		R	4C	7-26
MP	MULTIPLY DECIMAL	SS			A	SP	D ST	FC	8-9
MR	MULTIPLY	RR				SP	R	1C	7-26
MUN	MAKE UNADDRESSABLE	S	C	P	A[1]	SP	PT	B21E	10-7
MVC	MOVE (character)	SS			A		ST	D2	7-21
MVCIN	MOVE INVERSE	SS			A		ST	E8	7-22
MVCL	MOVE LONG	RR	C		A	SP	II R ST	OE	7-22
MVI	MOVE (immediate)	SI			A		ST	92	7-22
MVN	MOVE NUMERICS	SS			A		ST	D1	7-24
MVO	MOVE WITH OFFSET	SS			A		ST	F1	7-25
MVZ	MOVE ZONES	SS			A		ST	D3	7-26
MXD	MULTIPLY (long to extended)	RX			A	SP	EU EO	67	9-12
MXDR	MULTIPLY (long to extended)	RR				SP	EU EO	27	9-12
MXR	MULTIPLY (extended)	RR				SP	EU EO	26	9-12
N	AND	RX	C		A		R	54	7-7
NC	AND (character)	SS	C		A		ST	D4	7-7
NI	AND (immediate)	SI	C		A		ST	94	7-7
NR	AND	RR	C				R	14	7-7
O	OR	RX	C		A		R	56	7-27
OC	OR (character)	SS	C		A		ST	D6	7-27
OI	OR (immediate)	SI	C		A		ST	96	7-27
OR	OR	RR	C				R	16	7-27
PACK	PACK	SS			A		ST	F2	7-28
RRB	RESET REFERENCE BIT	S	C	P	A[1]			B213	10-8
RSP	RETRIEVE STATUS AND PAGE	SS	C	P	A		ST	D8	10-8
S	SUBTRACT	RX	C		A		IF R	5B	7-33
SCK	SET CLOCK	S	C	P	A	SP		B204	10-8
SCKC	SET CLOCK COMPARATOR	S		P	A	SP		B206	10-9
SD	SUBTRACT NORMALIZED (long)	RX	C		A	SP	EU EO LS	6B	9-14
SDR	SUBTRACT NORMALIZED (long)	RR	C			SP	EU EO LS	2B	9-14

Instructions Arranged by Mnemonic (Part 2 of 3)

