Xerox 560 Computer

Reference Manual

90 30 76A

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

Xerox 560 Computer

Reference Manual

FIRST EDITION

90 30 76A

January 1974

Price: $7.25

Printed in U.S.A.

RELATED PUBLICATIONS

Title	Publication No.
Xerox Symbol/LN, OPS Reference Manual	90 17 90
Xerox Meta-Symbol/LN, OPS Reference Manual	90 09 52
Xerox Macro-Symbol/LN, OPS Reference Manual	90 15 78

Manual Content Codes: BP – batch processing, LN – language, OPS – operations, RP – remote processing, RT – real-time, SM – system management, TS – time-sharing, UT – utilities.

CONTENTS

APPENDIXES

FIGURES

TABLES

1. XEROX 560 COMPUTER SYSTEM

INTRODUCTION

The Xerox 560 general-purpose, digital, computer system accommodates a variety of scientific, business, real-time, and time-sharing applications. A system includes system control, basic processor, I/O processor, and main memory (up to 256K words) with two ports. Each major system element performs asynchronously with respect to other elements.

The basic system can be readily expanded. Memory access paths can be increased from the basic two ports to a maximum of six ports. Input/output capability can be increased by adding more input/output processors (IOPs), device controllers, and peripheral devices.

The basic processor (BP) has an extensive instruction set that includes floating-point, byte-string, and decimal instructions.

The multiaccess memory units, with interleaving, afford a high level of system performance. Main memory can be expanded in 16K word increments to a maximum of 256K words. Address interleaving may be performed between memory units of like size. The number of ports to each memory unit can be expanded to allow independent access to memory by up to six "processor clusters" (i.e., functional groups).

Processor clusters are the grouping of two or more functions (such as a basic processor, an I/O processor, and interfaces) on a common bus. Clustering permits processors to share common facilities, e.g., buses and memory interfaces. Therefore, the hardware is less redundant, hence less complex, resulting in more reliability at a lower cost. There are multiple combinations of functional groups from which to select.

Existing Sigma 5-9 programs may be run on the system. The upward compatibility of the comprehensive, modular software (assemblers, compilers, mathematical and utility routines, and application packages) eliminates reprogramming.

Features have been incorporated in this design to enhance overall system reliability, maintainability, and availability. Centralized switches for system repartitioning may permit faulty units, or an entire subsystem, to be isolated for diagnosis or repair while the primary system continues operation. Parity checking is performed on each byte of information for most system interfaces and internal control signals. Most failed instructions are automatically retried, and uninterrupted processing continues. The only apparent effect may be an entry in the error log. In the event an error is irrecoverable, there are error storage registers that return complete data on the fault and the status of the system at that point.

GENERAL CHARACTERISTICS

The following system features and characteristics permit efficient operation in general-purpose, multiprocessor, time-sharing, real-time, and multiuse environments:

- Word-oriented memory (32-bit word plus parity bit per byte) that can be addressed and altered as byte (8-bit), halfword (2-byte), word (4-byte), and doubleword (8-byte) quantities.
- Memory expandable to 256K words (K = 1024) in modular units of 16K words each.
- Indirect addressing with or without postindexing.
- Displacement index registers, automatically self-adjusting for all data sizes.
- Immediate operand instructions for greater storage efficiency and increased speed.
- Four blocks of 16 general-purpose registers for addressing, indexing, and accumulating. Multiple registers permit rapid context switching.
- Hardware memory mapping, which virtually eliminates memory fragmentation and provides dynamic program relocation.
- Memory access protection for system and information security and protection.
- Memory write protection within memory units to prevent inadvertent destruction of critical areas of memory from any processor cluster.
- Watchdog timer to assure nonstop operation.
- Real-time priority interrupt system with automatic identification and priority assignment, fast response time, and 14 internal and up to 48 external levels that can be individually armed, enabled, and triggered by program control.
- Instructions with long execution times can be interrupted.
- Automatic traps for error or fault conditions, with masking capability and maximum recoverability, under program control.
- Power fail-safe for automatic shutdown and resumption of processing in event of power failure.
- Multiple interval timers with a choice of resolutions for independent time bases.
- Privileged instruction logic for program integrity in multiuse environments.

- Extensive instruction set that includes:
  - Byte, halfword, word, and doubleword operations.
  - Use of all memory-referencing instructions for register-to-register operations, with or without indirect addressing and postindexing, and within normal instruction format.
  - Multiple register operations.
  - Fixed-point integer arithmetic operations in halfword, word, and doubleword modes.
  - Immediate operand instructions.
  - Floating-point hardware operations in short and long formats with significance, zero, and normalization control and checking, all under full program control.
  - Full complement of logical operations (AND, OR, exclusive OR).
  - Comparison operations, including compare between limits (with limits in memory or in registers).
  - Call instructions that permit up to 64 dynamically variable, user-defined instructions, and allow a program access to operating system functions without operating system intervention.
  - Decimal hardware operations, including arithmetic, edit, and pack/unpack.
  - Byte-string instructions.
  - Push-down stack operations (hardware implemented) of single or multiple words, with automatic limit checking, for dynamic space allocation, subroutine communication, and recursive routine capability.
  - Automatic conversion operations, including binary/BCD and any other weighted-number systems.
  - Analyze instruction that facilitates effective address computation.
  - Interpret instruction that increases speed of interpretive programs.
  - Shift operations (left and right) of word or doubleword, including logical, circular, arithmetic, searching shift, and floating-point modes.
- Built-in reliability and maintainability features that include:
  - Extensive error logging. When a fault is detected, system status and fault information are available for program retrieval and logging for subsequent analysis.
  - Full parity checking on all data and addresses communicated in either direction on buses between memory units and processors, providing fault detection and location capability to permit the operating system or diagnostic program to quickly determine a faulty unit.
  - Address stop feature that permits operator or maintenance personnel to:

    Stop on any instruction address.

    Stop on any memory reference address.

    Stop when any word in a selected page of memory is referenced.
  - Traps that provide for detection of a variety of fault conditions, designed to enable a high degree of system recoverability.
  - Partitioning features that enable system reconfiguration via a centralized Configuration Control Panel. Units may be partitioned from the system by selectively disabling them from buses (assuming other system facilities can handle the additional load). Thus, faulty units, processors, devices, or an alternate system can be isolated from the operational system to enable diagnosis or repair while the primary system continues operation.
- Independently operating I/O system with the following features:
  - Direct input/output (READ DIRECT, WRITE DIRECT instructions) for transfer of 32-bit words between the specified general register and an external device; a 16-bit address is transferred for selection and control purposes; and each transfer is under direct program control.
  - Up to five independent I/O processor clusters (restricted only by the maximum number of 6 ports).
  - Multiplexor I/O processors (MIOPs) (up to 3 per I/O cluster), each providing for simultaneous operation of up to 16 devices per processor.
  - Data chaining for gather-read and scatter-write operations.
  - Command chaining for multiple record operations.
  - Write lock protect feature within memory unit for positive protection from all processors storing into memory.
- Comprehensive modular software that is program compatible with Sigma 5-9 computers:
  - Expands in capability and speed as system grows.
  - Operating system: Control Program-Five (CP-V).

- Language processors and utilities and applications software for both commercial and scientific users.

- Peripheral equipment includes:
  - Card equipment: Reading speeds up to 1500 cards per minute; punching speed of 100 cards per minute; intermixed binary and EBCDIC card codes.
  - Line printers: Fully buffered with speeds up to 1250 lines per minute; 132 print positions with character sets containing 64 or 95 characters.
  - Magnetic tape units: 9-track systems, single or dual density (1600 or 800/1600 BPI), industry-compatible; high-speed, automatic loading units operating at 125 inches per second with transfer rates up to 200,000 bytes per second; and at 75 inches per second with transfer rates up to 120,000 bytes per second.
  - Rapid Access Data (RAD) and disk files: RAD capacity of 2.9 million bytes, with a transfer rate of 750,000 bytes per second; disk capacities in increments of 86 million bytes (formatted) per unit with a transfer rate of 806,000 bytes per second, and in increments of 49 million bytes per unit with a transfer rate of 312,500 bytes per second.
  - Keyboard printers: 10 characters per second.
  - Data communications equipment: Complete line of character-oriented, message-oriented, and procedure-oriented equipment to connect remote user terminals (including remote batch) to the computer center via common carrier lines and local terminals directly.

STANDARD AND OPTIONAL FEATURES

A basic system has the following standard features:

- A basic processor (BP) that includes:
  - Full instruction set
  - Memory map with access protection
  - Register blocks (4)
  - Multiplexor Input/Output Processor (MIOP) with:
    - 16 subchannels
    - 1- or 4-byte interface
    - Input/Output Adapter
- Two memory units that include:
  - Dual port access
  - Memory write lock protection
- A system control processor that includes:
  - Real-time clocks (4)
  - Internal interrupts (14)
  - Power fail-safe detection
  - External Direct Input/Output Interface (DIO)
  - External Control Subsystem (ECS)
  - System Control Panel (SCP)
  - Configuration Control Panel (CCP)
  - Local and remote assist facility
- Error detection facilities
- Diagnostics

A system may have the following optional features:

- BP options:
  - Up to 48 external priority interrupts (in groups of 12)
- Memory options:
  - Memory expansion up to 256K words
  - Up to 4 additional access ports (in sets of 2).
- Input/Output options:
  - Multiple I/O clusters[†].
    - Up to 3 additional MIOPs, each with 16 subchannels, per cluster.
    - One Input/Output Adapter (for one MIOP) per cluster.
    - One Rotating Memory Processor (RMP) per cluster.

GENERAL-PURPOSE FEATURES

General-purpose computing applications are characterized by emphasis on computation and internal data handling.

[†]The aggregate of processor clusters is restricted by the maximum memory port limitation of 6.

Many operations are performed in floating-point format and on strings of characters. Other typical characteristics include decimal arithmetic operations, binary to decimal number conversion (for printing or display), and high system input/output transfer rates.

General-purpose features are described in the following paragraphs.

Floating-Point Hardware. Both short (32-bit) and long (64-bit) formats are available in the floating-point instructions. Under program control, the user may select optional zero checking, normalization, floating-point rounding and significance checking. Significance checking permits use of short floating-point format for high processing speed and storage economy and of long floating-point format when loss of significance is detected.

Decimal Arithmetic Hardware. Decimal arithmetic instructions operate on up to 31 digits plus sign. This instruction set includes pack/unpack instructions for converting to/from the packed format of two digits per byte, and a generalized edit instruction for zero suppression, check protection, and formatting, with punctuation to display or print it.

Indirect Addressing. Indirect addressing facilitates table linkages and permits keeping data sections of a program separate from procedure sections for ease of maintenance.

Displacement Indexing. Indexing by means of a "floating" displacement permits accessing a desired unit of data without considering its size. The index registers automatically align themselves appropriately; thus, the same index register may be used on arrays with different data sizes. For example, in a matrix multiplication of any array of full word, single-precision, fixed-point numbers, the results may be stored in a second array as double-precision numbers, using the same index quantity for both arrays. If an index register contains the value of k, then the user always accesses the kth element, whether it is a byte, halfword, word, or doubleword. Incrementing by various quantities according to data size is not required; instead, incrementing is always by units in a continuous array table regardless of the size of data element used.

Instruction Set. More than 100 major instructions permit short, highly optimized programs to be written. These are rapidly assembled and minimize both program space and execution time.

Translate Instruction. The Translate instruction permits rapid translation between any two 8-bit codes; thus, data from a variety of input sources can be handled and reconverted easily for output.

Conversion Instructions. Two generalized conversion instructions provide for bidirectional conversions between internal binary and any other weighted number system, including BCD.

Call Instructions. These four instructions permit handling up to 64 user-defined subroutines, as if they were built-in machine instructions. Call instructions also gain access to specified operating system services without requiring its intervention.

Interpret Instruction. The Interpret instruction simplifies and speeds interpretive operations such as compilation, thus reducing space and time requirements for compilers and other interpretive systems.

Four-Bit Condition Code. Checking results is simplified by automatically providing information on almost every instruction execution, including indicators for overflow, underflow, zero, minus, and plus, as appropriate, without requiring an extra instruction execution.

Direct Input/Output (DIO). Direct input/output facilitates in-line program control of asynchronous or special-purpose devices. This feature permits information to be transmitted directly to or from general-purpose registers.

Multiplexor Input/Output Processor (MIOP). Once initialized, I/O processors operate independently of the basic processor, freeing it to provide faster response to system needs. An MIOP requires minimal interaction with the basic processor. I/O command doublewords permit both command chaining and data chaining without intervening basic processor control. I/O equipment speeds range from slow rates involving human interaction (teletypewriter, for example) to transfer rates of rotating memory devices of over 750,000 bytes per second. Peripheral controllers attached to an MIOP may be operated simultaneously.

Rotating Memory Processor (RMP). An RMP supports up to 15 disk drives, one at a time, permitting large capacity, high transfer rate files. Dual access (between 2 RMPs) option is available.

TIME-SHARING FEATURES

Time-sharing is the ability of a system to share its total resources among many users at the same time. Each user may be performing a different task, requiring a different share of the available resources. Some users may be online in an interactive, "conversational" mode with the basic processor while other users may be entering work to be processed that requires only final output.

Time-sharing features are described in the following paragraphs.

Rapid Context Saving. When changing from one user to another, the operating environment can be switched quickly and easily. Stack-manipulating instructions permit storing in a push-down stack of 1 to 16 general-purpose registers by a single instruction. Stack status is updated automatically and information in the stack can be retrieved when needed

(also, by a single instruction). The current program status words, which contain the entire description of the current user's environment and mode of operation, may be stored anywhere in memory, and new program status words may be loaded, all with a single instruction.

Multiple Register Blocks. The availability of four blocks of 16 general-purpose registers improves response time by reducing the need to store and load register blocks. A distinct block may be assigned for different functions as needed; the program status words automatically select the applicable register block.

User Protection. The slave mode feature restricts each user to his own set of instructions while reserving to the operating system certain "privileged" (master mode) instructions that could destroy another user's program if used incorrectly. Also, a memory access – protection feature prevents a user from accessing any storage areas other than those assigned to him. It permits him to access certain areas for reading only, such as those containing public subroutines, while preventing him from reading, writing, or accessing instructions in areas set aside for other users.

Storage Management. Main memory is expandable to 256K (K = 1024) words. To make efficient use of available memory, the memory map hardware permits storing a user's program in fragments as small as a page of 512 words, wherever space is available; yet all fragments appear as a single, contiguously addressable block of storage at execution time. The memory map also automatically handles dynamic program relocation so that the program appears to be stored in a standard way at execution time, even though it may actually be stored in a different set of locations each time it is brought into memory. The memory map provides the ability to locate any 128K-word virtual program in the basic processor's logical addressing space. Thus, the system can always address a virtual memory of 128K words regardless of physical memory size.

Input/Output Capability. Time-sharing input/output requirements are handled by the same general-purpose input/output capabilities described under "General-Purpose Features".

Nonstop Operation. A "watchdog" timer assures that the system continues to operate even in case of halts or delays due to failure of special I/O devices. Multiple real-time clocks with varying resolutions permit independent time bases for flexible allocation of time slices to each user.

Reliability, Maintainability, Availability. Since time-sharing systems have many on-line users needing immediate system response, "downtime" defeats time sharing's primary purpose. Pooling of resources along with flexible reconfiguration control ensures a high level of continuous availability. Configuration controls are provided to switch the load from one unit to another in the event of a failure with no loss of functional capability, only capacity. In addition, a nonworking subset of the total system may be logically isolated (partitioned) so that maintenance may proceed on the subset while the remainder of the system continues to operate.

To minimize the effect of transient errors, automatic retry of failed instructions is performed.

REAL-TIME FEATURES

Real-time applications are characterized by a need for: (1) hardware that provides quick response to an external environment; (2) speed that is sufficient to keep up with the real-time process itself; (3) input/output flexibility to handle a wide variety of data types at different speeds; and (4) reliability features to minimize irreplaceable lost time.

Multilevel, Priority Interrupt System. The real-time-oriented system provides rapid response to external interrupt levels. Each interrupt is automatically identified and responded to according to its priority. For further flexibility, each level can be individually disarmed (to discontinue input acceptance) and disabled (to defer responses). Use of the disarm/disable feature makes programmed dynamic reassignment of priorities quick and easy, even while a real-time process is in progress.

Programs involving interrupts from specially designed equipment often require checkout before the equipment is actually available. To permit simulating this special equipment, any external interrupt level can be "triggered" by the basic processor through execution of a single instruction. This capability is also useful in establishing a modified hierarchy of responses. For example, in responding to a high-priority interrupt, after the urgent processing is completed, it may be desirable to assign a lower priority to the remaining portion so that the interrupt routine is free to respond to other critical stimuli. The interrupt routine can accomplish this by triggering a lower-priority level, which processes the remaining data only after other interrupts have been handled.

READ DIRECT and WRITE DIRECT instructions (described in Chapter 3) allow the program to completely interrogate, preserve, and alter the condition of the interrupt system at any time and to restore that system at a later time.

Nonstop Operation. When connected to special devices (on a ready/resume basis), the basic processor may be excessively delayed if the specific device does not respond quickly. As in the time-sharing environment, the built-in watchdog timer assures that the basic processor cannot be delayed for an excessive length of time.

Real-Time Clocks. Many real-time functions must be timed to occur at specific instants. Other timing information is also needed – for example, elapsed time since a given event, or the current time of day. The computer system can contain up to four real-time clocks with varying degrees of resolution to meet these needs. These clocks also allow easy handling of separate time bases and relative time priorities.

Rapid Context Switching. When responding to a new set of interrupt-initiated circumstances, a computer system must preserve the current operating environment, for continuance later, while setting up the new environment. This changing of environments must be done quickly, with a minimum of "overhead" time costs. Any one of the four blocks of general-purpose arithmetic registers can, if desired, be assigned to a specific environment. All relevant information about the current environment (instruction address, current general register block, memory-protection key, etc.) is kept in the program status words. A single instruction stores the current program status words anywhere in memory and loads new ones from memory to establish a new environment, which includes information identifying a new block of general-purpose registers. Thus, the system's operating environment can be preserved and changed completely through the execution of a single instruction.

Memory Protection. Both foreground (real-time) and background can run concurrently in the system because a foreground program is protected against destruction by an unchecked background program. Under operating system control, the memory access-protection feature prevents accessing memory for specified combinations of reading, writing, and instruction acquisition.

Variable Precision Arithmetic. Much of the data encountered in real-time systems are 16 bits or less. To process this data efficiently, both halfword and fullword arithmetic operations are provided. For extended precision, doubleword arithmetic operations are also included.

Direct Input/Output. For handling asynchronous I/O, a 32-bit word can be transferred directly between any general-purpose register and external devices.

Reliability, Maintainability, Availability. The capabilities described in the section, "Time-Sharing Features" apply equally to the real-time environment.

MULTIUSE FEATURES

As implemented in this system, "multiuse" combines two or more application areas. The real-time application is the most difficult general computing task because of its severe requirements. Similarly, another difficult multiuse task is a time-sharing application that includes one or more real-time processes. Because the system is designed on a real-time base, it is qualified for a mixture of applications in a multiuse environment. Many hardware features that prove valuable for certain application areas are equally useful in others, although in different ways. This multiple capability makes the system particularly effective in multiuse applications.

The major multiuse features are described in the following paragraphs.

Priority Interrupt System. In a multiuse environment, many elements operate simulatneously and asynchronously. Thus, an efficient priority interrupt system is essential. It allows the computer system to respond quickly, and in proper order, to the many demands made on it, with attendant improvements in resource efficiency.

Quick Response. The many features that combine to produce a quick-response system (multiple register blocks, rapid context saving, multiple push-pull operations) benefit all users because more of the system's resources are readily available at any instant.

Memory Protection. The memory protection features protect each user from every other user and guarantee the integrity of programs essential to critical real-time applications.