Mnemonic	Name	Characteristics					Op Code	Page No.
SE	SUBTRACT NORMALIZED (short)	RX C	A SP	EU EO LS			7B	9-14
SER	SUBTRACT NORMALIZED (short)	RR C	SP	EU EO LS			3B	9-14
SH	SUBTRACT HALFWORD	RX C	A	IF		R	4B	7-34
SIO	START I/O	S C	P	$			9C00*	12-21
SIOF	START I/O FAST RELEASE	S C	P	$			9C01*	12-21
SL	SUBTRACT LOGICAL	RX C	A			R	5F	7-34
SLA	SHIFT LEFT SINGLE	RS C		IF		R	8B	7-29
SLDA	SHIFT LEFT DOUBLE	RS C	SP	IF		R	8F	7-28
SLDL	SHIFT LEFT DOUBLE LOGICAL	RS	SP			R	8D	7-29
SLL	SHIFT LEFT SINGLE LOGICAL	RS				R	89	7-30
SLR	SUBTRACT LOGICAL	RR C				R	1F	7-34
SP	SUBTRACT DECIMAL	SS C	A	D DF		ST	FB	8-11
SPB	SET PAGE BITS	RS C	P A[1]				B5	10-9
SPKA	SET PSW KEY FROM ADDRESS	S	P				B20A	10-10
SPM	SET PROGRAM MASK	RR L					04	7-28
SPT	SET CPU TIMER	S	P A SP				B208	10-9
SR	SUBTRACT	RR C		IF		R	1B	7-33
SRA	SHIFT RIGHT SINGLE	RS C				R	8A	7-31
SRDA	SHIFT RIGHT DOUBLE	RS C	SP			R	8E	7-30
SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS	SP			R	8C	7-30
SRL	SHIFT RIGHT SINGLE LOGICAL	RS				R	88	7-31
SRP	SHIFT AND ROUND DECIMAL	SS C	A	D DF		ST	F0	8-10
SSK	SET STORAGE KEY	RR	P A[1] SP				08	10-10
SSM	SET SYSTEM MASK	S	P A SP		SO		80	10-11
ST	STORE	RX	A			ST	50	7-31
STC	STORE CHARACTER	RX	A			ST	42	7-32
STCAP	STORE CAPACITY COUNTS	S	P A			ST	B21F	10-11
STCK	STORE CLOCK	S C	A	$		ST	B205	7-32
STCKC	STORE CLOCK COMPARATOR	S	P A SP			ST	B207	10-11
STCM	STORE CHARACTERS UNDER MASK	RS	A			ST	BE	7-32
STCTL	STORE CONTROL	RS	P A SP			ST	B6	10-12
STD	STORE (long)	RX	A SP			ST	60	9-13
STE	STORE (short)	RX	A SP			ST	70	9-13
STH	STORE HALFWORD	RX	A			ST	40	7-33
STIDC	STORE CHANNEL ID	S C	P	$			B203	12-23
STIDP	STORE CPU ID	S	P A SP			ST	B202	10-12
STM	STORE MULTIPLE	RS	A			ST	90	7-33
STNSM	STORE THEN AND SYSTEM MASK	SI	P A			ST	AC	10-13
STOSM	STORE THEN OR SYSTEM MASK	SI	P A SP			ST	AD	10-13
STPT	STORE CPU TIMER	S	P A SP			ST	B209	10-13
SU	SUBTRACT UNNORMALIZED (short)	RX C	A SP	EO LS			7F	9-14
SUR	SUBTRACT UNNORMALIZED (short)	RR C	SP	EO LS			3F	9-14
SVC	SUPERVISOR CALL	RR		$			0A	7-34
SW	SUBTRACT UNNORMALIZED (long)	RX C	A SP	EO LS			6F	9-14
SWR	SUBTRACT UNNORMALIZED (long)	RR C	SP	EO LS			2F	9-14
SXR	SUBTRACT NORMALIZED (extended)	RR C	SP	EU EO LS			37	9-14
TCH	TEST CHANNEL	S C	P	$			9F00≠	12-24
TIO	TEST I/O	S C	P	$			9D00*	12-25
TM	TEST UNDER MASK	SI C	A				91	7-35
TR	TRANSLATE	SS	A			ST	DC	7-36
TRT	TRANSLATE AND TEST	SS C	A			R	DD	7-36
TS	TEST AND SET	S C	A	$		ST	93	7-35
UNPK	UNPACK	SS	A			ST	F3	7-37
X	EXCLUSIVE OR	RX C	A			R	57	7-16
XC	EXCLUSIVE OR (character)	SS C	A			ST	D7	7-16
XI	EXCLUSIVE OR (immediate)	SI C	A			ST	97	7-16
XR	EXCLUSIVE OR	RR C				R	17	7-16
ZAP	ZERO AND ADD	SS C	A	D DF		ST	F8	8-11

Instructions Arranged by Mnemonic (Part 3 of 3)