Input/Output. Because of the wide range of capacities and speeds, the I/O system simultaneously satisfies the needs of many different application areas economically, both in terms of equipment and programming.

Instruction Set. The comprehensive instruction set provides the computational and data-handling capabilities required for widely differing application areas; therefore, each user's program length and running time is minimized, and the throughput is maximized.

MULTIPROCESSOR FEATURES

System design readily permits expansion to shared memory in a multiprocessor system. The system can contain a combination of functional clusters, each of which in turn may contain multiple processors. The total number of clusters is restricted to the maximum port limitation of six. All processors in a system may share common memory.

The following paragraphs describe the major multiprocessor features of the system.

MULTIPROCESSOR INTERLOCK

In a multiprocessor system, the basic processors often need exclusive control of a system resource. This resource may be a region of memory, a particular peripheral device, or, in some cases, a specific software process. There is a special instruction to provide this required multiprocessor interlock. This special instruction, LOAD AND SET, unconditionally sets a "1" bit in the sign position of the referenced memory location during the restore cycle of the memory operation. If this bit had been previously set by another processor, the interlock is said to be "set" and the testing program proceeds to another task. On the other hand, if the sign bit of the tested location is a zero, the resource is allocated to the testing processor, and simultaneously the interlock is set for any other processor.

MULTIPORT MEMORY SYSTEM

The system has growth capability of up to 6 ports per memory unit. A memory unit may contain 16K or 32K words. This architecture allows flexibility in growth patterns and provides high memory bandwidth, essential to multiprocessor systems.

MANUAL PARTITIONING CAPABILITY

Manual partitioning capability is afforded for all system units. Thus, besides the primary advantage of increased throughput, a secondary advantage of a multiprocessor system is the "fail-soft" ability. Given a duplicate unit, any unit can be partitioned by selectively disabling it from the system buses. Depending on the type of failing unit, the system will be operable, with some degree of degraded performance. An alternate processor bus with dual system capabilities can be provided.

MULTIPROCESSOR CONTROL FUNCTION

A multiprocessor control function is provided on all multiprocessor systems. This function provides these basic features:

1. Control of the External Direct Input/Output bus (External DIO), used for controlling system maintenance and special purpose units such as analog to digital converters.

2. Central control of system partitioning.

3. Centralized interrupt system, providing capability for the operating system to use interrupts to schedule tasks independently of the number of basic processors present in a system.

4. Processor to processor communication via processor buses.

SHARED INPUT/OUTPUT

In a multiprocessor system, any basic processor may direct I/O actions to any I/O processor. Specifically, any basic processor can issue an SIO, TIO, TDV, or HIO instruction to begin, test, or stop any I/O process. However, the "end-action" sequence of the I/O process is directed to one of the basic processors in the system by the System Control Processor. This feature (accomplished by setting a pair of configuration control switches) allows dedicating I/O end-action tasks to a single processor and avoids conflict resolution problems.

2. SYSTEM ORGANIZATION

The elements of this computer system include a basic processor (BP), input/output processors (IOPs), memory, I/O device controllers, and devices (see Figure 1). The processors and interfaces clustered into functional groups, interconnected via buses and controlled from a Configuration Control Panel and a System Control Processor. Elements within a processor cluster share an access path for intracluster communications. Thus, the total computer system can be viewed functionally as a group of program-controlled processor clusters communicating with each other and a common memory. Each processor cluster operates asynchronously and semi-independently, automatically overlapping the operation of elements within as well as the operation of other processor clusters for greater speed (when circumstances permit).

PROCESSOR CLUSTERS

Processors (basic processor and MIOP, for example) are grouped functionally along with a Memory Interface (MI) and a Processor Interface (PI) into a processor cluster. Elements within a processor cluster share an access path (the cluster bus) to the Memory Interface, which connects to the memory system via a memory bus. The Memory Interface resolves contention problems and controls use of the cluster bus by the elements in the cluster.

A processor communicates with processors in other processor clusters through the Processor Interface, which connects directly to a processor bus. Via the processor bus, any processor can communicate with or control any other processor anywhere in the system configuration.

Note: Although two processor buses are provided, a Processor Interface can be connected to one or the other of the processor buses, but not to both at the same time.

Within a basic processor-MIOP processor cluster, the basic processor primarily performs overall control and data reduction tasks whereas the MIOP performs the task associated with the exchange of digital information between main memory and selected peripheral devices. The MIOP communicates with device controllers via the I/O bus, which connects to the Controller Interface (CI).

SYSTEM CONTROL PROCESSOR

The System Control Processor performs these primary functions in the overall system:

1. System control.
2. External Control Subsystem.
3. Internal and external interrupt processing.
4. External and certain internal direct I/O (DIO) control.

It provides these major interfaces with other parts of the system:

1. System console interface.
2. System control bus interface.
3. Processor bus interface.
4. Internal and external interrupt interfaces.
5. External and certain internal DIO interfaces.
6. System clock interface.

In addition to these major interfaces it provides paths for other signals including system reset, 1.024 MHz clock, power on/power off trap requests, and external real-time clocks.

Figure 1 shows the interconnection of a System Control Processor to processor clusters via a processor bus as well as interconnection to the system console, external Direct Input/Output (DIO), and external interrupts.

BASIC PROCESSOR

This section describes the organization and operation of the basic processor in terms of instruction and data formats, information processing, and program control. The basic processor comprises a fast memory and an arithmetic and control unit as functionally shown in Figure 2.

Note: Functionally associated with the basic processor but physically located elsewhere are a memory map, memory access protection codes, and memory write protection codes. Memory control storage for the memory map and access codes is located in the Memory Interface, and the memory control storage for the write protection codes (write locks) is located in the memory. These functions are described in "Memory System", later in this chapter.

GENERAL REGISTERS

A fast (integrated circuit) memory consisting of ninety-six 32-bit registers is used within the basic processor. A group of 24 registers is referred to as a register block; thus, a basic processor contains four register blocks. A 2-bit control field (called a register block pointer) in the program status words (PSWs) selects the register block currently

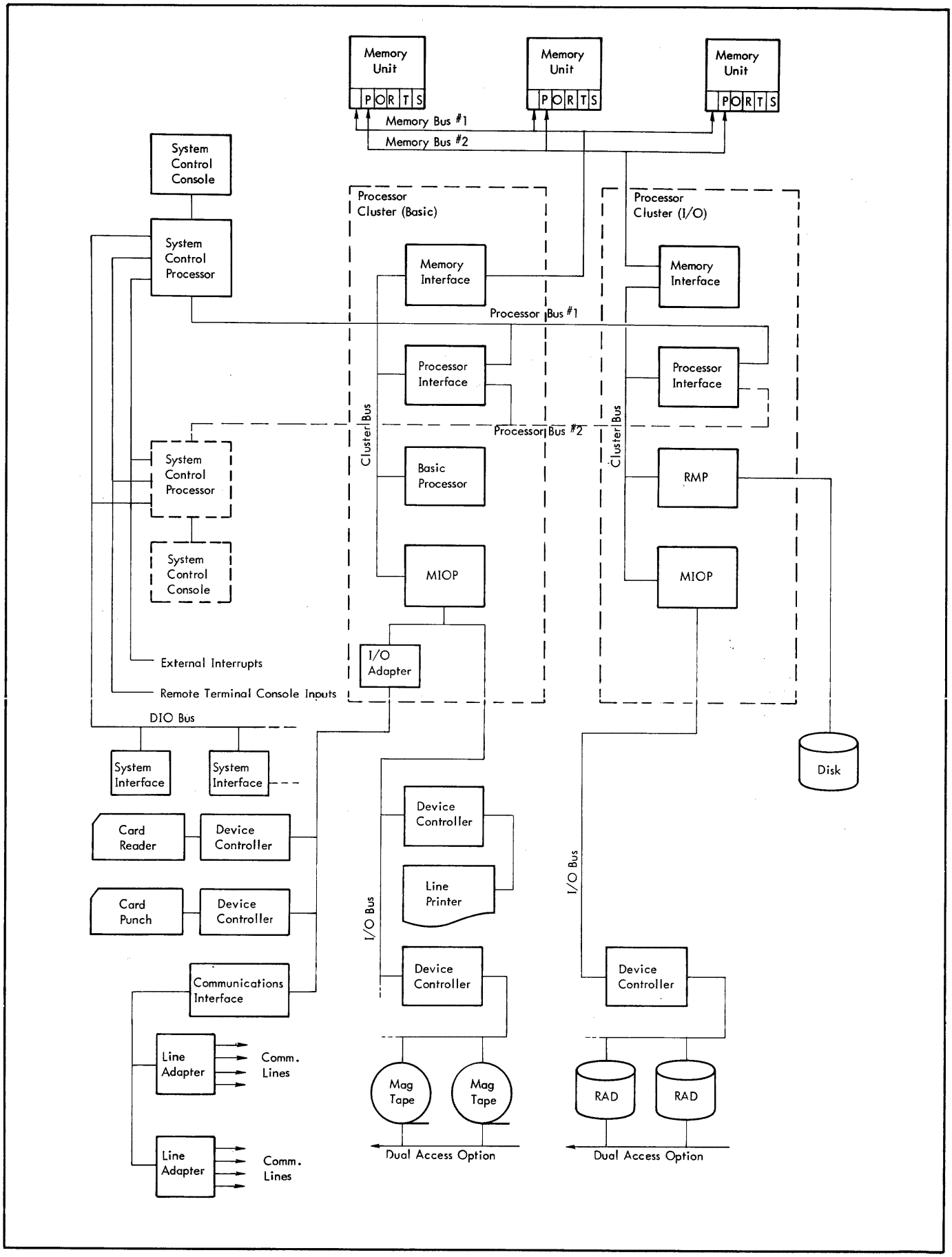


Figure 1. A Xerox 560 Computer System

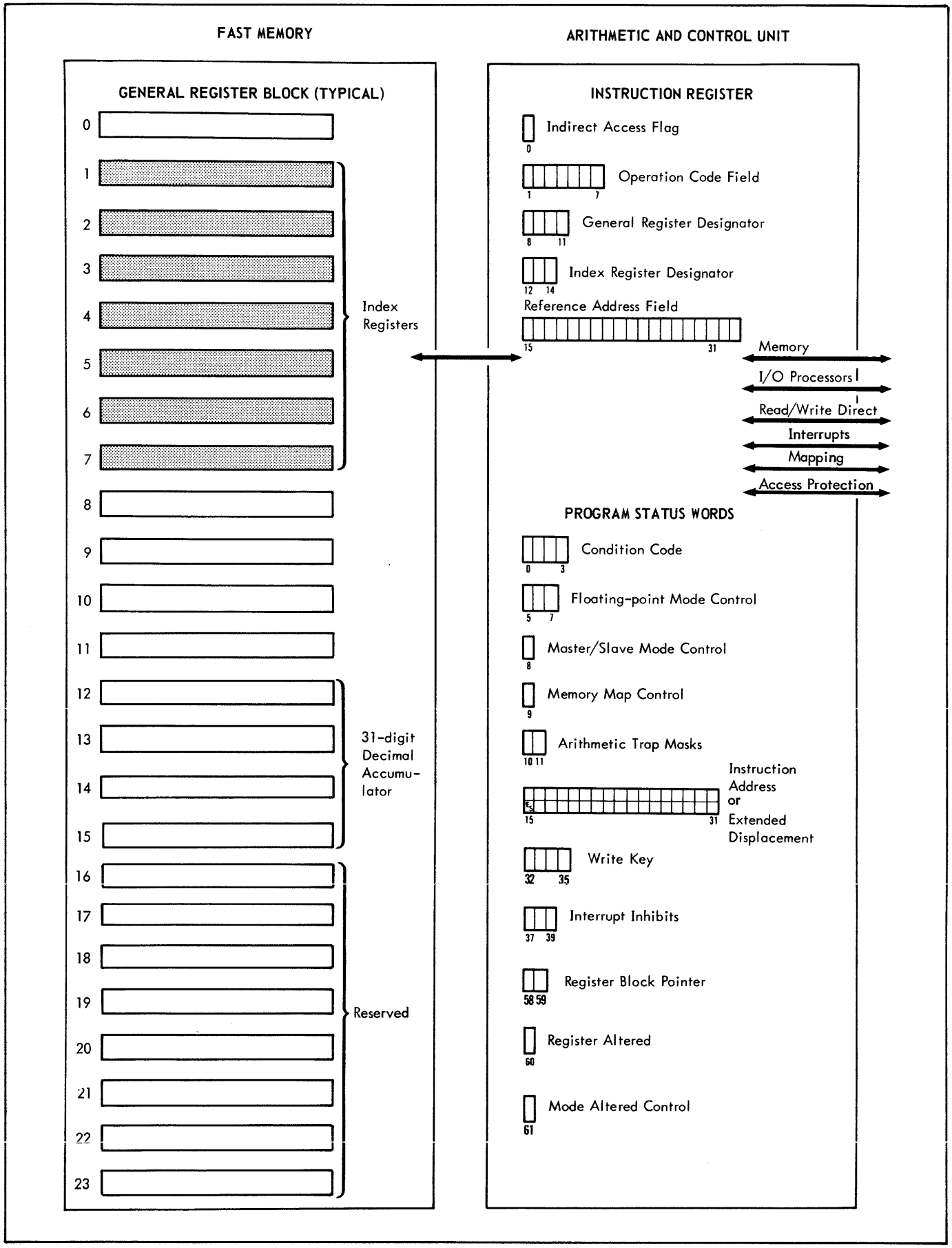


Figure 2. The Basic Processor

available to a program. The register block pointer can be changed when the basic processor is in the master mode or the master-protected mode. Only the first 16 general registers of a register block may be used by programs; the last eight are reserved.

Each of the first 16 general registers in a register block is identified by a 4-bit code in the range 0000_2 through 1111_2 (0 through 15 in decimal notation, or X'0' through X'F' in hexadecimal notation). Any of these 16 registers can be used as a fixed-point temporary data storage location, or to contain control information such as a data address, count, pointer, etc. General registers 1 through 7 can be used as index registers and registers 12 through 15 can be used as a decimal accumulator capable of containing a decimal number of 31 digits plus sign. Registers 12 through 15 are always used when a decimal instruction is executed.

MEMORY CONTROL STORAGE

The memory control storage for the memory map and the associated memory access protection codes are contained in the Memory Interface (MI). Memory control storage for the 4-bit write locks are contained in the memory units. Memory control storage can be modified when the basic processor is in the master mode or the master-protected mode.

MEMORY MAP

Two terms are essential in understanding the memory mapping concept: actual (i.e., absolute or real) address and virtual address.

An actual address is used within the memory unit (memory address registers) to access a specific, physical memory location for storage or retrieval of information as required by the execution sequence of an instruction. Actual addresses are fixed and are dependent on the wired-in hardware.

A virtual address refers to a logical location as required by an individual program. Like an actual address, a virtual address may designate a location that contains a program instruction, an element of data, a data address (indirect address), or it may also be an explicit quantity. Normally, virtual addresses are derived from programmer-supplied labels through an assembly (or compilation) process followed by a loading process. Virtual addresses may also be computed during a program's execution. Virtual addresses include all instruction addresses, indirect addresses, and addresses used as counts within a stored program, as well as those instructions computed by the program. (See "Virtual and Real Memory", later in this chapter.)

Memory mapping transforms virtual addresses as seen by the individual program into actual addresses as seen by the memory system. Thus, when the memory map is in effect, any program can be broken into 512-word pages and dynamically relocated throughout memory in whatever pages of space are available.

When the memory map is not in effect, all virtual address values above 15_{10} are used by the memory as actual addresses. Virtual addresses 0 through 15 are always[†] used by the basic processor as general register addresses rather than as memory addresses. For example, if an instruction uses virtual address 5 to address the location where a result is to be stored, the basic processor stores that result in general register 5 in the current register block instead of in memory location 5.

When the basic processor is operating with the memory map in effect, virtual addresses 0 through 15 are still used as general register addresses. Virtual addresses above 15 are transformed into actual addresses by replacing the high-order portion of the virtual address with a value obtained from the memory map. (The memory map address replacement process is described in "Memory Address Control", later in this chapter.)

MEMORY ACCESS PROTECTION

When the basic processor is operating with the memory map in the slave mode or the master-protected mode, the access protection codes determine whether the program may access instructions from, read from, or write into specific regions of the virtual address continuum (virtual memory). If the slave mode or master-protected mode program attempts to access a protected region of virtual memory, a trap occurs (see "Memory Address Control", "Virtual and Real Memory", and "Trap System", later in this chapter).

MEMORY WRITE PROTECTION

The memory write-protection feature operates independently of access protection and the memory map. The 4-bit write lock operates in conjunction with a 4-bit field, called the write key, in bits 32-35 of the Program Status Words (PSWs). The lock and the key determine whether any program may alter any word of main memory. The write key can be changed when the basic processor is in the master mode or the master-protected mode. (The functions of the write lock and key are described in "Memory Address Control", later in this chapter.)

COMPUTER MODES

The basic processor operates in one of three modes: master, master-protected, or slave. The operation mode is determined by the setting of three bits (bits 8, 9, and 61) of the Program Status Words (PSWs). (See "Program Status Words", later in this chapter.) Additionally, the basic processor operates in a mapped mode or an unmapped mode.

[†]Except for the READ DIRECT (RD)/WRITE DIRECT (WD) instructions which can read from and store into these locations.

MASTER MODE

The master/slave control bit (bit 8 of the PSWs) must contain a zero for the basic processor to operate in master mode. In this mode the basic processor can perform all of its control functions and can modify any part of the system. The restrictions upon the basic processor's operations in this mode are those imposed by the write locks on certain protected parts of memory. It is assumed that there is a resident operating system (operating in the master mode) that controls and supports the operation of other programs (which may be in the master, master-protected, or slave mode).

MASTER-PROTECTED MODE

The master-protected mode of operation provides additional protection for programs that operate in the master mode. The master-protected mode occurs when the basic processor is operating in the master mode with the memory map in effect and the mode altered control bit (bit 61 of the PSWs) is on. In this mode the memory protection violation trap occurs (location X'40', with CC4 = 1), as it does in all mapped slave programs, if a program makes a reference to a virtual page to which access is prohibited by the current setting of the access protection codes.

SLAVE MODE

The slave mode of operation is the problem-solving mode of the basic processor. In this mode, access protection codes apply to the slave mode program if mapping is in effect, and all "privileged" operations are prohibited. Privileged operations are those relating to input/output and to changes in the fundamental control state of the basic processor. All privileged operations are performed in the master or master-protected mode by a group of privileged instructions. Any attempt by a program to execute a privileged instruction while the basic processor is in the slave mode results in a trap. The master/slave mode control bit (bit 8 of the PSWs) can be changed when the basic processor is in the master or master-protected mode. Nevertheless, a slave mode program can gain direct access to certain executive program operations by means of CALL instructions. The operations available through CALL instructions are established by the resident operating system.

MAPPED MODE

Although the memory map is located in the Memory Interface (MI), it functions as part of the basic processor. The basic processor communicates with memory through the MI. Mapping is effective for all the words of real memory, and is invoked when bit 9 (MM) of the PSWs contains a one. Memory mapping generates real page addresses from virtual addresses. The memory map can be loaded with either 11-bit real page addresses or 8-bit real page addresses by means of the MOVE MEMORY CONTROL (MMC) privileged instruction (see Chapter 3, "Control Instructions"). Eleven-bit real page addresses are always provided for in the map, thus if 8-bit real page addresses are generated, the three high-order bits contain zeros. The memory map always maps 17-bit virtual addresses into 20-bit real addresses (see "Memory Address Control", later in this chapter for a discussion of how the map is used).

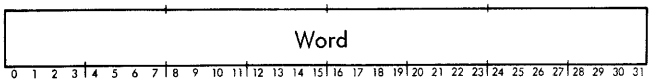
UNMAPPED MODE

When the basic processor is operating in the unmapped mode, there is a direct one-to-one relationship between the effective virtual address of each instruction and the actual address used to access main memory. (See "Real Addressing", later in this chapter.)

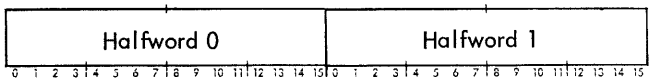
INFORMATION FORMAT

Nomenclature associated with digital information within the computer system is based on functional and/or physical attributes. A "word" may be either a 32-bit instruction word or a 32-bit data word.

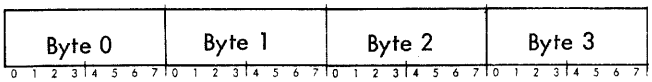
The bit positions of a word are numbered from 0 through 31 as follows:



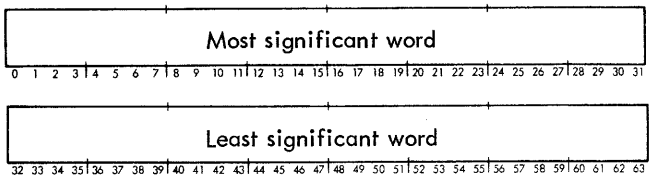
A word can be divided into two 16-bit parts (halfwords) in which the bit positions are numbered from 0 through 15 as follows:



A word can also be divided into four 8-bit parts (bytes) in which the bit positions are numbered 0 through 7 as follows:



Two words can be combined to form a 64-bit element (a doubleword) in which the bit positions are numbered 0 through 63 as follows:



In fixed-point binary arithmetic each element of information represents numerical data as a signed integer (bit 0 represents the sign, remaining bits represent the magnitude, and the binary point is assumed to be just to the right of the least significant or rightmost bit). Negative values are represented in two's complement form. Other formats required for floating-point and decimal instructions are described in Chapter 3.

INFORMATION BOUNDARIES

Basic processor instructions assume that bytes, halfwords, and doublewords are located in main memory according to the following boundary conventions:

1. A byte is located in bit positions 0 through 7, 8 through 15, 16 through 23, and 24 through 31 of a word.
2. A halfword is located in bit positions 0 through 15 and 16 through 31 of a word.
3. A doubleword is located such that bit positions 0 through 31 are contained within an even-numbered word, and bit positions 32 through 63 are contained within the next consecutive word (which is odd-numbered).

Figure 3 illustrates these boundaries.

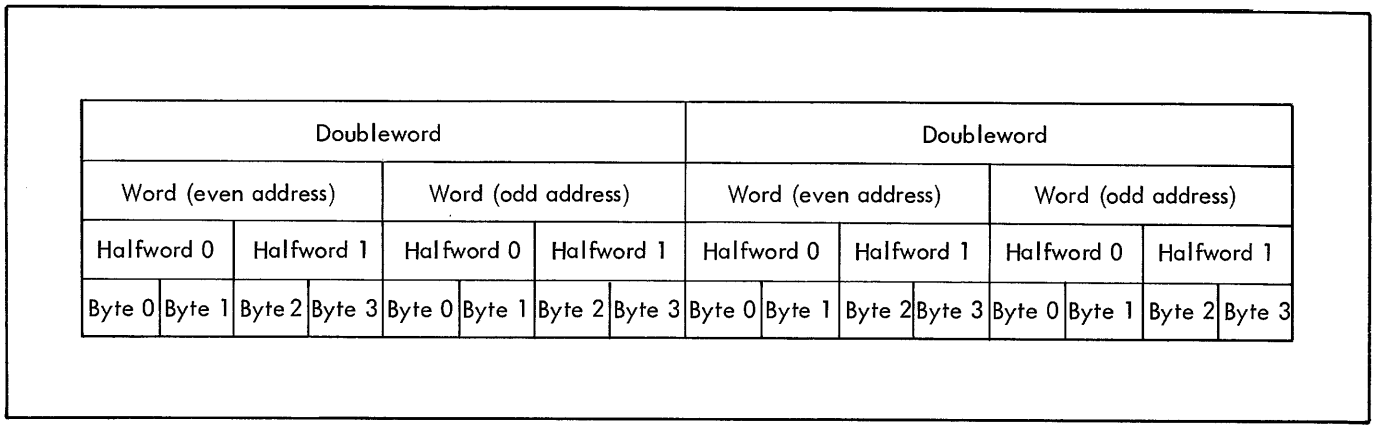


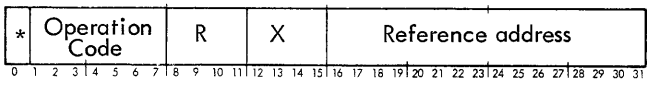
Figure 3. Information Boundaries

INSTRUCTION REGISTER

The instruction register contains the instruction the basic processor is currently executing. The format and fields of the two general types of instructions (memory reference and immediate operand) are described below. Specific formats for each instruction are given in Chapter 3.

MEMORY REFERENCE INSTRUCTIONS

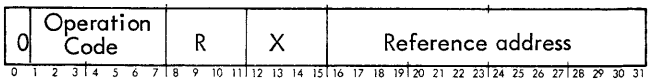
Instructions that make reference to an operand in main memory may have the following format:



Bits	Description
0	Indirect addressing. One level of indirect addressing is performed only if this bit position contains a one.
1-7	Operation code. This 7-bit field contains the code that designates the operation to be performed. See the inside front and back covers for complete listings of operation codes.
8-11	R field. For most instructions this 4-bit field designates one of the first 16 general registers of the current register block as an operand source, result destination, or both.
12-14	X field. This 3-bit field designates one of general registers 1-7 of the current register block as an index register. If X contains zero, indexing will not be performed; hence register 0 cannot be used as an index register. (See "Address Modification Example: Indexing (Real and Virtual Addressing)", later in this chapter for a description of the indexing process.)
15-31	Reference address. This 17-bit field normally contains the reference address of the instruction operand. The reference address is translated into an effective virtual address in accordance with the addressing type (real, real extended, or virtual) and the address modification required (direct/indirect or indexing). (See "Memory Reference Addresses" later in this chapter.)

IMMEDIATE OPERAND INSTRUCTIONS

Immediate operand type instructions are particularly efficient because the required operand is contained within the instruction word. Hence, memory reference, indirect addressing, and indexing are not required.



Bits	Description
0	Bit position 0 must be coded with a zero. If it contains a one, the instruction is interpreted as being nonexistent. (See "Trap System", later in this chapter.)

Bits	Description

1-7 Operation code. This 7-bit field contains the code that designates the operation to be performed. When the basic processor encounters any immediate operand operation, it interprets bits 12-31 of the instruction word as an operand. These are the immediate operand operation codes:

Operation Code	Instruction Name	Mnemonic
X'02'	Load Conditions and Floating Control Immediate	LCFI
X'20'	Add Immediate	AI
X'21'	Compare Immediate	CI
X'22'	Load Immediate	LI
X'23'	Multiply Immediate	MI

8-11 R field. This 4-bit field designates one of the first 16 general registers in the current general register block. The register may contain another operand and/or be designated as the register in which the results of the operation are to be stored or accumulated.

12-31 Operand. This 20-bit field contains the immediate operand. Negative numbers are represented in two's complement form. For arithmetic operations bit 12 (the sign bit) is extended by duplication to the left through bit position 0 to form a 32-bit operand.

The byte-string instructions (described in Chapter 3) are similar to immediate-operand instructions in that they cannot be modified by indexing. Nevertheless, the operand field of byte-string instructions contains either a byte address displacement or a byte address that is a virtual address subject to modification by the memory map. If a byte-string instruction has a one in bit position zero, the basic processor treats it as a nonexistent instruction (see "Trap System", later in this chapter).

MAIN MEMORY

The memory system comprises memory units, memory interfaces (MIs), and memory buses. Figure 4 illustrates the relationships among these components.

The primary technology for main memory is magnetic core. The maximum physical storage is 256K words. Memory units can be interleaved on a two-way interleave basis. Each memory unit is provided with a set of starting address switches on the Configuration Control Panel (see Chapter 6) together with a two-position switch that selects one of two possible clock and power sources. Memory units may contain two, four, or six ports, which have a fixed priority order for the resolution of contention problems.

The following sections describe the organization and operation of the memory system. Also described are the various modes and types of addressing, including indexing.

MEMORY UNIT

Main memory is divided physically and logically into one to eight module assemblies called memory units. Because the memory unit is a logical component that contains all the functions available in the entire memory, the minimum memory is one memory unit. The minimum storage capacity per memory unit is 16K words; the maximum is 32K words. A memory location stores a word of 36 bits; the first 32 bits are information and the last 4 are byte parity bits (the latter being unavailable to the program). Each memory unit comprises a specific storage capacity, drive and sense circuits, a set of operational registers (address, data, and status), a set of write lock control registers for 32K words of memory, and a timing and control unit.

CORE MEMORY MODULES

Core memory modules (CMMs) provide a storage facility of standard modules (see Figure 4).

MEMORY DRIVER

The memory driver in each memory unit performs all memory operations except storage (provided for by the CMMs) and the few operations performed by the ports. The major functions of the memory driver are:

1. Store address word.
2. Store data-in and data-out words during memory cycles.
3. Store write locks in special memory (other than CMMs).
4. Perform parity generation and checking on address and memory bus data words, and on core memory module words.
5. Generate and store status words.
6. Control and time all transfers of address words, data words, status words, write locks, and write key among the ports, CMM, and the storage registers.
7. Control and time all data, parity, and control signals issued to the memory bus.
8. Accept one of two or more simultaneous memory requests on the basis of port positional priority and other priority status information such as "high priority" and "memory reserved".

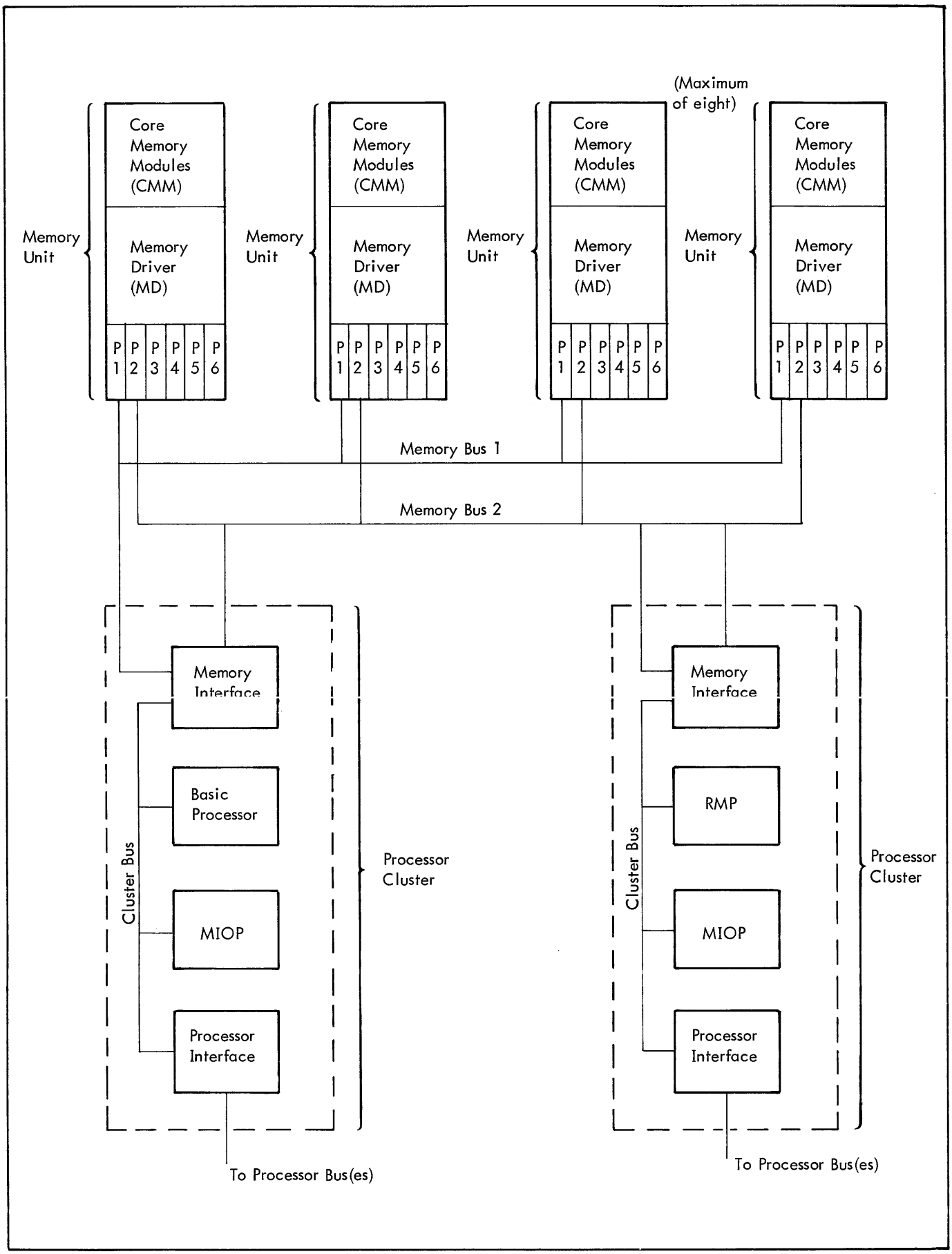


Figure 4. Main Memory

PORTS AND MEMORY BUSES

A memory unit may contain two, four, or six ports, which have a fixed priority order for the resolution of access contention. Each port allows the memory unit to communicate via a memory bus with a different external system (i.e., a processor cluster), which communicates with the memory bus via the Memory Interface (MI) (see Figure 4). Ports are numbered from 1 (top priority) to 6 (lowest priority). The selection logic is biased to select port 1 (the fast port) whenever the memory is quiescent. Thus performance is improved for the Memory Interface (MI) connected to that port, and hence to the processors connected to that MI.

A memory reserve function insures proper execution of instructions that require guaranteed re-access to a memory location before a second processor can access it.

Each port is equipped with an inhibit function that can be activated from the Configuration Control Panel (see Chapter 6).

Other major functions performed by the ports are:

1. Address recognition.

2. Address interleaving.

The memory system is built up by interconnection of identically numbered ports of all memory units. Each interconnecting cable is called a memory bus, which is dedicated to a single processor cluster (see Figure 4).

PORT PRIORITY

The multiport structure allows two simultaneous requests for memory to be processed immediately if the requests are received on different ports for different memory units, and neither memory unit is busy. If a requested memory unit is busy or receives simultaneous requests, the memory port logic selects the highest priority request first.

Normally, all ports in a memory unit operate on the fixed priority basis (the fast port has the highest priority and the highest-numbered normal port the lowest). Thus, if a single memory unit simultaneously receives requests on port 2 and port 4, port 2 has first access to the memory unit.

Each port also has associated with it a high-priority line which, upon receiving a high-priority request, raises the port's priority above that of all other ports except for any higher priority port, which also has a high-priority request on its line.

MEMORY INTERLEAVING

Memory interleaving is a hardware feature that distributes sequential addresses into two independently operating memory units. Interleaving increases the probability that a processor (i.e., basic processor, RMP, or MIOP) can gain access to a given memory location without encountering interference from another processor that is making sequential requests.

Two memory units of the same size can be two-way interleaved. Both memory units transform an incoming address, as follows:

Size of Each Memory Unit	Address Bits Interchanged
32K	16 and 31
16K	17 and 31

As a result of the address transformation, even incoming addresses are assigned to one memory unit and odd incoming addresses to the other. Note that the incoming address (untransformed) is stored in the status register of the accessed unit in each cycle and is available as are other types of dynamic status information. (Interleaved memory units have two status registers, one in each of the units.)

MEMORY UNIT STARTING ADDRESS

Each memory unit is individually identified by starting address switches located on the Configuration Control Panel (see Chapter 6). These switches define the range of addresses the memory unit responds to when servicing memory requests. All addresses, including the starting address, for a given memory unit are the same for all ports in that unit; that is, the address of a given word remains the same regardless of the port used to access the word. The starting address of a memory unit must be on a boundary equal to a multiple of the size of the memory unit when two memory units (of the same size) are interleaved. The starting address of one memory unit must be a multiple of the size of the two memory units together; the second memory unit must have a starting address higher than that of its companion by its own size. Another way to say this is that the starting address for the combined units must be on a boundary equal to a multiple of the total size of the interleaved assembly.

MAINTAINABILITY AND PERFORMANCE

Memory maintainability is enhanced by the following features:

1. Error detection. Each memory unit senses and remembers parity errors in the CMM data as well as parity errors in the address word or the memory bus data, port selection errors, CMM selection error, and undefined operations. This status information is available to diagnostic programs to facilitate error localization in space and time of occurrence. The memory unit senses and reports, but does not remember (for diagnostic purposes) a write lock violation.

2. Modularity. For ease of replacement, the logic and storage circuitry is packaged on modules that are removable from backpanels without requiring cable disconnections.

3. Diagnostic logic. Each memory driver module carries logic used exclusively for localizing faulty elements in that module. The benefit derived from this diagnostic logic depends on such external factors as the accessibility to a module tester.

Memory system performance depends on these factors:

1. Access time of memory unit.
2. Cycle time of memory unit.
3. Type of cycle requested.
4. Number of memory units.
5. Interleaving.
6. Type of port (fast or normal) selected.
7. Self or mutual interference between memory requests.

All these factors characterize not only memory performance but also system performance.

Port access time and cycle time are essential memory speed characteristics pertaining to CMM operations.

1. Port access time. This is the time interval measured between the clock pulse that transmits an address word from the Memory Interface (MI) to an idle memory unit and the clock pulse that translates a memory word from the same memory unit to the MI.
2. Cycle time. Cycle time depends on the operation being performed and on the sequence of operation. Cycle time determines the maximum rate at which a memory unit can accept requests.

VIRTUAL AND REAL MEMORY

Virtual memory is the address space available to an individual program. The maximum size of virtual memory is 128K words, broken into as many as 256 pages of 512 words each distributed throughout the available pages of real memory.

Real memory corresponds to the physical memory, and its size is equal to the total number of words contained within all memory units in the system. The size of real memory ranges from a minimum of 16K words to a maximum of 256K words.

Note: Real memory address space is 1 million words.

MEMORY REFERENCE ADDRESS

Memory locations 0 through 15 are not normally accessible to the programmer because their memory addresses are reserved as register designators for "register-to-register" operations. Nevertheless an instruction treats any of the first 16 registers of the current register block as if it were a location in main memory. Furthermore, the register block can hold an instruction (or a series of as many as 16 instructions) for execution just as though the instruction (or instructions) were in main memory.

The following terms are used in the various types of addressing described in subsequent sections. See also Figure 5, which illustrates the control and data flow during address generation.

1. Instruction Address. This is the address of the next instruction to be executed. For real, real-extended, and virtual addressing the 17-bit instruction address is contained within bits 15-31 of the program status words (PSWs).
2. Reference Address. This is the 17-bit or 20-bit address associated with any instruction (except that in a trap or interrupt location that has a 0 in bit position 10). For real, real extended, direct, and virtual addressing, the reference address is the address contained within bits 15-31 of the instruction itself.