Op Code	Name	Mne-monic	Characteristics				Page No.
04	SET PROGRAM MASK	SPM	RR L				7-28
05	BRANCH AND LINK	BALR	RR			B R	7-8
06	BRANCH ON COUNT	BCTR	RR			B R	7-9
07	BRANCH ON CONDITION	BCR	RR		^{1}	B	7-9
08	SET STORAGE KEY	SSK	RR	P A^1 SP			10-10
09	INSERT STORAGE KEY	ISK	RR	P A^1 SP		R	10-5
0A	SUPERVISOR CALL	SVC	RR		$		7-34
0E	MOVE LONG	MVCL	RR C	A SP	II	R ST	7-22
0F	COMPARE LOGICAL LONG	CLCL	RR C	A SP	II	R	7-14
10	LOAD POSITIVE	LPR	RR C		IF	R	7-20
11	LOAD NEGATIVE	LNR	RR C			R	7-20
12	LOAD AND TEST	LTR	RR C			R	7-19
13	LOAD COMPLEMENT	LCR	RR C		IF	R	7-19
14	AND	NR	RR C			R	7-7
15	COMPARE LOGICAL	CLR	RR C				7-13
16	OR	OR	RR C			R	7-27
17	EXCLUSIVE OR	XR	RR C			R	7-16
18	LOAD	LR	RR			R	7-19
19	COMPARE	CR	RR C				7-11
1A	ADD	AR	RR C		IF	R	7-7
1B	SUBTRACT	SR	RR C		IF	R	7-33
1C	MULTIPLY	MR	RR	SP		R	7-26
1D	DIVIDE	DR	RR	SP	IK	R	7-16
1E	ADD LOGICAL	ALR	RR C			R	7-7
1F	SUBTRACT LOGICAL	SLR	RR C			R	7-34
20	LOAD POSITIVE (long)	LPDR	RR C	SP			9-11
21	LOAD NEGATIVE (long)	LNDR	RR C	SP			9-11
22	LOAD AND TEST (long)	LTDR	RR C	SP			9-10
23	LOAD COMPLEMENT (long)	LCDR	RR C	SP			9-10
24	HALVE (long)	HDR	RR	SP	EU		9-9
25	LOAD ROUNDED (extended to long)	LRDR	RR	SP	EO		9-11
26	MULTIPLY (extended)	MXR	RR	SP	EU EO		9-12
27	MULTIPLY (long to extended)	MXDR	RR	SP	EU EO		9-12
28	LOAD (long)	LDR	RR	SP			9-10
29	COMPARE (long)	CDR	RR C	SP			9-8
2A	ADD NORMALIZED (long)	ADR	RR C	SP	EU EO LS		9-6
2B	SUBTRACT NORMALIZED (long)	SDR	RR C	SP	EU EO LS		9-14
2C	MULTIPLY (long)	MDR	RR	SP	EU EO		9-12
2D	DIVIDE (long)	DDR	RR	SP	EU EO FK		9-8
2E	ADD UNNORMALIZED (long)	AWR	RR C	SP	EO LS		9-7
2F	SUBTRACT UNNORMALIZED (long)	SWR	RR C	SP	EO LS		9-14
30	LOAD POSITIVE (short)	LPER	RR C	SP			9-11
31	LOAD NEGATIVE (short)	LNER	RR C	SP			9-11
32	LOAD AND TEST (short)	LTER	RR C	SP			9-10
33	LOAD COMPLEMENT (short)	LCER	RR C	SP			9-10
34	HALVE (short)	HER	RR	SP	EU		9-9
35	LOAD ROUNDED (long to short)	LRER	RR	SP	EO		9-11
36	ADD NORMALIZED (extended)	AXR	RR C	SP	EU EO LS		9-6
37	SUBTRACT NORMALIZED (extended)	SXR	RR C	SP	EU EO LS		9-14
38	LOAD (short)	LER	RR	SP			9-10
39	COMPARE (short)	CER	RR C	SP			9-8
3A	ADD NORMALIZED (short)	AER	RR C	SP	EU EO LS		9-6
3B	SUBTRACT NORMALIZED (short)	SER	RR C	SP	EU EO LS		9-14
3C	MULTIPLY (short to long)	MER	RR	SP	EU EO		9-12
3D	DIVIDE (short)	DER	RR	SP	EU EO FK		9-8
3E	ADD UNNORMALIZED (short)	AUR	RR C	SP	EO LS		9-7
3F	SUBTRACT UNNORMALIZED (short)	SUR	RR C	SP	EO LS		9-14
40	STORE HALFWORD	STH	RX	A		ST	7-33
41	LOAD ADDRESS	LA	RX			R	7-19
42	STORE CHARACTER	STC	RX	A		ST	7-32
43	INSERT CHARACTER	IC	RX	A		R	7-18
44	EXECUTE	EX	RX	A SP	EX		7-17
45	BRANCH AND LINK	BAL	RX			B R	7-8
46	BRANCH ON COUNT	BCT	RX			B R	7-9
47	BRANCH ON CONDITION	BC	RX			B	7-9