   The reference address may be modified by using indirect addressing, indexing, and memory mapping. A reference address becomes an effective virtual address after the indirect addressing and/or postindexing (if required) is performed.
3. 20-Bit Trap or Interrupt Reference Address. If bit position 10 of any instruction in a trap or interrupt location contains a 0, bits 12-31 of that instruction are used as a 20-bit reference address. This 20-bit reference address can be modified only by using indirect addressing. This 20-bit reference address cannot be indexed or mapped. (See "Interrupt and Trap Entry Addressing", later in this chapter.)
4. Direct Reference Address. If neither indirect addressing nor indexing is called for by the instruction (i.e., if bit 0 and the X field contain zero), the reference address of the instruction (as defined above) becomes the effective virtual address. Direct addressing may be used during real, virtual, or real extended addressing modes, including trap and interrupt operations. Direct addressing during virtual addressing does not preclude memory mapping.
5. Indirect Reference Address. The 7-bit operation code field of the instruction word format provides for as many as 128 instruction operation codes, nearly all of which can use indirect addressing (except immediate-operand and byte-string instructions). If the instruction calls for indirect addressing (bit position 0 contains a 1), the reference address (as defined above) is used to access a word location that contains the direct reference address in bit positions 15-31, or bit positions 12-31 for certain real extended addressing operations. The indirect addressing operation is limited to one level, regardless of the contents of the word location pointed to by the reference address field of the instruction. Indirect addressing occurs before indexing; that is, the 17-bit

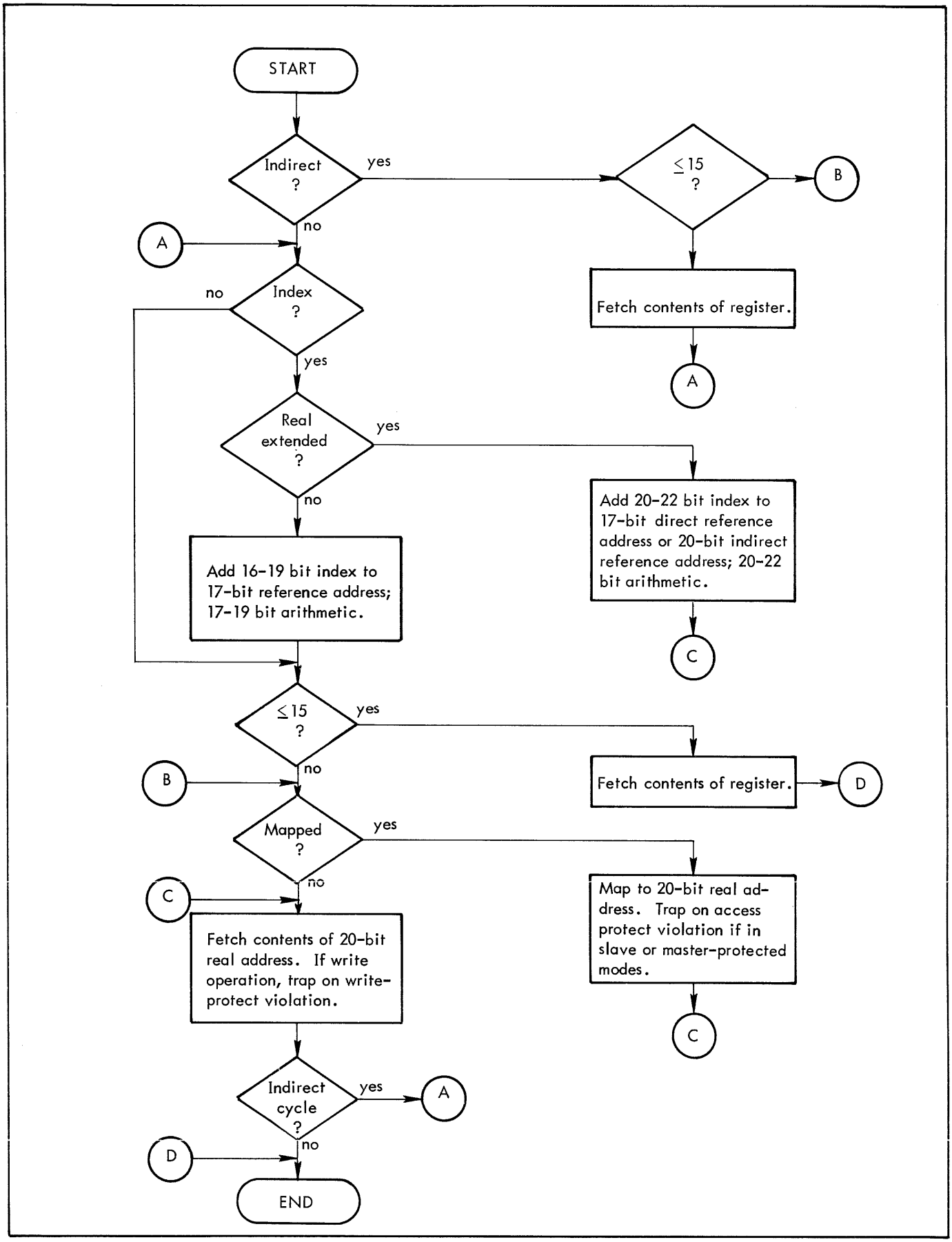


Figure 5. Addressing Logic

reference address field of the instruction is used to obtain a word, and the 17 or 20 low-order bits of the word thus obtained effectively replace the initial reference address field; then indexing is carried out according to the operation code of the instruction. See Figures 7 and 9, later in this chapter.

6. Index Reference Address. If indexing is called for in the instruction (a value other than zero in bits 12-14 of the instruction), the direct or indirect reference address is modified by addition of the displacement value in the general register (index) called for by the instruction (after scaling the displacement according to the instruction type). This final reference address value (after indirect addressing, indexing, or both) is defined as the effective virtual address of the instruction. Indexing after indirect addressing is called postindexing. See also Figures 7 and 9, later in this chapter.

7. Displacements. Displacements are the 16- to 22-bit values used in index registers and by byte-string instructions to generate effective addresses of the appropriate size (byte, halfword, word, or doubleword).

8. Register Address. If any instruction provides a virtual address that is a memory reference (i.e., a direct, indirect, or indexed reference address) in the range 0 through 15, the basic processor does not attempt to read from or write into main memory locations 0 through 15. Instead, the four low-order bits of the reference address are used as a general register address and the general register corresponding to this address is used as the operand location or result destination. Thus, the instruction can use any of the first 16 registers in the current register block as the source of an operand, the location of a direct address, or the destination of a result. Such usage is called a "register-to-register" operation.

9. Actual Address. This is the address value actually used by the basic processor to access main memory via the memory address register (see Figure 5). If the effective virtual address is in the range 0 through 15 (X'0 through X'F'), one of the first 16 general registers in the current register block is being addressed. If the basic processor is operating in the virtual addressing mode, all addresses greater than 15 (X'F') are transformed (usually into addresses in a different memory page) by the memory map into actual addresses. Contrarily, if the basic processor is operating in either real or real extended mode, no transformation via the memory map takes place.

10. Effective Address. The effective address is defined as the final virtual address computed for an instruction. Note, however, that some instructions do not use the effective address as a location reference; instead, the effective address is used to control the operation of the instruction (as in a shift instruction), to designate the address of an input/output device (as in an input/output instruction), or to designate a specific element of the system (as in a READ DIRECT or WRITE DIRECT instruction).

11. Effective Location. An effective location is defined as the actual location (in main memory or in the current register block) that is to receive the result of a memory-referencing instruction, and is referenced by means of an effective address. Because an effective address may be either an actual address or a virtual address, when applicable, this definition of an effective location assumes the transformation of a virtual address into an actual address.

12. Effective Operand. An effective operand is defined as the contents of an actual location (in main memory or in the current register block) that is to be used as an operand by a memory-referencing instruction, and is referred to by means of an effective address. This also presupposes the transformation of a virtual address into an actual address.

TYPES OF ADDRESSING

Except for the special type of addressing performed by some interrupt and trap instructions, all addressing within the computer system is real, real extended, or virtual.

REAL ADDRESSING

In real addressing, a one-to-one relationship prevails between the effective virtual address of each instruction and the actual address used to access main memory. Real addressing has these characteristics:

1. Each reference address is a 17-bit word address.

2. The reference address may be direct or indirect, with or without postindexing.

3. Displacements associated with indexing are automatically aligned, as required, using the full 32-bit contents of the index register. The final result is truncated to the left of the high-order bit of the original 17-bit reference address, and the effective real address is a 16-bit doubleword address, 17-bit word address, 18-bit halfword address, or a 19-bit byte address.

4. If indirect addressing is invoked, the 17-bit reference address in the instruction word is used to access the indirect address word in memory. The low-order 17 bits of this word then replace the reference address of the instruction word in the calculations described in (3), above.

5. Memory mapping and memory access protection are never invoked.

6. Memory write protection is automatically invoked.

7. Leading zeros are automatically appended to the effective address to generate an actual word address as required by the main memory.

8. Real addressing is allowed in master mode and in slave mode, and is specified when bit positions 9 and 61 of the PSWs both contain zero.

VIRTUAL ADDRESSING

Virtual addressing uses the memory map to determine the actual address to be associated with a particular reference address of each instruction. Virtual addressing differs from real addressing in that there is normally no exact relationship between the effective virtual address and the actual address. These are the characteristics of virtual addressing:

1. Each reference address is a 17-bit address.

2. The reference address may be direct or indirect, with or without postindexing.

3. Displacements associated with indexing are automatically aligned, as required, using the full 32-bit contents of the index register. The final result is truncated to the left of the high-order bit of the original 17-bit reference address, and the effective virtual address is a 16-bit doubleword address, 17-bit word address, 18-bit halfword address, or a 19-bit byte address.

4. Virtual memory access protection is always invoked. If the access protection code is invalid, the instruction aborts and traps to location X'40'. (See "Trap System", later in this chapter.)

5. Memory mapping translates the 8 most significant bits of the effective virtual address (the page portion) into an 11-bit page address. This page address is concatenated with the 9 least significant bits of the reference address. The resultant 20-bit word address is the actual address used to access memory. This feature permits any one user at any given time to have a virtual memory of as many as 128K words (256 pages) located throughout real (actual) memory comprising as many as 256K words (512 pages). Although virtual memory may be physically fragmented, it is logically contiguous.

   Note that Sigma 6/7 programs may run on this computer system without requiring change to the mapping structure. The memory map is loaded with 8-bit page addresses (the 3 high-order bits of the 11-bit real page address are reset to zeros). The most significant 8 bits of the effective virtual address are then translated into the designated 8-bit page address.

6. The memory write-protection feature is invoked for the actual address in real memory.

7. Virtual addressing may be used in all modes (master, master-protected, and slave) and is specified when bit 9 of the PSWs contains a one.

ADDRESS MODIFICATION EXAMPLE: INDEXING (REAL AND VIRTUAL ADDRESSING)

Figure 6 shows how the indexing operation takes place during real and virtual addressing operations. The instruction is brought from memory and loaded into a 34-bit instruction register that initially contains zeros in the two low-order bit positions (32 and 33). The displacement value from the index register is then aligned with the instruction register (as an integer) according to the address type of the instruction; that is, if it is a byte operation, the low-order bit of the displacement is aligned with the least significant bit of the 34-bit instruction register (bit position 34). The displacement is then shifted one bit to the left of this position for a halfword operation, two bits to the left for a word operation, and three bits to the left for a doubleword operation. An addition process then takes place to develop a 19-bit address, referred to as the effective address of the instruction. High-order bits of the 32-bit displacement are ignored in the development of this effective address (i.e., the 15 high-order bits are ignored for word operations, the 25 high-order bits are ignored for shift operations, and the 16 high-order bits are ignored for doubleword operations). The displacement value, however, can cause the effective address to be less than the initial reference address (within the instruction) if the displacement value contains a sufficient number of high-order 1_2's (i.e., if the displacement value is a negative integer in two's complement form).

The effective virtual address of an instruction is always a 19-bit byte address value. This value, however, is automatically adjusted to the information boundary conventions. Thus, for halfword operations the low-order bit of the effective halfword address is zero; for word operations the two low-order bits of the effective word address are zeros; and for doubleword operations the three low-order bits of the effective doubleword address are zeros.

In a byte operation with no indexing, the effective byte is the first byte (byte 0 in bit positions 0-7) of a word location; in a halfword operation with no indexing, the effective halfword is the first halfword (halfword 0 in bit positions 0-15) of a word location. A doubleword operation always involves a word at an even numbered address and the word at the next sequential (which is odd numbered) word address. Thus, if an odd numbered word location is specified for a doubleword operation, the low-order bit of the effective address field (bit position 31) is automatically forced to zero. This means that in a doubleword operation an odd numbered word (reference) address designates the same doubleword as the next lower even numbered word address.

In the real addressing mode, the 19-bit effective virtual address is concatenated with 3 leading zeros to form a 22-bit actual address. In the virtual addressing mode, the 8 most significant bits of the 19-bit virtual address are mapped (using the memory map) into the 11-bit actual page address, thus forming a 22-bit actual address.

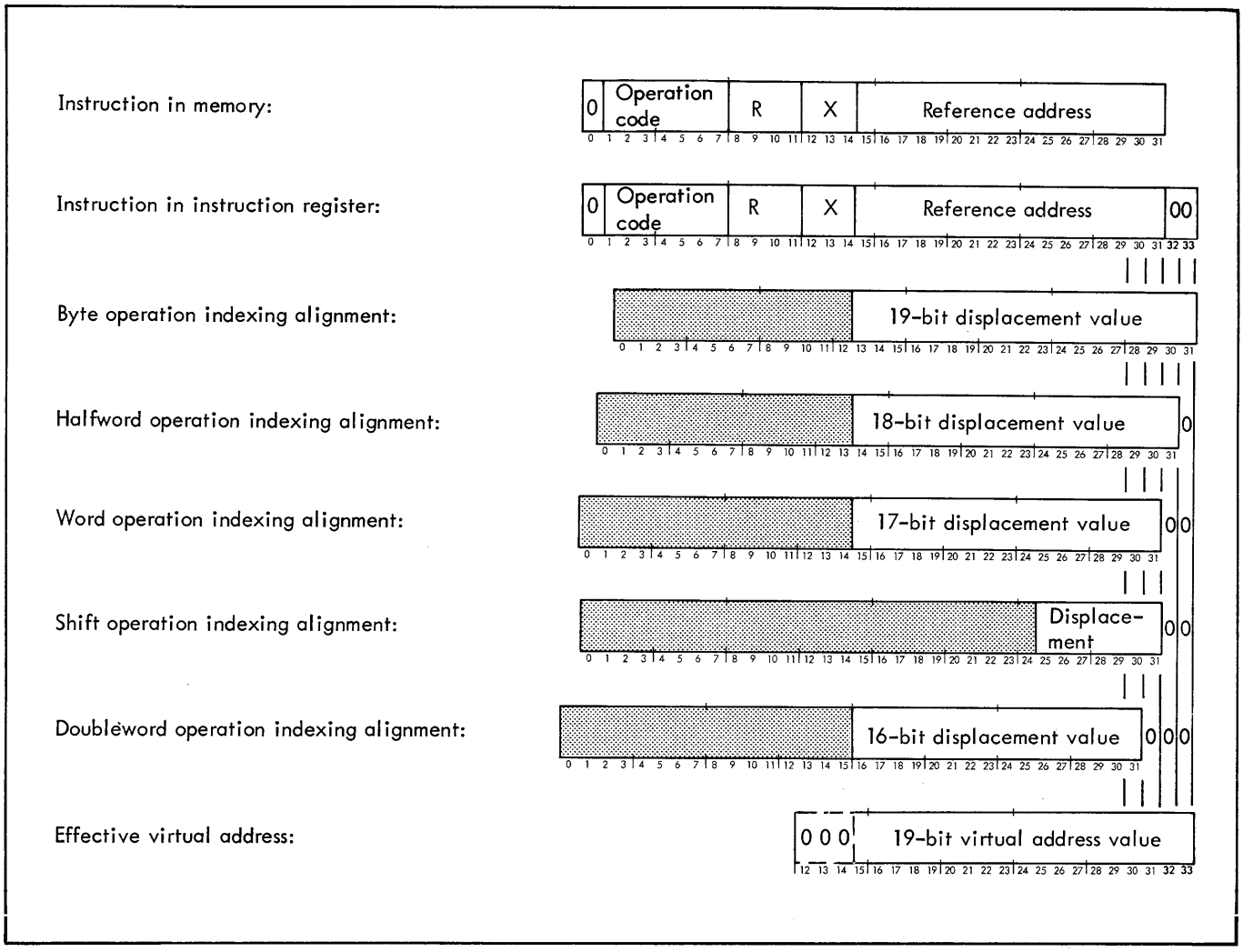


Figure 5. Index Displacement Alignment (Real and Virtual Addressing Modes)

ADDRESS MODIFICATION EXAMPLE: INDIRECT, INDEXED HALFWORD (VIRTUAL ADDRESSING)

Figure 7 illustrates the address modification and mapping process for an indirectly addressed, indexed, halfword operation. As shown, reference address 1 is the content of the reference address field in the instruction stored in memory. The instruction is brought into the instruction register, and if the value of the reference address field is greater than 15, the memory map converts the 19-bit effective virtual address into a 22-bit actual address. The 17 low-order bits of the main memory location pointed to by the actual address, labeled reference address 2, then replaces reference address 1 in the instruction register. The index register designated by the X field of the instruction is subsequently aligned for incrementing at the halfword-address level. The final effective virtual address is formed by the address generator, and if the value of the reference address is greater than 15, the effective virtual address is transformed through the memory map into an actual address. The resultant 22-bit actual (main memory) address, which automatically contains a low-order 0, is then used to access the halfword to be used as the operand for the instruction.

Note that for the real addressing mode, the modifications required for indirect, indexed halfword operation are the same with one exception: reference address 1 and the final effective address are concatenated with three leading zeros (as opposed to being transformed by the memory map).

REAL-EXTENDED ADDRESSING

Real-extended addressing is similar to real addressing in that a direct relationship exists between the effective virtual address of each instruction and the actual address. The function of real-extended addressing is to facilitate operations in a memory system larger than 128K words.

Note: Instructions and indirect addresses that involve real-extended address calculations must themselves reside in the first 128K words of memory (or in the general registers), although they in turn may ultimately access operands in locations beyond the first 128K words of memory.