Instructions Arranged by Operation Code (Part 1 of 3)

Op Code	Name	Mne-monic	Characteristics				Page No.
48	LOAD HALFWORD	LH	RX	A		R	7-20
49	COMPARE HALFWORD	CH	RX C	A			7-13
4A	ADD HALFWORD	AH	RX C	A	IF	R	7-7
4B	SUBTRACT HALFWORD	SH	RX C	A	IF	R	7-34
4C	MULTIPLY HALFWORD	MH	RX	A		R	7-26
4E	CONVERT TO DECIMAL	CVD	RX	A		ST	7-16
4F	CONVERT TO BINARY	CVB	RX	A	D IK	R	7-15
50	STORE	ST	RX	A		ST	7-31
54	AND	N	RX C	A		R	7-7
55	COMPARE LOGICAL	CL	RX C	A			7-13
56	OR	O	RX C	A		R	7-27
57	EXCLUSIVE OR	X	RX C	A		R	7-16
58	LOAD	L	RX	A		R	7-19
59	COMPARE	C	RX C	A			7-11
5A	ADD	A	RX C	A	IF	R	7-7
5B	SUBTRACT	S	RX C	A	IF	R	7-33
5C	MULTIPLY	M	RX	A SP		R	7-26
5D	DIVIDE	D	RX	A SP	IK	R	7-16
5E	ADD LOGICAL	AL	RX C	A		R	7-7
5F	SUBTRACT LOGICAL	SL	RX C	A		R	7-34
60	STORE (long)	STD	RX	A SP		ST	9-13
67	MULTIPLY (long to extended)	MXD	RX	A SP	EU EO		9-12
68	LOAD (long)	LD	RX	A SP			9-10
69	COMPARE (long)	CD	RX C	A SP			9-8
6A	ADD NORMALIZED (long)	AD	RX C	A SP	EU EO LS		9-6
6B	SUBTRACT NORMALIZED (long)	SD	RX C	A SP	EU EO LS		9-14
6C	MULTIPLY (long)	MD	RX	A SP	EU EO		9-12
6D	DIVIDE (long)	DD	RX	A SP	EU EO FK		9-8
6E	ADD UNNORMALIZED (long)	AW	RX C	A SP	EO LS		9-7
6F	SUBTRACT UNNORMALIZED (long)	SW	RX C	A SP	EO LS		9-14
70	STORE (short)	STE	RX	A SP		ST	9-13
78	LOAD (short)	LE	RX	A SP			9-10
79	COMPARE (short)	CE	RX C	A SP			9-8
7A	ADD NORMALIZED (short)	AE	RX C	A SP	EU EO LS		9-6
7B	SUBTRACT NORMALIZED (short)	SE	RX C	A SP	EU EO LS		9-14
7C	MULTIPLY (short to long)	ME	RX	A SP	EU EO		9-12
7D	DIVIDE (short)	DE	RX	A SP	EU EO FK		9-8
7E	ADD UNNORMALIZED (short)	AU	RX C	A SP	EO LS		9-7
7F	SUBTRACT UNNORMALIZED (short)	SU	RX C	A SP	EO LS		9-14
80	SET SYSTEM MASK	SSM	S	P A SP	SO		10-11
82	LOAD PSW	LPSW	S L	P A SP	$		10-7
83	DIAGNOSE			P DM			10-3
86	BRANCH ON INDEX HIGH	BXH	RS			B R	7-10
87	BRANCH ON INDEX LOW OR EQUAL	BXLE	RS			B R	7-10
88	SHIFT RIGHT SINGLE LOGICAL	SRL	RS			R	7-31
89	SHIFT LEFT SINGLE LOGICAL	SLL	RS			R	7-30
8A	SHIFT RIGHT SINGLE	SRA	RS C			R	7-31
8B	SHIFT LEFT SINGLE	SLA	RS C		IF	R	7-29
8C	SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS	SP		R	7-30
8D	SHIFT LEFT DOUBLE LOGICAL	SLDL	RS	SP		R	7-29
8E	SHIFT RIGHT DOUBLE	SRDA	RS C	SP		R	7-30
8F	SHIFT LEFT DOUBLE	SLDA	RS C	SP	IF	R	7-28
90	STORE MULTIPLE	STM	RS	A		ST	7-33
91	TEST UNDER MASK	TM	SI C	A			7-35
92	MOVE (immediate)	MVI	SI	A		ST	7-21
93	TEST AND SET	TS	S C	A	$	ST	7-35
94	AND (immediate)	NI	SI C	A		ST	7-7
95	COMPARE LOGICAL (immediate)	CLI	SI C	A			7-13
96	OR (immediate)	OI	SI C	A		ST	7-27
97	EXCLUSIVE OR (immediate)	XI	SI C	A		ST	7-16
98	LOAD MULTIPLE	LM	RS	A		R	7-20
9C00*	START I/O	SIO	S C	P	$		12-21
9C01*	START I/O FAST RELEASE	SIOF	S C	P	$		12-21
9D00*	TEST I/O	TIO	S C	P	$		12-25
9D01*	CLEAR I/O	CLRIO	S C	P	$		12-14