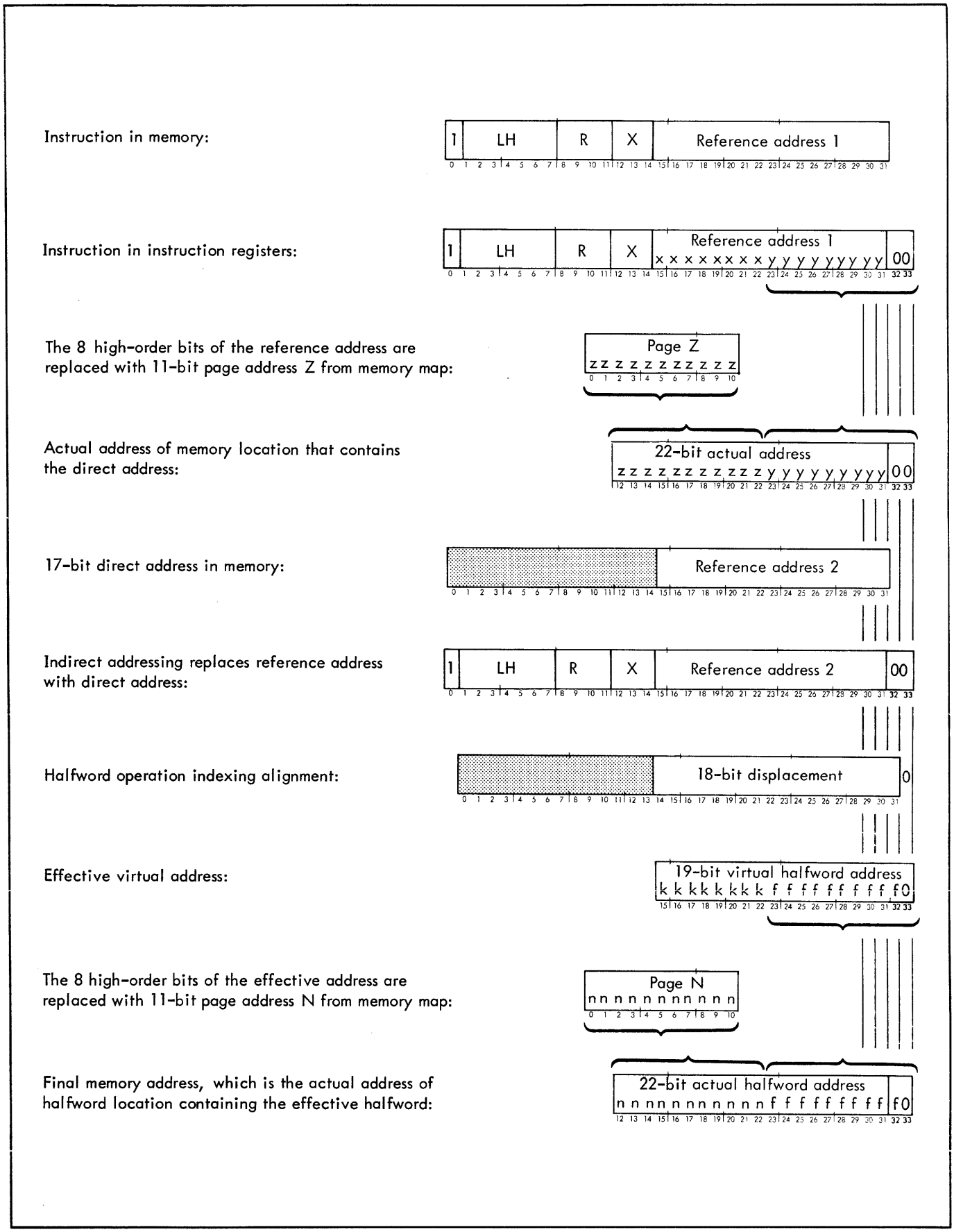


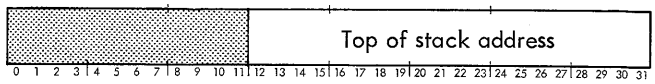
Figure 7. Generation of Actual Addresses Indirect, Virtual Addressing

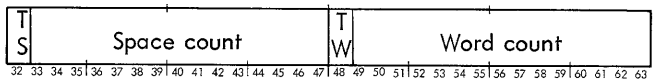
Real-extended addressing is specified when PSWs bit location 9 contains zero and PSWs bit location 61 contains one. In real-extended addressing, the 17-bit reference address in the instruction word is expanded to a 20-bit reference address by the appendage of 3 bit positions to the left of the reference address (see Figure 8). If indexing or indirect addressing are not specified in the instruction, these 3 bit positions contain zeros. Otherwise, address calculations are performed in this manner: If indexing is specified (X field in the instruction contains a value other than zero), the contents of the specified index register are properly aligned with respect to the 17-bit reference address according to the general alignment rules. Arithmetic on the aligned quantities then takes place using the full 32-bit contents of the index register. The final result is truncated 3 bits to the left of the original 17-bit reference address, these 3 bits having been acquired from the index register plus any carry resulting from the addition of the 17-bit reference address with the index register contents.

If the instruction specifies indirect addressing (bit position 0 contains one), the 17-bit reference address is used to access an indirect word in memory. The low-order 20 bits of the indirect word then replace the 17-bit reference address from the instruction. If indexing is also specified, the appropriate alignment of the 32-bit contents of the index register is then made and the addition operation performed. The result is truncated to the left of the 20-bit operand obtained from the indirect address word.

In real-extended addressing, 20-bit address calculations actually encompass 22-, 21-, 20-, and 19-bit calculations, respectively, for byte, halfword, word, and doubleword alignments (see Figures 8 and 9).

The stack pointer doubleword for push-down instructions contains a 20-bit word address for the top of stack address field, as shown in the following format:





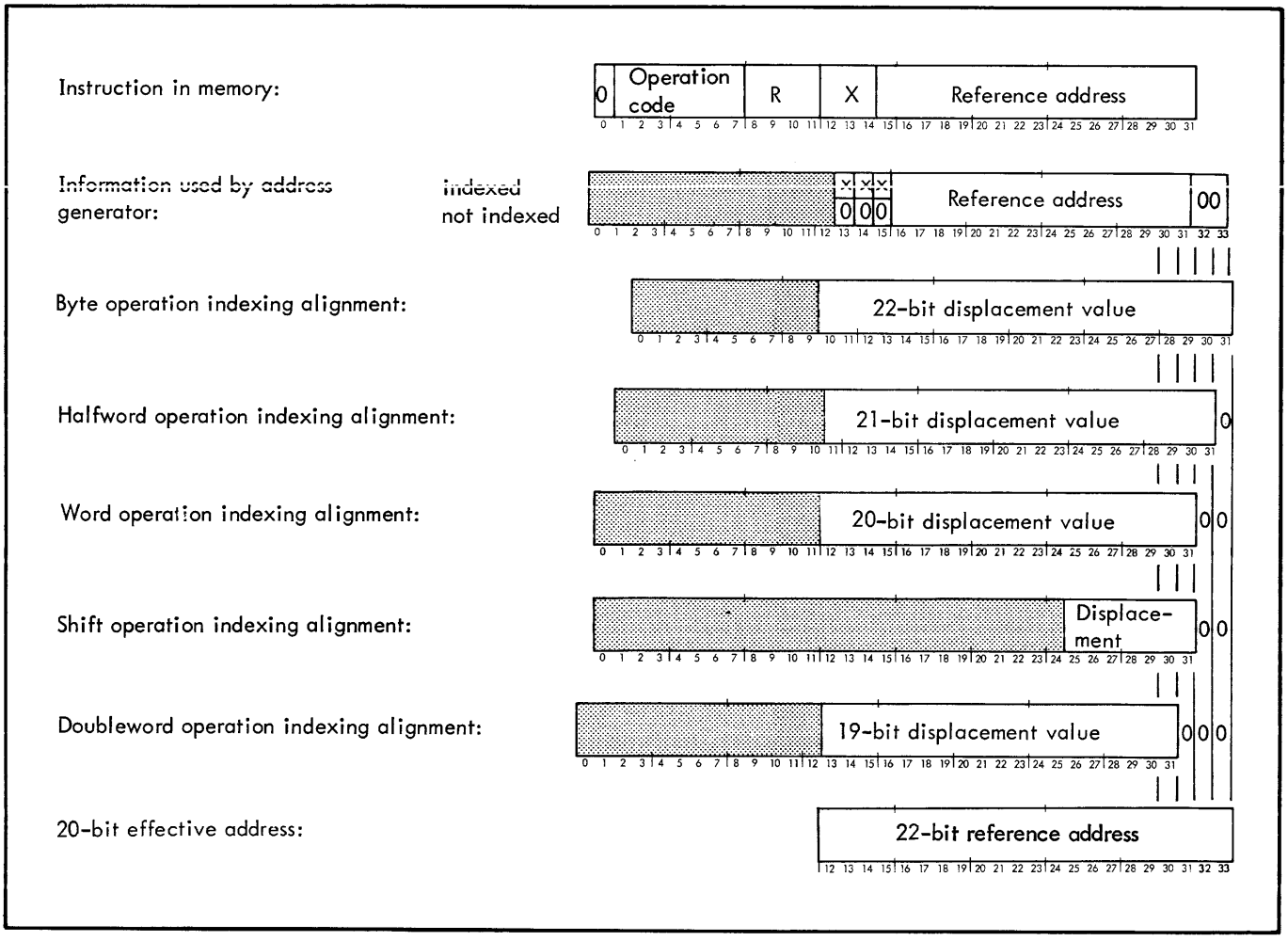


Figure 8. Index Displacement Alignment (Real-Extended Addressing)

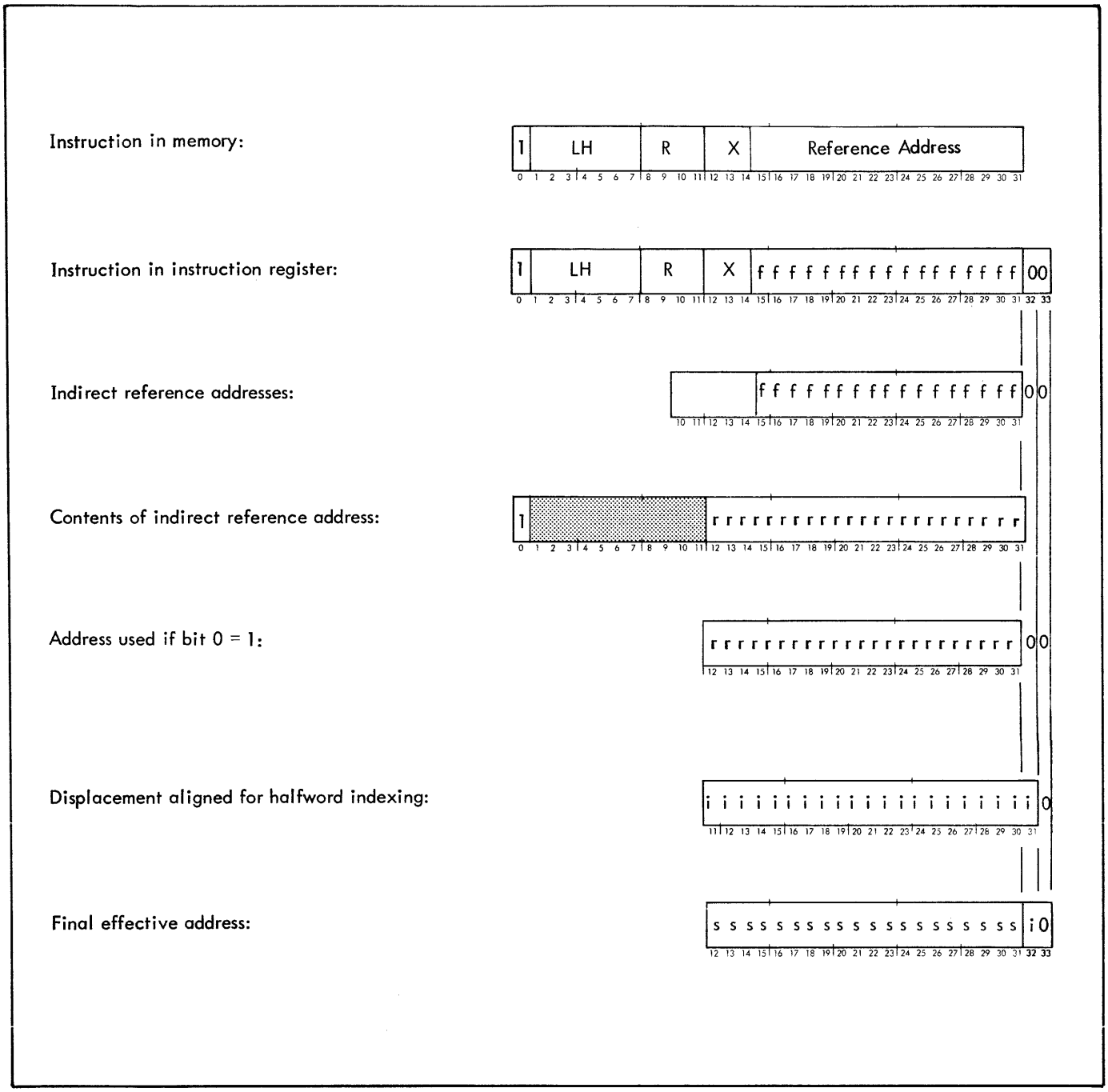
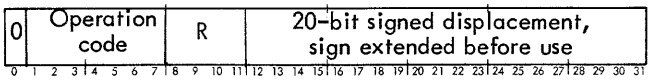


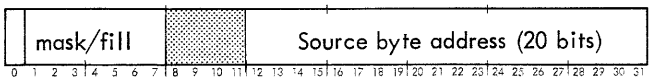
Figure 9. Generation of Effective Virtual Address (Indirect Real-Extended Addressing)

These are the register formats for byte-string instructions:

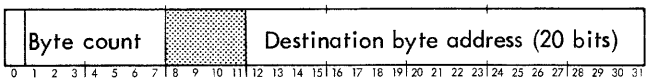
Byte-string instruction:



Register R:



Register Rul:



During real-extended addressing memory write protection is invoked.

Table 1 summarizes the addressing characteristics.

Table 1. Basic Processor Operating Modes and Addressing Cases

PSW BIT			Mode and Addressing Characteristics
MS	MM	MA	
0	0	0	Master mode, unmapped, 17-bit calculations, real addressing (128K words, maximum).
1	0	0	Slave mode, unmapped, 17-bit calculations, real addressing (128K words, maximum).
0	0	1	Master mode, unmapped, 20-bit calculations, real-extended addressing, 17-bit instruction reference address (instructions and indirect words in first 128K words only), indexed and indirect addresses are 20 bits.
1	0	1	Slave mode, unmapped, 20-bit calculations, real-extended addressing, 17-bit instruction reference address (instructions and indirect words in first 128K words only), indexed and indirect addresses are 20 bits.
0	1	0	Master mode, mapped, 17-bit calculations, virtual addressing (128K words, maximum), map to 1M words, real (Sigma 6/7 map to first 128K words by virtue of loading map with three high-order zeros for all pages).
1	1	-	Slave mode, mapped, 17-bit calculations, virtual addressing (128K words, maximum), map to 1M words, real (Sigma 6/7 map to first 128K words by virtue of loading map with three high-order zeros for all pages).
0	1	1	Master-protected mode, mapped, 17-bit calculations, virtual addressing (128K words, maximum), map to 1M words, real (access protection invoked).

INTERRUPT AND TRAP ENTRY ADDRESSING

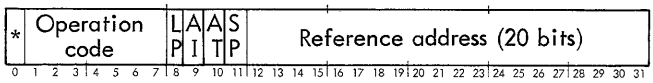
An instruction residing in an interrupt location (see "Centralized Interrupt System" later in this chapter) and executed as the direct result of an interrupt sequence is defined as an interrupt instruction. Both conditions must be true simultaneously. Thus an instruction in an interrupt location is not an interrupt instruction if it is executed as the result of a program branch to the interrupt location under normal program control. The only valid interrupt instructions are XPSD, PSS, MTW, MTH, and MTB.

Similarly, a trap instruction (see "Trap System", later in this chapter) is defined as an instruction in a trap location executed as a direct result of a trap condition. The only valid trap instructions are XPSD and PSS.

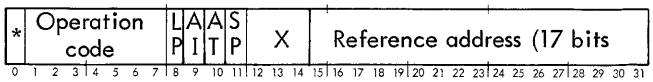
XPSD Address Calculations. Address calculations associated with XPSD instructions deviate from the standard forms. Two basic formats are used in XPSD instructions, depending on whether subjective or objective addressing is being used.

Bit 10 of the XPSD instruction is the addressing type (AT) designator. In the circumstances described below, it designates whether the reference address in the XPSD instruction is to be considered unconditionally as a 20-bit real address or whether the current mode of addressing calculations is to be applied to it.

Format 1:



Format 2:



Format 1 is used in these circumstances:

1. Bit position 10 (AT) of the XPSD contains zero. In this format the reference address is a 20-bit actual address (i.e., no mapping). Note that this is true regardless of whether the instruction is in a trap, interrupt, or normal location and independent of the mode (mapped, unmapped, real-extended) of the current PSWs. If indirect addressing is specified, the indirect word contains a 20-bit address with exactly the same properties.

2. Bit position 10 (AT) of the XPSD contains one, the instruction is in a trap or interrupt location, the instruction is being executed as the result of a trap or interrupt, and the current mode of the PSWs is not real-extended. In this format, the reference address is a 20-bit actual

address if PSWs bit 9 is zero (no map), or a 20-bit virtual address if PSWs bit 9 is one (map). If indirect addressing is specified, the indirect word contains a 20-bit address with exactly the same properties.

Format 2 is used in all other circumstances, namely:

1. Bit position 10 (AT) contains a one, and

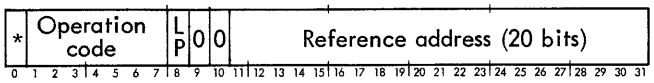
   a. The XPSD is not being executed as the result of a trap or interrupt, or

   b. It is in a trap or interrupt location, is being executed as the result of a trap or interrupt, but the current mode of the PSWs is real-extended.

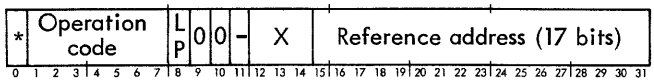
In these cases, all of the normal rules of address calculations hold, i.e., indirect, index, and map.

PSS Address Calculations. PUSH STATUS (PSS) address calculations are similar to but simpler than those for the XPSD instruction. Two basic formats are used:

Format 1:



Format 2:



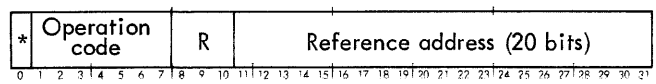
Format 1 is used when the PSS is executed in an interrupt or trap location as a result of an interrupt or trap sequence. No indexing is possible because its designator field is preempted by the reference address. Indirect addressing is permitted with the same constraint against indexing; the indirect address word contains a 20-bit real address with precisely the same properties as the reference address. In the case of a trap instruction, the 20-bit reference address can be either a real address or a virtual address according to the value in PSWs bit position 9.

Format 2 is used when the PSS instruction is executed in the course of normal program execution. Addressing in this case is completely standard, including indexing and indirect addressing.

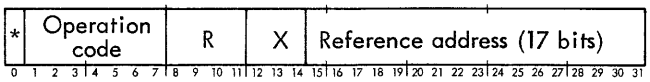
During the execution of the PSS instruction the interrupt stack pointer is accessed from real memory locations 0 and 1. The interrupt stack address therein is a real 20-bit address with no indexing or mapping used.

MTW, MTH, and MTB Address Calculations. Two basic formats are used in modify and test instructions:

Format 1:



Format 2:



Format 1 is used when the modify and test instruction is executed in an interrupt or trap location as a result of an interrupt or trap sequence. When used as an interrupt instruction, the MTW, MTH, or MTB instruction uses the 20-bit reference address as a real address (except counter 4), without indexing or mapping. Interrupt Counter 4 uses the map if mapping is called for. Access protect and write lock violations are not active.

When used as a trap instruction, the MTW, MTH, or MTB instruction uses the 20-bit address without indexing; if the PSWs specify mapping, however, the map is used, with bits 12-14 of the address ignored.

Format 2 is used when the modify and test instruction is executed in the normal course of program execution. Addressing in this case is completely standard, including indexing and indirect addressing.

RD and WD Address Calculations. The final output address for a READ DIRECT (RD) or a WRITE DIRECT (WD) instruction is the low-order 16 bits of the effective virtual address. If indexing is specified in the instruction, the low-order 17 bits of the instruction are modified by the indexing operation, and the resultant 17-bit address is truncated to 16 bits and transmitted as the final address. No mapping takes place.

If indirect addressing is specified in the instruction, the indirect address word is generated in the standard manner according to the mode bits in the PSWs. Thus mapping will occur if it is specified in the PSWs. If indexing is also specified, the indirect address in the indirect word is modified by the indexing operation and the resultant address is truncated to 16 bits and transmitted as the final address.

MEMORY ADDRESS CONTROL

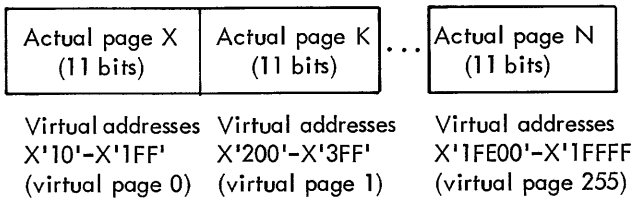
Two methods of program control of main memory are the memory map and the memory locks. The memory map provides for dynamic relocation of programs and for access protection through inhibitions imposed on slave or master-protected mode programs. Access protection violations in either mode are trapped to location X'40'. The memory locks provide memory write protection for all modes of programs throughout all real memory. The memory locks apply to input/output operations as well as basic processor operations. This protection is effective at the page level, is for real addresses, and is operative in addition to the protection provided virtual addresses at the page level. Memory protection violations in any mode are trapped to location X'40'.

Note: A WD instruction used to write into main memory locations 0 through 31 is not subject to write protection.

MEMORY MAPPING AND ACCESS PROTECTION

The memory map is physically an array of 256 11-bit registers. The array resides in the Memory Interface (MI) of the processor cluster containing the basic processor. Each register has an 8-bit address (that corresponds to an 8-bit virtual page address) and contains an 11-bit actual page address for a specific 512-word page of memory. Mapping always transforms a 17-bit virtual address into a 20-bit real address.

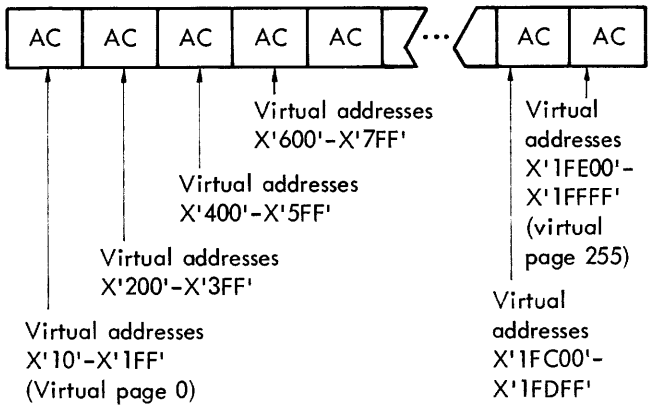
The actual page addresses are assigned to pages of virtual addresses in this manner:



Just prior to a memory reference, the most significant 8 bits of a 17-bit virtual address are used as the address of an element of the map array. The 11 bits contained within that element are then used in conjunction with the low-order 9 bits of the 17-bit virtual address to produce a 20-bit actual address.

Sigma 6/7 compatible mapping is accomplished by loading the map with 8-bit address elements (instead of 11-bit address elements) via the MOVE TO MEMORY CONTROL (MMC) instruction. The 8 bits are stored in the low-order 8 bits of each map element and the 3 high-order bit positions are reset to zero. Thus the map will always relocate to the same address in the first 128K words of real memory and be compatible for Sigma 6/7 programs.

Associated with the memory map feature is another array of 256 2-bit registers, also located in the Memory Interface. Each register contains a 2-bit access control code for a specific 512-word page of virtual addresses. The access-protection code indicates the allowed use or availability of the corresponding page of virtual memory. Access protection applies to all pages of the virtual address space of the active program, and is only active when the memory map is invoked.



The memory page address and access-control codes can be changed only by use of the privileged MMC instruction (see Chapter 3, "Control Instructions).

Access protection is in effect whenever the memory map is in effect (PSWs 9 = 1) and the basic processor is operating in the slave mode (PSWs 8 = 1) or in the master-protected mode (PSWs 61 = 1). Access protection is not in effect when the basic processor is operating in the master mode.

When the memory map is in effect, all memory references used by the program (including instruction addresses) whether direct, indirect, or indexed, are referred to as virtual addresses. Virtual addresses in the range 0 through 15 are not used to address main memory; instead the 4 low-order bits of the virtual address comprise a general register address. If, however, an instruction produces a virtual address greater than 15, the 8 high-order bits of the virtual address are used to obtain the appropriate 11-bit actual memory page address and 2-bit access control codes. For example, if the 8 high-order bits of the virtual address are 0000 0000, the first page address code and the first access control code are used; if the 8 high-order bits of the virtual address are 0000 0001, the second page address code and the second access control code are used, etc., through the 256th page address and access control codes. Thus each 512-word page of virtual addresses is associated with its own memory page address and access control codes.

When the memory map is accessed during a slave mode or master-protected mode program, the basic processor determines whether there are any inhibitions to using the virtual address.

These are the four types of access protection codes:

- 00 A slave mode or master-protected mode program can write into, read from, or access instructions from this page of virtual address.
- 01 A slave mode or master-protected mode program cannot write into this page of virtual addresses; it can, however, read from or access instructions from this page of virtual addresses.
- 10 A slave mode or master-protected mode program cannot write into or access instructions from this page of virtual addresses; it can, however, read from this page of virtual addresses.
- 11 A slave mode or master-protected mode program is denied any access to this page of virtual addresses.

If the instruction being executed by the slave or master-protected program fails the foregoing test, the instruction is aborted and the basic processor traps to location X'40', the "non-allowed operation" trap (see "Trap System", later in this chapter).

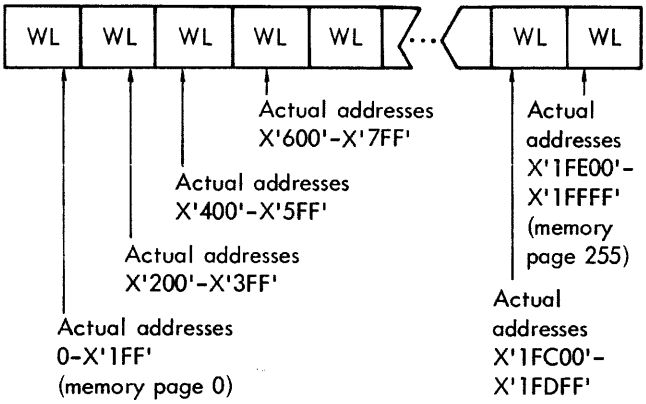
Contrarily, if the instruction being executed by the slave mode or master-protected mode program passes this test (or if the basic processor is operating in the master mode), the

11-bit page address in the accessed element of the memory map array replaces the 8 high-order bits of the virtual address to produce the actual address of the main memory location to be used by the instruction (20-bit word address that is automatically adjusted as required for doubleword, halfword, or byte operation). See Figure 7.

Note: If the 11-bit page address in the accessed element of the memory map is all zeros, and an actual address is produced that corresponds to a word address in the range 0 through 15, when the 11-bit page address is combined with the 9 low-order bits of the virtual address, the corresponding general register in the current register block is not accessed. In this one particular instance a word address in the range 0 through 15 corresponds to an actual main memory location rather than a general register.

REAL MEMORY WRITE LOCKS

Additional memory protection, independent of the access protection, is provided by a write lock and key technique. A 4-bit write protect lock (WL) is provided for each 512-word page of actual memory. Thus, for the maximum 1M-word real memory there would be 2048 4-bit write locks. Write locks are assigned to pages of actual addresses as follows:



The write protect locks can be changed only by executing the privileged instruction MOVE TO MEMORY CONTROL (see Chapter 3, "Control Instructions").

The write key (a 4-bit field in the PSWs for any operating program, or in the command doubleword for I/O operations) works in conjunction with the write lock to determine whether any program (slave, master-protected, or master mode) can write into a specific page of main memory locations. The write key and lock control access for writing according to these rules:

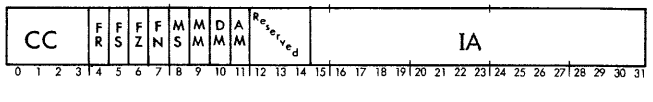
1. A lock value of 0000 means that the corresponding memory page is unlocked; write access to that page is permitted independent of the key value.
2. A key value of 0000 is a "skeleton" key that will open any lock; thus write access to any memory page is permitted independent of its lock value.
3. A lock value other than 0000 for a memory page permits write access to that page only if the key value (other than 0000) is identical to the lock value.

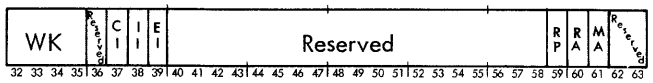
Thus a program can write into a given memory page if the lock value is 0000, if the key value is 0000, or if the key value matches the lock value.

Note: The memory access protection feature operates during virtual addressing modes and on virtual addresses, whereas the memory write protection feature always operates on actual memory addresses. Thus, if the memory access protection feature is invoked (that is, if the basic processor is operating in the slave mode or the master-protected mode and is using the memory map), the access protection codes are examined when the virtual address is converted into an actual address. Then the lock and key are examined to determine whether the program (master, master-protected, or slave mode) is allowed to alter the contents of the main memory location corresponding to the final actual address. If an instruction attempts to write into a write-protected memory page, the basic processor aborts the instruction, and traps to location X'40', the "nonallowed operation" trap (see "Trap System", later in this chapter). If an I/O procedure attempts to write into a write-protected memory page, the write lock violation bit in the IOP status byte is set, and can be tested by the AIO, TIO, and TDV instructions.

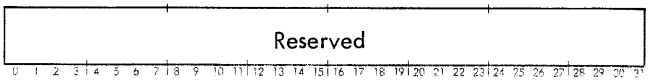
PROGRAM STATUS WORDS

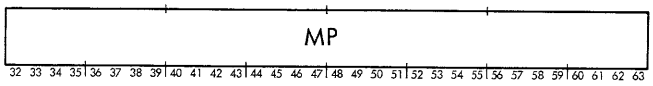
The critical control conditions of the basic processor are defined within 64 bits of information collectively referred to as the program status words (PSWs). The current PSWs may be considered as one 64-bit internal basic processor register, although they actually exist as a collection of separate registers and flip-flops (see Figure 2 appearing earlier in this chapter). When stored in memory, the PSWs have the following format:





They may be optionally followed by an additional two words with the following format:





Designation	Function
CC	Condition code. This generalized 4-bit code indicates the nature of the results of an instruction. The significance of the condition code bits depends upon the particular instruction just executed. After an instruction is executed, the BRANCH ON CONDITIONS SET (BCS) and BRANCH ON CONDITIONS RESET (BCR) instructions can be used singly or in combination to test for a particular condition code setting. (These instructions are described in Chapter 3, "Execute/Branch Instructions"). In some operations only a portion of the condition code is involved; thus, the term CC1 refers to the first bit of the condition code, CC2 to the second bit, and CC3 and CC4, respectively, to the third and fourth bits. Any program can change the current value of the condition code by executing either the LOAD CONDITIONS AND FLOATING CONTROL IMMEDIATE (LCFI) or the LOAD CONDITIONS AND FLOATING CONTROL (LCF) instruction. Any program can store the current condition code by executing the STORE CONDITIONS AND FLOATING CONTROL (STCF) instruction. These instructions are described in Chapter 3, "Load/Store Instructions".
FR	Floating round mode control (see FN below).
FS	Floating significance mode control (see FN below).
FZ	Floating zero mode control (see FN below).
FN	Floating normalize mode control. The four floating-point mode control bits (FR, FS, FZ, and FN) control the operation of the basic processor with respect to invoking the round-off mode of floating-point calculations, checking floating-point significance, generating zero results, and normalizing the results of floating-point additions and subtractions, respectively. (The floating-point mode controls are described in Chapter 3, "Floating-Point Instructions".) Any program can change the state of the current floating-point mode controls by executing either the LCFI or the LCF instruction. Any program can store the current state of the current floating-point mode controls by executing the STCF instruction.
MS	Master/slave mode control. The basic processor is in the master mode when this bit and the mode altered bit (bit 61) both contain zero; it is in the slave mode when this bit contains one. (See MS for a description of master-protected mode.) A master mode or master-protected mode program can change this mode control bit by executing the LOAD PROGRAM STATUS WORDS (LPSD), EXCHANGE PROGRAM STATUS WORDS (XPSD), PUSH STATUS (PSS), or PULL STATUS (PLS) instruction. These privileged instructions are described in Chapter 3, "Control Instructions".
MM	Memory map control. The memory map is in effect when this bit position contains a one. A master mode or master-protected mode program can change the memory map control by executing an LPSD, XPSD, PSS, or PLS instruction.
DM	Decimal mask. The decimal arithmetic trap (see "Trap System", later in this chapter) is permitted to occur when this bit position contains a one. The conditions that cause a decimal arithmetic trap are described in Chapter 3, "Decimal Instructions". The decimal trap mask can be changed by a master mode or master-protected mode program executing the LPSD, XPSD, PSS, or PLS instruction.
AM	Arithmetic mask. The fixed-point arithmetic overflow trap is permitted to occur when this bit contains one. The instructions that can cause fixed-point overflow are described in the section "Trap System", later in this chapter. The arithmetic trap mask can be changed by a master mode or master-protected mode program executing an LPSD, XPSD, PSS, or PLS instruction.
IA	Instruction address. This 17-bit field contains the virtual address of the next instruction to be executed.
WK	Write key. This field contains the 4-bit key used in conjunction with a write lock in the memory write protection feature. A master mode or master-protected mode program can change the value of the write key by executing an LPSD, XPSD, PSS, or PLS instruction.
CI	Counter interrupt group inhibit (see EI, below).
II	Input/output interrupt group inhibit (see EI, below).

Designation	Function
EI	External interrupt group inhibit. The three interrupt group inhibit bits (CI, II, and EI) determine whether certain interrupts are allowed to occur. The function of these group interrupt inhibits are described in "Centralized Interrupt System", later in this chapter. A master mode or master-protected mode program can change the group interrupt inhibits by executing an LPSD, XPSD, PSS, PLS, or WRITE DIRECT (WD) instruction. These privileged instructions are described in Chapter 3, "Control Instructions".
RP	Register pointer. This 2-bit field selects one of the 4 possible blocks of general-purpose registers as the current register block. A master or master-protected mode program can change the register pointer by executing LPSD, XPSD, PSS, PLS, or the LOAD REGISTER POINTER (LRP) instruction. LRP is described in Chapter 3, under "Control Instructions".
RA	Register altered bit. When a trap occurs, this bit is set to one when any general register or location in memory has been altered in the execution or partial execution of the instruction that caused the trap.
MA	Mode altered. This bit is used to invoke both the master-protected mode of operation and the real-extended addressing mode). Table 1 details the function of the setting of this bit in conjunction with the setting of the MS (bit 8) and MM (bit 9) fields. The bits are set by an LPSD, XPSD, PSS, or PLS instruction.
MP	Memory protection violation address. If the XPSD instruction is being executed in a trap routine as a result of a memory protection violation and the SP bit in the XPSD is a one, the effective virtual address causing the violation is stored in the fourth word. This storage may be invoked so that memory protection violations can be recorded.

CENTRALIZED INTERRUPTS

The system includes a single, centralized interrupt feature. All interrupts are terminated in the System Control Processor. The System Control Processor is described earlier and also in Chapters 5 and 6.

When a condition that will result in an interrupt is sensed, a signal is sent to the corresponding interrupt level. If that level is "armed", it advances to the waiting state. When all the conditions for acknowledging the interrupt have been achieved, the basic processor stops executing the current program and executes the instruction in the corresponding interrupt location. After the basic processor has successfully accessed the interrupt instruction, it advances the interrupt level to the active state. The basic processor may actually execute many program instructions between the time that the interrupt-requesting condition is sensed and the time that the actual interrupt acknowledgment occurs. After the interrupt is completely processed, the basic processor returns to the interrupted program and resumes its execution.

STATES OF AN INTERRUPT LEVEL

An interrupt level is mechanized by means of three flip-flops. Two flip-flops are used to define four mutually exclusive states: disarmed, armed, waiting, and active. The third flip-flop provides the disabled/enabled function and is independent of the defined state. The various states and the conditions of interrupt levels are described in the following paragraphs. Figure 10 conceptually illustrates the operational state changes of a typical interrupt level.

DISARMED

When an interrupt level is in the disarmed state, no signal is admitted to that interrupt level; that is, the level neither accepts nor remembers an interrupt event, nor is any program interrupt caused by it at any time.

Although an interrupt level can change from any state to the disarmed state, only a special form of the WRITE DIRECT instruction (WD) can cause a disarmed level to change to another state. The WD instruction is described in Chapter 3, "Control Instructions".

ARMED

When an interrupt level is in the armed state, it can accept and remember an interrupt signal. The receipt of such a signal advances the interrupt level to the waiting state where it remains until it is allowed to advance to the active state. A special form of the WD instruction can cause an armed level to be advanced directly to the active state.

A level can change from any state to the armed state.

WAITING

For an interrupt level to be in the waiting state, that level must have been previously armed and received an interrupt signal. The signal may have been generated externally, internally, or have resulted from a WD operation. Any signals received by an interrupt level already in the waiting state are ignored.

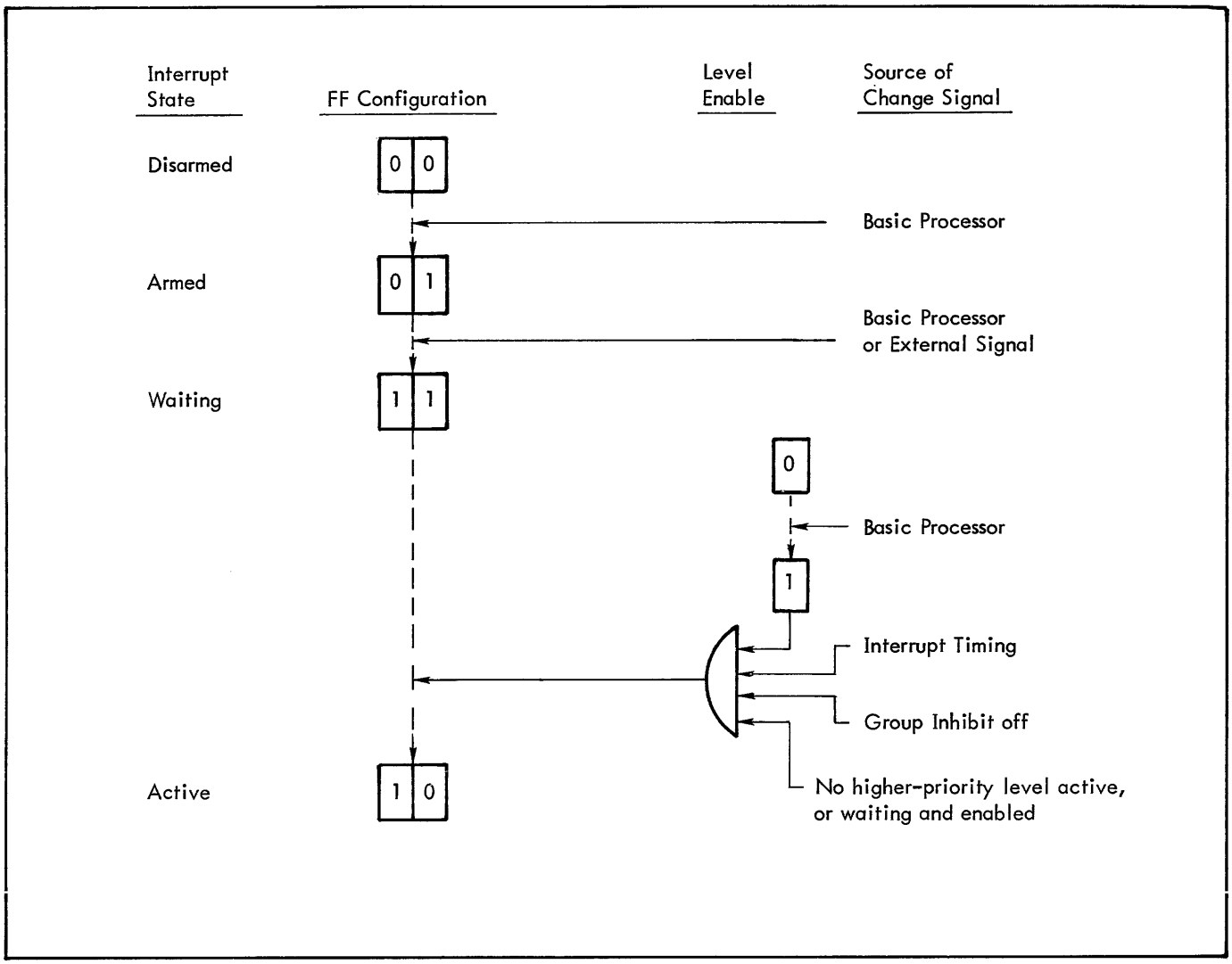


Figure 10. Operational States of an Interrupt Level

When an interrupt level is in the waiting state, the following conditions must all exist simultaneously before the level advances to the active state:

1. The level must be enabled (i.e., its enable/disable flip-flop must be set to one).

2. The group inhibit (CI, II, or EI, if applicable) must be zero.

3. No higher-priority interrupt level is in the active state, or is in the waiting state, enabled, and not inhibited.

4. The basic processor must be at an interruptible point in the execution of a program.

Note that one or more interrupt levels of higher priority can also be in the waiting state if they are disabled, inhibited, or both disabled and inhibited.