Instructions Arranged by Operation Code (Part 2 of 3)

Op Code	Name	Mne-monic	Characteristics								Page No.
9E00*	HALT I/O	HIO	S	C	P			$			12-19
9E01*	HALT DEVICE	HDV	S	C	P			$			12-16
9F00≠	TEST CHANNEL	TCH	S	C	P			$			12-24
AC	STORE THEN AND SYSTEM MASK	STNSM	SI		P	A				ST	10-13
AD	STORE THEN OR SYSTEM MASK	STOSM	SI		P	A	SP			ST	10-13
AF	MONITOR CALL	MC	SI				SP		MO		7-21
B0	CONNECT PAGE	CTP	RS	C	P	A[1]	SP		PT	R	10-3
B202	STORE CPU ID	STIDP	S		P	A	SP			ST	10-12
B203	STORE CHANNEL ID	STIDC	S	C	P			$			12-23
B204	SET CLOCK	SCK	S	C	P	A	SP				10-8
B205	STORE CLOCK	STCK	S	C		A		$		ST	7-32
B206	SET CLOCK COMPARATOR	SCKC	S		P	A	SP				10-9
B207	STORE CLOCK COMPARATOR	STCKC	S		P	A	SP			ST	10-11
B208	SET CPU TIMER	SPT	S		P	A	SP				10-9
B209	STORE CPU TIMER	STPT	S		P	A	SP			ST	10-13
B20A	SET PSW KEY FROM ADDRESS	SPKA	S		P						10-10
B20B	INSERT PSW KEY	IPK	S		P					R	10-5
B213	RESET REFERENCE BIT	RRB	S	C	P	A[1]					10-8
B215	CLEAR PAGE	CLRP	S		P	A[1]			PS	ST	10-3
B21B	DECONFIGURE PAGE	DEP	S		P	A[1]	SP		PT		10-3
B21C	DISCONNECT PAGE	DCTP	S	C	P	A[1]	SP		PT		10-5
B21D	MAKE ADDRESSABLE	MAD	S	C	P	A[1]			PT		10-7
B21E	MAKE UNADDRESSABLE	MUN	S	C	P	A[1]	SP		PT		10-7
B21F	STORE CAPACITY COUNTS	STCAP	S		P	A				ST	10-11
B4	INSERT PAGE BITS	IPB	RS		P	A[1]				R	10-5
B5	SET PAGE BITS	SPB	RS	C	P	A[1]					10-9
B6	STORE CONTROL	STCTL	RS		P	A	SP			ST	10-12
B7	LOAD CONTROL	LCTL	RS		P	A	SP				10-6
B8	LOAD FRAME INDEX	LFI	RS	C	P					R	10-6
BA	COMPARE AND SWAP	CS	RS	C		A	SP	$		R ST	7-11
BB	COMPARE DOUBLE AND SWAP	CDS	RS	C		A	SP	$		R ST	7-11
BD	COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS	C		A					7-13
BE	STORE CHARACTERS UNDER MASK	STCM	RS			A				ST	7-32
BF	INSERT CHARACTERS UNDER MASK	ICM	RS	C		A				R	7-18
D1	MOVE NUMERICS	MVN	SS			A				ST	7-24
D2	MOVE (character)	MVC	SS			A				ST	7-21
D3	MOVE ZONES	MVZ	SS			A				ST	7-26
D4	AND (character)	NC	SS	C		A				ST	7-7
D5	COMPARE LOGICAL (character)	CLC	SS	C		A					7-13
D6	OR (character)	OC	SS	C		A				ST	7-27
D7	EXCLUSIVE OR (character)	XC	SS	C		A				ST	7-16
D8	RETRIEVE STATUS AND PAGE	RSP	SS	C	P	A				ST	10-8
DC	TRANSLATE	TR	SS			A				ST	7-36
DD	TRANSLATE AND TEST	TRT	SS	C		A				R	7-36
DE	EDIT	ED	SS	C		A		D		ST	8-6
DF	EDIT AND MARK	EDMK	SS	C		A		D		R ST	8-9
E8	MOVE INVERSE	MVCIN	SS			A				ST	7-22
F0	SHIFT AND ROUND DECIMAL	SRP	SS	C		A		D	DF	ST	8-10
F1	MOVE WITH OFFSET	MVO	SS			A				ST	7-25
F2	PACK	PACK	SS			A				ST	7-28
F3	UNPACK	UNPK	SS			A				ST	7-37
F8	ZERO AND ADD	ZAP	SS	C		A		D	DF	ST	8-11
F9	COMPARE DECIMAL	CP	SS	C		A		D			8-5
FA	ADD DECIMAL	AP	SS	C		A		D	DF	ST	8-4
FB	SUBTRACT DECIMAL	SP	SS	C		A		D	DF	ST	8-11
FC	MULTIPLY DECIMAL	MP	SS			A	SP	D		ST	8-9
FD	DIVIDE DECIMAL	DP	SS			A	SP	D	DK	ST	8-5

Instructions Arranged by Operation Code (Part 3 of 3)