Generally, if the enable/disable flip-flop is off (level is disabled), the interrupt level can undergo all state changes except that of moving from the waiting to the active state (see exception case, below). Furthermore, if the interrupt level is disabled, it is completely removed from the chain that determines the priority of access to the basic processor. Thus a disabled interrupt level in the waiting state does not prevent an enabled, waiting interrupt level of lower priority from moving to the active state.

Note this exception to the foregoing description: Although generally no interrupt level can move from the waiting state to the active state unless it is enabled, a special form of the WD instruction can move a waiting level to the active state whether or not the level is enabled.

ACTIVE

After the basic processor has successfully accessed the interrupt instruction, then the interrupting level advances to the active state. When all the conditions for acknowledgment have been achieved, the interrupt level causes the

basic processor to execute the contents of the assigned interrupt location as the next instruction. (Interrupt locations are defined in "Physical Organization", later in this chapter.) The instruction address portion of the program status words (PSWs) remains unchanged until the instruction in the interrupt location is executed.

The instruction in the interrupt location must be one of the following: XPSD, PSS, MTB, MTH, or MTW. If the execution of any other instruction in an interrupt location is attempted as the result of an interrupt level advancing to the conditions for acknowledgment, an instruction exception trap occurs.

The use of the privileged instruction XPSD or PSS in an interrupt location permits an interrupt-servicing routine to save the entire current machine environment. If working registers are needed by the routine and additional register blocks are available, the contents of the current register block can be saved automatically with no time loss. This is accomplished by changing the value of the register pointer (using the LOAD REGISTER POINTER instruction), which results in the assignment of a new block of 24 registers to the routine. The instruction LOAD REGISTER POINTER (LRP) is described in Chapter 3, "Control Instructions".

An interrupt level remains in the active state until it is cleared (removed from the active state and returned to the disarmed or armed state) by the execution of the LPSD, PLS, or WD instruction. An interrupt-servicing routine can itself be interrupted (whenever a higher priority interrupt level meets all the conditions for becoming active) and then continued (after the higher priority interrupt is cleared). However, an interrupt-servicing routine cannot be interrupted by an interrupt of the same or lower priority as long as the higher priority interrupt level remains in the active state. Any signals received by an interrupt level in the active state are ignored. Normally, the interrupt-servicing routine clears its interrupt level and transfers program control back to the point of interrupt by means of an LPSD instruction with the same effective address as the XPSD instruction in the interrupt location.

DIALOGUE BETWEEN THE BASIC PROCESSOR AND THE INTERRUPT SYSTEM DURING AN INTERUPT-ENTERING SEQUENCE

When an interrupt level is ready to be moved to the active state, a dialogue takes place between the interrupt system and the basic processor. This dialogue takes place over the processor bus and involves the Processor Interface (PI) associated with the processor cluster of which the basic processor is a member. When the processor bus becomes available and the basic processor is at an interruptible point, the interrupt system transmits the interrupt address to the basic processor. It initiates its interrupt actions (i.e., executes the instruction in the interrupt location and services the interrupt at the appropriate time to avoid race conditions, and communicates with the interrupt system with an indication to move the level to the active state. This latter transmission is delayed until the new inhibit states of the basic processor are known; these states are transmitted to the interrupt system so the latter can record the new basic processor status.

DIALOGUE DURING AN INTERRUPT-EXITING SEQUENCE

When the basic processor exits an interrupt-servicing routine, it must notify the interrupt system to move the interrupt level associated with that routine from the active state to either the armed or disarmed state. To do this it must gain access to the processor bus and the interrupt system, either of which may be busy at the time access is requested. When communication with the interrupt system is established, the basic processor transmits information for setting the level state to armed or disarmed, and new inhibit states it has assumed as a result of the exit operation.

PHYSICAL ORGANIZATION

Up to 62 interrupt levels are available, each with a unique location (see Table 2) assigned in the System Control Processor, and with a unique priority. The basic processor can selectively arm, enable, or arm and enable any interrupt level. The basic processor can also "trigger" any interrupt level (supply a signal at the same physical point where the signal from the external source would enter the interrupt level). The triggering of an interrupt permits testing special systems programs before the special systems equipment is available. The basic processor also permits an interrupt-servicing routine to defer a portion of the processing associated with an interrupt level by processing the urgent portion of an interrupt-servicing routine, triggering a lower priority level (for a routine that handles the less urgent part), then clearing the high-priority interrupt level so that other interrupts can occur before the deferred interrupt response is processed.

INTERRUPT GROUPS

Interrupt levels are organized in standard group configurations that are connected in a predetermined and fixed priority chain (see Table 2 and Figure 11). The priority of each level within a group is fixed; the first level has the highest priority and the last level has the lowest.

INTERNAL INTERRUPTS

Standard internal interrupts are provided with the system and include all group D levels (internal override, counter-equals-zero, and I/O).

Table 2. Interrupt Locations

Group	Address		Function	PSWs Inhibit	DIO Address	
	Dec	Hex			Group	Register Bit
Internal Override (optional)	82	52	Counter 1 count pulse	none	0	16
	83	53	Counter 2 count pulse			17
	84	54	Counter 3 count pulse			18
	85	55	Counter 4 count pulse			19
	86	56	Processor fault			20
	87	57	Memory Fault			21
External Override	112	70	External group 3 (first 12 levels)	EI	3	16
	113	71				17
	114	72				18
	115	73				19
	116	74				20
	117	75				21
	118	76				22
	119	77				23
	120	78				24
	121	79				25
	122	7A				26
	123	7B				27
Counter-Equals-Zero	88	58	Counter 1 zero	CI	0	22
	89	59	Counter 2 zero			23
	90	5A	Counter 3 zero			24
	91	5B	Counter 4 zero			25
I/O	92	5C	Input/Output	II	0	26
	93	5D	Control panel			27
	94	5E	Reserved			28
	95	5F	Reserved			29
External Group 2 (optional)	96	60	External group 2 (first 12 levels)	EI	2	16
	.	.				.
	.	.				.
	107	6B				27
External Group 4 (optional)	128	80	External group 4 (first 12 levels)	EI	4	16
	.	.				.
	.	.				.
	139	8B				27
External Group 5 (optional)	144	90	External group 5 (first 12 levels)	EI	5	16
	.	.				.
	.	.				.
	155	9B				27

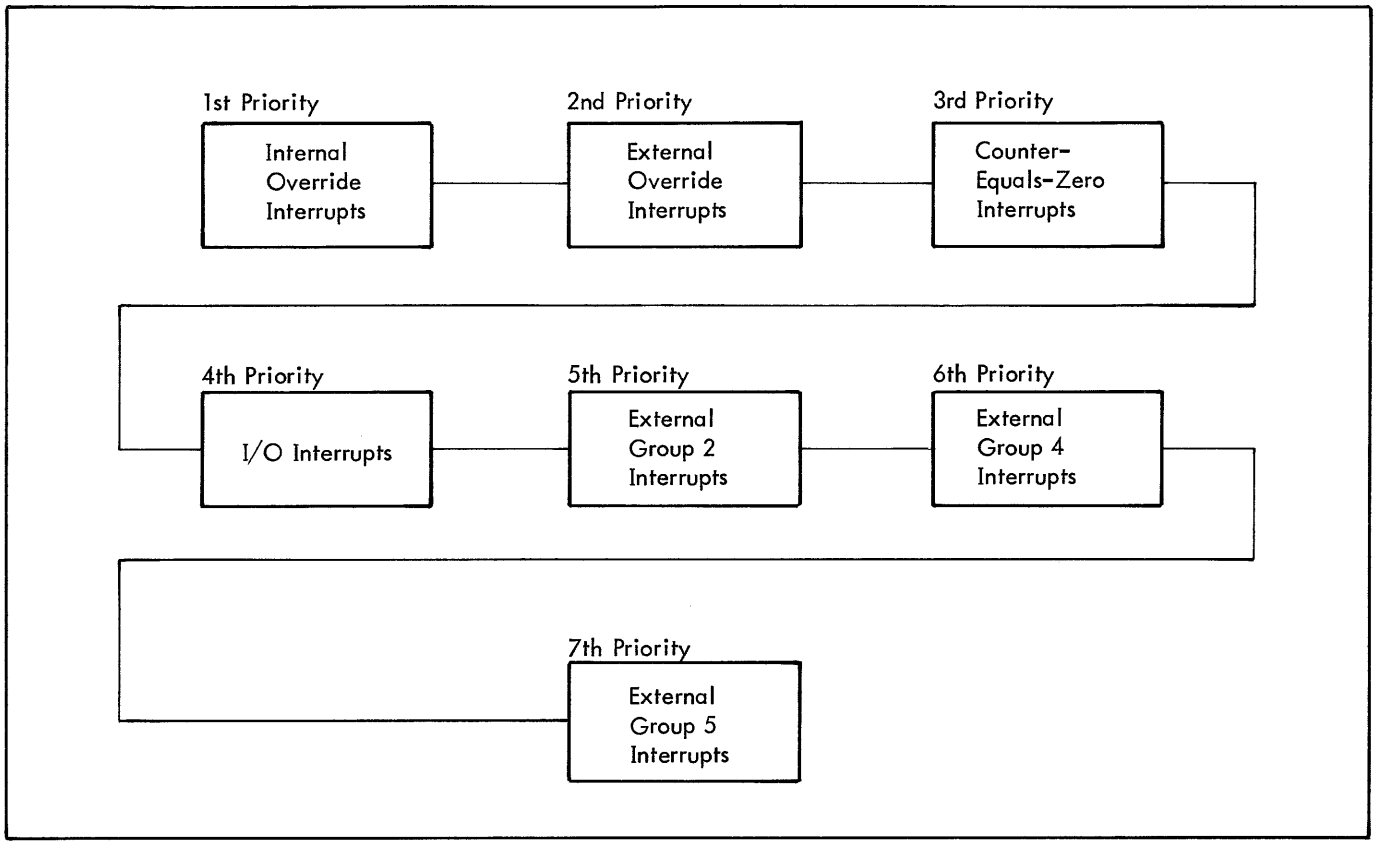


Figure 11. Interrupt Priority Chain

Internal Override Group (Locations X'52' through X'57'). The six interrupt levels of this group always have the highest priority in the system. The four count-pulse interrupt levels are triggered by pulses from clock sources. Counter 4 has a constant frequency of 500 Hz. Counters 1, 2, and 3 can be individually set to any of four manually switchable frequencies – the commercial line frequency, 500 Hz, 2000 Hz, or a user-supplied external signal – that may be different for each counter. Each of the count pulse interrupt locations must contain one of the modify and test instructions (MTB, MTH, or MTW), an XPSD, or a PSS instruction. When the modification (of the effective byte, halfword, or word) causes a zero result, the appropriate counter-equals-zero interrupt level (see "Counter-Equals-Zero Group") is triggered.

Note: Count pulse interrupt level 4 is a subjective time counter with the following special attribute: When the instruction in location X'55' is executed as the result of an interrupt, it must be an MTB, MTH, or MTW; otherwise, an instruction exception trap (X'40') will occur.

The internal override group also contains a processor fault and a memory fault interrupt level. Both locations normally contain an XPSD or a PSS instruction. The processor fault interrupt level is triggered by a signal when certain fault conditions are detected. A POLR instruction must be used to reset the fault. The memory fault interrupt level is triggered by a signal that the memory generates when it detects certain fault conditions. An LMS instruction must be used to reset the fault. (See "Trap System" later in this chapter for further information on processor and memory faults.)

Counter-Equals-Zero Group (Locations X'58' through X'5B'). Each interrupt level in the counter-equals-zero group is associated with a corresponding count-pulse interrupt level in the internal override group. When the execution of a modify and test instruction in the count-pulse interrupt location causes a zero result in the effective byte, halfword, or word location, the corresponding counter-equals-zero interrupt level is triggered. The counter-equals-zero interrupt locations normally contain an XPSD or a PSS instruction and they can be inhibited or permitted as a group. If bit 37 (CI) of the current PSW contains a zero, the counter-equals-zero interrupt levels are allowed to interrupt the program being executed. If the CI bit contains a one, the counter-equals-zero interrupt levels are inhibited from being allowed to interrupt the program. These interrupt levels wait until the CI bit is reset to zero and then interrupt the program according to priority.

Input/Output Group (Locations X'5C' through X'5F'). This interrupt group comprises the input/output (I/O) interrupt level, the control panel interrupt level, and two levels reserved for future use. The I/O interrupt level accepts interrupt signals from the I/O system. The I/O interrupt location

is assumed to contain an XPSD or a PSS instruction that transfers program control to a routine for servicing all I/O interrupts. The I/O routine should contain an ACKNOWLEDGE I/O INTERRUPT (AIO) instruction that identifies the source and reason for the interrupt. (The AIO instruction is discussed in Chapter 3 "Input/Output Instructions".)

The control panel interrupt level is activated from the operator's console. This location normally contains an XPSD or a PSS instruction. The operator can thus trigger this interrupt level to initiate a specific routine.

The interrupt levels in the I/O group can be inhibited or permitted by means of bit position 38 (II) of the PSWs. If II is reset to zero, interrupt signals affecting the I/O group interrupt levels are allowed to interrupt the program being executed. If the II bit is set to one, interrupt signals in this group are inhibited from interrupting the program.

EXTERNAL INTERRUPTS

A system can contain 4 optional groups of external interrupt levels. The external override group, group 3, contains the first 12 external interrupt levels. External groups 2, 4, and 5 each contain 12 external interrupt levels. (See Table 2 and Figure 11.) External levels may be triggered by external sources or via WD instructions, while internal levels may be triggered by internal sources or via WD instructions.

All external interrupt levels normally contain XPSD or PSS instructions and can be inhibited or permitted by means of the setting of bit position 39 (EI) of the program status words. If EI contains a zero, external interrupts are allowed to interrupt a program; if EI contains a one, all external interrupts are inhibited from interrupting the program.

NUMBER OF INTERRUPT GROUPS

The 14 internal interrupt levels are standard in every system and all external levels are optional. The addition of the external groups (12 levels per group) raises the number of interrupt levels to a maximum of 62.

CONTROL OF THE INTERRUPT SYSTEM

The system has two points of interrupt control. One point of interrupt control is achieved by means of the interrupt inhibit bits (CI, II, and EI) in the program status words (PSWs). The basic processor is inhibited from interrupting a program if the interrupt inhibit bit for a corresponding class of interrupt levels is set to one, that is, no interrupt level in the inhibited group can advance from the waiting state to the active state, and the entire group is disabled (removed from the interrupt recognition priority chain). Consequently, a waiting, enabled, interrupt level in an inhibited group does not prevent a lower priority, waiting, enabled interrupt level in an uninhibited group from interrupting the program. However, if an interrupt group is inhibited while a level in that group is in the active state, no lower priority interrupt level can advance to the active state.

Note also this special case: When the processor detected fault (PDF) flag is set to 1 (see "Processor Detected Faults", later in this chapter), the processor fault, memory fault, and count pulse interrupts are automatically inhibited.

The second point of interrupt control is at the individual interrupt level. The basic processor can interact with any interrupt level by means of special modes of the RD and WD instructions (described in Chapter 3, "Control Instructions"). For this purpose, the interrupt levels are organized into the following DIO address groups (see last two columns in Table 2):

1. The 14 levels of internal interrupts (internal override group, counter-equals-zero group, and I/O group) are designated as group code 0 in bits 28-31 of the effective address of the RD or WD instruction.

2. The 12 levels of each group of external interrupts are designated as group codes 2, 3, 4, and 5. That is, external group 2 is designated group code 2, external group 3 is designated group code 3, etc.

3. There is no group code 1.

The addressing of an individual interrupt level within its DIO group of 12 or 14 is accomplished by an assigned selection bit within the low-order 16-bit positions of the R register designated in the RD or WD instruction (see last column in Table 2).

The WD instruction can individually arm, disarm, enable, disable, or trigger (move to the active state) any interrupt level. The RD instruction can determine which interrupt levels within a selected DIO group are in the armed or waiting state, waiting or active state, or are enabled.

TIME OF INTERRUPT OCCURRENCES

The basic processor permits an interrupt to occur during the following time intervals (related to the execution cycle of an instruction) provided the SCP basic processor (BP) status indicators are either in the RUN or WAIT condition:

1. Between instructions an interrupt is permitted between the completion of any instruction and the initiation of the next instruction.

2. Between instruction initiations an interrupt is also permitted to occur during the execution of the following multiple-operand instructions:

   MOVE BYTE STRING (MBS)

   COMPARE BYTE STRING (CBS)

   TRANSLATE BYTE STRING (TBS)

TRANSLATE AND TEST BYTE STRING (TTBS)

EDIT BYTE STRING (EBS)

DECIMAL MULTIPLY (DM)

DECIMAL DIVIDE (DD)

MOVE TO MEMORY CONTROL (MMC)

The control and immediate results of these instructions reside in registers and memory; thus, the instruction can be interrupted between the completion of one iteration (operand execution cycle) and that time (during the next iteration) when a memory location or register is modified. If an interrupt occurs during this time, the current iteration is aborted and the instruction address portion of the program status words (PSWs) remains pointing to the interrupted instruction. After the interrupt-servicing routine is completed, the instruction continues from the point at which it was interrupted and does not begin anew.

SINGLE-INSTRUCTION INTERRUPTS

A single-instruction interrupt occurs in this situation: an interrupt level is activated, the current program is interrupted, the single instruction in the interrupt location is executed, the interrupt level is automatically cleared and armed, and the interrupted program continues without being disturbed or delayed (except for the time required to execute the single instruction).

If any of the following instructions is executed in any interrupt location, then the corresponding interrupt is automatically a single-instruction interrupt:

MODIFY AND TEST BYTE (MTB)

MODIFY AND TEST HALFWORD (MTH)

MODIFY AND TEST WORD (MTW)

A modify and test instruction modifies the effective byte, halfword, or word (as described in Chapter 3, "Fixed-Point Arithmetic Instructions") but the current condition code remains unchanged (even if overflow occurs). The effective address of a modify and test instruction in an interrupt location (except counter 4) is always treated as an actual address, regardless of whether the memory map is currently being used. Counter 4 uses the mapped location if mapping is currently invoked (as specified in the PSWs). The execution of a modify and test instruction in an interrupt location, including mapped and unmapped counter 4, is independent of the virtual memory access-protection code and the real memory write lock; thus, a memory protection violation trap cannot occur as the result of overflow caused by executing MTH or MTW in an interrupt location.

The execution of a modify and test instruction in an interrupt location automatically clears and arms the corresponding interrupt level, allowing the interrupted program to continue.

When a modify and test instruction is executed in a count-pulse interrupt location, all of the above conditions apply as well as the following: If the resultant value in the effective location is zero, the corresponding counter-equals-zero interrupt is triggered.

TRAP SYSTEM

A trap is similar to an interrupt in that when a trap condition occurs, program execution automatically branches to a predesignated location. A trap differs from an interrupt in that a trap location must contain an XPSD or PSS instruction. The time of trap occurrence can vary: The instruction in the trap location can be executed immediately (i.e., the current instruction in the program being executed is aborted), or when the current instruction has been partially executed (i.e., in the case of a long byte-string operation), or upon completion of the current instruction. The trap instruction is not held in abeyance by higher priority traps, whereas interrupts possibly may not be processed before an entire sequence of instructions is executed.