Appendix C. Condition-Code Settings

Instruction	Condition Code			
	0	1	2	3
ADD, ADD HALFWORD	zero	< zero	> zero	overflow
ADD DECIMAL	zero	< zero	> zero	overflow
ADD LOGICAL	zero, no carry	not zero, no carry	zero, carry	not zero, carry
ADD NORMALIZED	zero	< zero	> zero	–
ADD UNNORMALIZED	zero	< zero	> zero	–
AND	zero	not zero	–	–
CLEAR I/O	no operation in progress	CSW stored	channel busy	not operational
COMPARE, COMPARE HALFWORD	equal	low	high	–
COMPARE AND SWAP	equal	not equal	–	–
COMPARE DECIMAL	equal	low	high	–
COMPARE DOUBLE AND SWAP	equal	not equal	–	–
COMPARE LOGICAL	equal	low	high	–
COMPARE LOGICAL CHARACTERS UNDER MASK	equal	low	high	–
COMPARE LOGICAL LONG	equal	low	high	–
CONNECT PAGE	successful	already connected	unsuccessful	–
DISCONNECT PAGE	successful	already disconnected	–	–
EDIT, EDIT AND MARK	zero	< zero	> zero	–
EXCLUSIVE OR	zero	not zero	–	–
HALT DEVICE	interruption pending/busy	CSW stored	channel working	not operational
HALT I/O	interruption pending	CSW stored	burst op. stopped	not operational
INSERT CHARACTERS UNDER MASK	all zeros	1st bit one	1st bit zero	–
LOAD AND TEST	zero	< zero	> zero	–
LOAD COMPLEMENT (fixed point)	zero	< zero	> zero	overflow
LOAD COMPLEMENT (floating point)	zero	< zero	> zero	–
LOAD FRAME INDEX	addressable	connected	disconnected	address invalid
LOAD NEGATIVE	zero	< zero	–	–
LOAD POSITIVE (fixed point)	zero	–	> zero	overflow
LOAD POSITIVE (floating point)	zero	–	> zero	–
MAKE ADDRESSABLE	successful	already addressable	–	–
MAKE UNADDRESSABLE	successful	already connected	–	–
MOVE LONG	length equal	length low	length high	destr. overlap
OR	zero	not zero	–	–
RESET REFERENCE BIT	R bit zero, C bit zero	R bit zero, C bit one	R bit one, C bit zero	R bit one, C bit one
RETRIEVE STATUS AND PAGE	valid	–	–	invalid
SET PAGE BITS	R bit zero, C bit zero	R bit zero, C bit one	R bit one, C bit zero	R bit one, C bit one

Summary of Condition-Code Settings (Part 1 of 2)

Instruction	Condition Code			
	0	1	2	3
SET CLOCK	set	secure	–	not operational
SHIFT AND ROUND DECIMAL	zero	< zero	> zero	overflow
SHIFT LEFT (DOUBLE or SINGLE)	zero	< zero	> zero	overflow
SHIFT RIGHT (DOUBLE or SINGLE)	zero	< zero	> zero	–
START I/O, START I/O FAST RELEASE	successful	CSW stored	busy	not operational
STORE CHANNEL ID	ID stored	CSW stored	busy	not operational
STORE CLOCK	set	not set	error	not operational
SUBTRACT, SUBTRACT HALFWORD	zero	< zero	> zero	overflow
SUBTRACT DECIMAL	zero	< zero	> zero	overflow
SUBTRACT LOGICAL	–	not zero, no carry	zero, carry	not zero, carry
SUBTRACT NORMALIZED	zero	< zero	> zero	–
SUBTRACT UNNORMALIZED	zero	< zero	> zero	–
TEST AND SET	left zero	left one	–	–
TEST CHANNEL	available	interruption pending	burst mode	not operational
TEST I/O	available	CSW stored	busy	not operational
TEST UNDER MASK	all zeros	mixed	–	all ones
TRANSLATE AND TEST	zero	incomplete	complete	–
ZERO AND ADD	zero	< zero	> zero	overflow

Explanation:

> zero	Result is greater than zero
< zero	Result is less than zero
high	First operand compares high
low	First operand compares low
length	Length of first operand

Note: The condition code may also be changed by DIAGNOSE, EXECUTE, LOAD PSW, SET PROGRAM MASK, and SUPERVISOR CALL, and by an interruption.

Summary of Condition-Code Settings (Part 2 of 2)

Index

d

e

f

k

l

m

n

o

p

r

t

IBM 4300 Processors
Principles of Operation
for ECPS:VSE Mode
Order No. GA22-7070-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

_____

_____

_____

Please use pressure-sensitive or other gummed tape to seal this form.

What is your occupation? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the front cover or title page.)

Reader's Comment Form

Fold and tape — Please Do Not Staple — Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department B98
P.O. Box 390
Poughkeepsie, New York 12602

Fold and tape — Please Do Not Staple — Fold and tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Cut or Fold Along Line

IBM 4300 Processors Principles of Operation for ECPS: VSE Mode (File No. 4300-01) Printed in U.S.A. GA22-7070-1

Cut or Fold Along Line

GA22-7070-1

IBM 4300 Processors Principles of Operation for ECPS:VSE Mode (File No. 4300-01) Printed in U.S.A. GA22-7070-1

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601