TRAP ENTRY SEQUENCE

A trap entry sequence begins when the basic processor detects the trap condition and ends when the new program status words (PSWs) have successfully replaced the old PSWs. Detection of any condition (function) listed in Table 3, which summarizes the trap system, results in a trap to a unique location in memory. When a trap condition occurs, the basic processor sets the trap state. The operation the basic processor is currently performing may or may not be carried to completion, depending on the type of trap and the operation being performed. In any event, the program instruction is terminated with a trap sequence (branch to the appropriate trap location). During this sequence the program counter is not advanced; instead, the XPSD instruction in the trap location is executed. If any interrupt level is ready to move to the active state at the same time an XPSD trap instruction is in process, the interrupt acknowledgment will not occur until the XPSD trap instruction is completed. Should a trap location not contain an XPSD or PSS instruction, a second trap sequence is immediately invoked (see "Instruction Exception Trap" later in this chapter).

TRAP ADDRESSING

Trap addressing is described under "Interrupt and Trap Entry Addressing".

Table 3. Summary of Trap Locations

Locations		Function	PSWs Mask Bit	Time of Occurrence	Trap Condition Code
Dec.	Hex.				
64	40	Nonallowed operation			
		1. Nonexistent instruction	None	At instruction decode.	Set TCC1[†]
		2. Nonexistent memory address	None	Prior to memory access.	Set TCC2
		3. Privileged instruction in slave mode	None	At instruction decode.	Set TCC3
		4. Memory protection violation	None	Prior to memory access.	Set TCC4
		5. Write lock violation	None	Prior to memory access.	Set TCC3, TCC4
65	41	Reserved			
66	42	Push-down stack limit reached	TW, TS (in stack pointer)	At the time of stack limit detection. (The aborted pushdown instruction does not change memory, registers, or the condition code.)	None
67	43	Fixed-point arithmetic overflow	AM	For all instructions except DW and DH, trap occurs after completion of instruction. For DW and DH, instruction is aborted with memory, register, CC1, CC3, and CC4 unchanged.	None
68	44	Floating-point arithmetic fault		At detection.	
		1. Characteristic overflow	None	(The floating-point instruction is aborted without changing any registers. The condition code is set to indicate the reason for the trap.)	None
		2. Divide by zero	None		None
		3. Significance check	FS, FZ, FN		None
69	45	Decimal arithmetic fault	DM	At detection. (The aborted decimal instruction does not change memory, registers, CC3, or CC4.)	None
70	46	Watchdog timer runout	None	At runout. (The PDF[††] flag will be set.)	Set TCC2 if basic processor using processor bus; set TCC3 if basic processor using memory bus; and set TCC4 if basic processor using DIO bus.
71	47	Programmed trap	None	Interruptible point reached upon completion of WD.	None

[†] See "Trap Condition Code", later in this chapter.

[††] See "Processor Detected Faults", later in this chapter.

Table 3. Summary of Trap Locations (cont.)

Locations		Function	PSWs Mask Bit	Time of Occurrence	Trap Condition Code
Dec.	Hex.				
72	48	CALL1	None	At instruction decode.	Equal to R field of CALL instruction.
73	49	CALL2	None	At instruction decode.	Equal to R field of CALL instruction.
74	4A	CALL3	None	At instruction decode.	Equal to R field of CALL instruction.
75	4B	CALL4	None	At instruction decode.	Equal to R field of CALL instruction.
76	4C	Hardware error trap	None	At time of basic processor detection (the PDF[t] flag will be set).	TCC1, 2, 3 = 0 TCC4 = 0 if parity error on general register or internal control register. TCC4 = 1 if other hardware errors.
77	4D	Instruction exception trap	None	(The PDF[t] flag will be set.)	Set TCC3 if MMC configuration illegal; set TCC = X'C' if trap or interrupt sequence with illegal instruction; set TCC = X'F' if trap or interrupt sequence and processor detected fault; set TCC4 if invalid register designation (odd register on AD, SD, FAL, FSL, FML, FDL, TBS, TTBS, EBS, and register 0 on EBS).
78	4E	Reserved			
79	4F	Reserved			
80	50	Power on		Interruptible point.	
81	51	Power off		Interruptible point.	

[t]See "Processor Detected Faults", later in this chapter.

TRAP MASKS

The programmer may mask the four trap conditions described below in the program status words (PSWs) or the stack pointer doubleword, as appropriate; other traps cannot be masked.

1. The push-down stack limit trap is masked within the stack pointer doubleword for each individual stack.
2. The fixed-point overflow trap is masked in bit position 11 (AM) of the PSWs. If this bit position contains a zero, the trap is allowed to occur; if bit 11 contains a zero, the trap is not allowed to occur. AM can be masked by operator intervention, or by execution of the XPSD, PSS, PLS, or LPSD privileged instructions.
3. The floating-point significance check trap is masked by a combination of the floating significance (FS), floating zero (FZ), and floating normalize (FN) mode control bits in the PSWs (see "Floating-Point Arithmetic Fault Trap", later in this chapter). FS, FZ, and FN can be set or cleared by the execution of any of the following instructions:

   LOAD CONDITIONS AND FLOATING CONTROL (LCF)

   LOAD CONDITIONS AND FLOATING CONTROL IMMEDIATE (LCFI)

   EXCHANGE PROGRAM STATUS WORDS (XPSD)

   LOAD PROGRAM STATUS WORDS (LPSD)

   PUSH STATUS (PSS)

   PULL STATUS (PLS)

4. The decimal arithmetic fault trap is masked by bit position 10 (DM) of the PSWs. If DM contains a one, the trap is allowed; if DM contains a zero, the trap is not allowed. DM can be masked by execution of the XPSD, PSS, PLS, or LPSD privileged instruction.

TRAP CONDITION CODE

For the push-down stack limit trap, fixed-point overflow trap, floating-point fault trap, and decimal fault trap, the normal condition code register (CC1-CC4) is loaded with more detailed information about the trap condition just before the trap occurs. These condition codes are saved as part of the old program status words when the XPSD or PSS instruction is executed in response to the trap.

For the nonallowed operation trap, watchdog timer runout trap, hardware error trap, instruction exception trap, and CALL trap, a special register (trap condition codes TCC1-TCC4) is loaded just before the trap occurs. When the XPSD or PSS instruction is executed in response to the trap, this register is added to the new program address if bit 9 (MM) contains a one; TCC1-TCC4 are also logically ORed with the condition code bits (CC1-CC4) of the new PSWs when loading CC1-CC4. See also "Instruction Exception Trap" later in this chapter for more information on the trap condition code.

NONALLOWED OPERATION TRAP

The attempt to perform a nonallowed operation always causes the basic processor to abort the instruction being executed when the nonallowed operation is detected and to immediately execute the XPSD or PSS instruction in trap location X'40'. A nonallowed operation cannot be masked.

NONEXISTENT INSTRUCTION

Any instruction that is not standard is defined as nonexistent. This includes immediate operand instructions that specify indirect addressing (a one in bit 0 of the instruction). If a nonexistent instruction is detected, the basic processor traps to location X'40' when the nonexistent instruction is decoded. No general registers or memory locations are changed; the PSWs point to the instruction trapped. With respect to the condition code and instruction address fields of the program status words, the operation of the XPSD or PSS in location X'40' is as follows:

1. Store the current PSWs. The condition codes stored are those that existed at the end of the last instruction prior to the nonexistent instruction.
2. Store the 16 general registers of the current register block if instruction in trap location is a PSS.
3. Load the new PSWs.
4. Modify the new PSWs.
   a. Set CC1 to one. The other condition code bits remain unchanged from the values loaded from memory.
   b. If bit position 9 (AI) of the XPSD or PSS instruction contains a one, the program counter is incremented by eight. If AI contains a zero, the program counter remains unchanged from the value loaded from memory.

NONEXISTENT MEMORY ADDRESS

Any attempt to access a nonexistent memory address causes a trap to location X'40' at the time of the request for memory service. A nonexistent memory address condition is detected when an actual address is presented to the memory

system. If the basic processor is in the map mode, the program address will already have been modified by the memory map to generate an actual (but nonexistent) address. (See Table 5 for possible changes to registers and memory locations later in this chapter.) The operation of the XPSD or PSS in location X'40' is as follows:

1. Store the current PSWs.
2. Store general registers if PSS.
3. Load the new PSWs.
4. Modify the new PSWs.
    a. Set CC2 to one. The other condition code bits remain unchanged from the values loaded from memory.
    b. If bit position 9 (AI) of the XPSD or PSS instruction contains a one, the program counter is incremented by four. If AI contains a zero, the program counter remains unchanged from the value loaded from memory.

PRIVILEGED INSTRUCTION IN SLAVE MODE

An attempt to execute a privileged instruction while the basic processor is in the slave mode causes a trap to location X'40' before the privileged operation is performed. No general registers or memory locations are changed, and the PSWs point to the instruction trapped. The operation of the XPSD or PSS in trap location X'40' is as follows:

1. Store the current PSWs.
2. Store general registers if PSS.
3. Load the new PSWs.
    a. Set CC3 to one. The other condition code bits remain unchanged from the values loaded from memory.
    b. If bit position 9 (AI) of the XPSD or PSS contains a one, the program counter is implemented by two. If AI contains a zero, the program counter remains unchanged from the values loaded from memory.

MEMORY PROTECTION VIOLATION

A memory protection violation occurs because of a memory map access control bit violation (by a program executed in slave mode or master-protected mode using the memory map). When memory protection violation occurs, the basic processor aborts execution of the current instruction without changing protected memory and traps to location X'40'. Refer to Table 5 for possible changes to registers and memory locations. (The virtual page address that caused the violation is in the fourth PSW word.) The operation of the XPSD or PSS in trap location X'40' is as follows:

1. Store the current PSWs.
2. Store general registers if PSS.
3. Load the new PSWs.
4. Modify the new PSWs.
    a. Set CC4 to one. The other condition code bits remain unchanged from the values loaded from memory.
    b. If bit position 9 (AI) of the XPSD or PSS contains a one, the program counter is incremented by one. If AI contains a zero, the program counter remains unchanged from the value loaded from memory.

WRITE LOCK VIOLATION

A memory write lock violation occurs when an instruction (program in master, master-protected, or slave mode) tries to alter the contents of a write-protected memory page. If a write lock violation occurs, the basic processor aborts execution of the current instruction without changing protected memory and traps to location X'40'. (Refer to Table 5 for possible changes to registers and memory locations.) (The virtual page address that caused the violation is the fourth PSW word.) The operation of the XPSD or PSS in trap location X'40' is as follows:

1. Store the current PSWs.
2. Store general registers if PSS.
3. Load the new PSWs.
4. Modify the new PSWs.
    a. Set CC3 and CC4 to ones. The other condition code bits remain unchanged from the values loaded from memory.
    b. If bit position 9 (AI) of the XPSD or PSS contains a one, the program counter is incremented by three. If AI contains a zero, the program counter remains unchanged from the value loaded from memory.

PUSH-DOWN STACK LIMIT TRAP

Push-down stack overflow or underflow can occur during execution of any of the following instructions:

Instruction	Mnemonic	Operation Code
Push Word	PSW	X'09'
Pull Word	PLW	X'08'
Push Multiple	PSM	X'0B'
Pull Multiple	PLM	X'0A'
Modify Stack Pointer	MSP	X'13'

During the execution of any stack-manipulating instruction (see Chapter 3, "Push-down Instructions"), the stack is either pushed (words added to stack) or pulled (words removed from stack). In either case, the space (S) and words (W) fields of the stack pointer doubleword are tested prior to moving any words. If execution of the instruction would cause the space (S) field to become less than 0 or greater than $2^{15}-1$, the instruction is aborted with memory and registers unchanged. If TS (bit 32) of the stack pointer doubleword is set to 0, the basic processor traps to location X'42'. If TS is set to 1, the trap is inhibited and the basic processor processes the next instruction. If execution of the instruction would cause the words (W) field to become less than 0 or greater than $2^{15}-1$, the instruction is aborted with memory and registers unchanged. If TW (bit 48) of the stack pointer doubleword is set to 0, the basic processor traps to location X'42'. If the TW is set to 1, the trap is inhibited and the basic processor processes the next instruction. If trapping is inhibited, CC1 or CC3 is set to 1 to indicate the reason for aborting the instruction. The stack pointer doubleword, memory, and registers are modified only if the instruction is successfully executed.

If a push-down instruction traps, the execution of XPSD or PSS in trap location X'42' is as follows:

1. Store the current PSWs. The condition codes that are stored are those that existed prior to execution of the aborted push-down instruction.
2. Store general registers if PSS.
3. Load the new PSWs. The condition code and instruction address portions of the PSWs remain at the value loaded from memory.

FIXED-POINT OVERFLOW TRAP

Overflow can occur for any of the following instructions:

Instruction	Mnemonic	Operation Code
Load Absolute Word	LAW	X'3B'
Load Absolute Doubleword	LAD	X'1B'
Load Complement Word	LCW	X'3A'
Load Complement Doubleword	LCD	X'1A'
Add Halfword	AH	X'50'
Subtract Halfword	SH	X'58'
Divide Halfword	DH	X'56'
Add Immediate	AI	X'20'
Add Word	AW	X'30'
Subtract Word	SW	X'38'
Divide Word	DW	X'36'
Add Doubleword	AD	X'10'
Subtract Doubleword	SD	X'18'
Modify and Test Halfword	MTH	X'53'
Modify and Test Word	MTW	X'33'
Add Word to Memory	AWM	X'66'

Except for the instructions DIVIDE HALFWORD (DH) and DIVIDE WORD (DW), instruction execution is allowed to proceed to completion. CC2 is set to 1 and CC3 and CC4 represent the actual result (0, -, or +) after overflow.

If the fixed-point arithmetic trap mask (bit 11 of PSWs) is a 1, the basic processor traps to location X'43' instead of executing the next instruction in sequence.

For DW and DH, the instruction execution is aborted without changing any register, and CC2 is set to 1; CC1, CC3, and CC4 remain unchanged from their values at the end of the instruction immediately prior to the DW or DH. If the fixed-point arithmetic trap mask is a 1, the basic processor traps to location X'43' instead of executing the next instruction in sequence.

The execution of XPSD or PSS in trap location X'43' is as follows:

1. Store the current PSWs. (Store general registers if PSS.) If the instruction trapped was any instruction other than DW or DH, the stored condition code is interpreted as follows:

CC1[t]	CC2	CC3	CC4	Meaning
-[tt]	1	0	0	Result after overflow is zero.
-	1	0	1	Result after overflow is negative.
-	1	1	0	Result after overflow is positive.

[t] CC1 remains unchanged for instructions LCW, LAW, LCD, and LAD.

[tt] A hyphen indicates that the condition code bits are not affected by the condition given under the "Meaning" heading.

CC1[†]	CC2	CC3	CC4	Meaning
0	-	-	-	No carry out of bit 0 of the adder (add and subtract instructions only).
1	-	-	-	Carry out of bit 0 of the adder (add and subtract instructions only).

If the instruction trapped was a DW or DH, the stored condition code is interpreted as follows:

CC1	CC2	CC3	CC4	Meaning
-[††]	1	-	-	Overflow

2. Load the new PSWs. The condition code and instruction address portions of the PSWs remain at the value loaded from memory.

FLOATING-POINT ARITHMETIC FAULT TRAP

Floating-point fault detection is performed after the operation called for by the instruction code is performed, but before any results are loaded into the general registers. Thus, a floating-point operation that causes an arithmetic fault is not carried to completion in that the original contents of the general registers are unchanged.

Instead, the basic processor traps to location X'44' with the current condition code indicating the reason for the trap. A characteristic overflow or an attempt to divide by zero always results in a trap condition. A significance check or a characteristic underflow results in a trap condition only if the floating-point mode controls (FS, FZ, and FN) in the current program status words are set to the appropriate state.

If a floating-point instruction traps, the execution of XPSD or PSS in trap location X'44' is as follows:

1. Store the current PSWs. (Store general registers if PSS.) If division is attempted with a zero divisor or if characteristic overflow occurs, the stored condition code is interpreted as follows:

CC1	CC2	CC3	CC4	Meaning
0	1	0	0	Zero divisor.
0	1	0	1	Characteristic overflow, negative result.
0	1	1	0	Characteristic overflow, positive result.

If none of the above conditions occurred but characteristic underflow occurs with floating zero mode bit (FZ) = 1, the stored condition code is interpreted as follows:

CC1	CC2	CC3	CC4	Meaning
1	1	0	1	Characteristic underflow, negative result.
1	1	1	0	Characteristic underflow, positive result.

If none of the above conditions occurred but an addition or subtraction results in either a zero result (with FS = 1 and FN = 0), or a postnormalization shift of more than two hexadecimal places (with FS = 1 and FN = 0), the stored condition code is interpreted as follows:

CC1	CC2	CC3	CC4	Meaning
1	0	0	0	Zero result of addition or subtraction.
1	0	0	1	More than two postnormalizing shifts, negative result.
1	0	1	0	More than two postnormalizing shifts, positive result.

2. Load the new PSWs. The condition code and instruction address portions of the PSWs remain at the values loaded from memory.

DECIMAL ARITHMETIC FAULT TRAP

When either of two decimal fault conditions occurs (see Chapter 3, "Decimal Instructions"), the normal sequencing of instruction is halted, CC1 and CC2 are set according to the reason for the fault condition, and CC3, CC4, memory, and the decimal accumulator remain unchanged by the instruction. If the decimal arithmetic trap mask (bit position 10 of PSW1) is a 0, the instruction execution sequence continues with the next instruction in sequence at the time of fault detection; however, if the decimal arithmetic trap mask contains a 1, the basic processor traps to location X'45' at the time of fault detection. The following are the fault conditions for decimal instructions:

Instruction Name	Mnemonic	Fault
Decimal Load	DL	Illegal digit
Decimal Store	DS	Illegal digit
Decimal Add	DA	Overflow, illegal digit
Decimal Subtract	DS	Overflow, illegal digit
Decimal Multiply	DM	Illegal digit

[†] CC1 remains unchanged for instructions LCW, LAW, LCD, and LAD.

[††] A hyphen indicates that the condition code bits are not affected by the condition given under the "Meaning" heading.

Instruction Name	Mnemonic	Fault
Decimal Divide	DD	Overflow, illegal digit
Decimal Compare	DC	Illegal digit
Decimal Shift Arithmetic	DSA	Illegal digit
Pack Decimal Digits	PACK	Illegal digit
Unpack Decimal Digits	UNPK	Illegal digit
Edit Byte String	EBS	Illegal digit

The execution of XPSD or PSS in trap location X'45' is as follows:

1. Store the current PSWs. (Store general registers if PSS.) The stored condition code is interpreted as follows:

CC1	CC2	CC3	CC4	Meaning
0	1	-	-	All digits legal and overflow.
1	0	-	-	Illegal digit detected.

2. Load the new PSWs. The condition code and instruction address portions of the PSWs remain at the values loaded from memory.

WATCHDOG TIMER RUNOUT TRAP

The watchdog timer monitors and controls the maximum amount of basic processor time each instruction can take. The timer is normally in operation at all times and is initialized at the beginning of each instruction. If the instruction is not completed by the time the watchdog timer has completed its count, the instruction is aborted, TCC1 is set to 0, and a trap occurs immediately to location X'46'. Additional information as to probable cause of delay is provided: TCC2 is set if the basic processor was using the processor bus, TCC3 is set if the basic processor was using the memory bus, TCC4 is set if the basic processor was using the DIO bus. The register altered flag of the PSWs is also set if any register or main memory location has been changed when the trap occurred.

A watchdog timer runout is considered a basic processor fault and the PDF is set. (See "Processor Detected Fault Flag", later in this chapter.)

PROGRAMMED TRAP

The programmed trap occurs at instruction interruptible point. It is set by a WRITE DIRECT (WD). See Chapter 3. The basic processor traps to location X'47'.

CALL INSTRUCTION TRAP

The four CALL instructions (CAL1, CAL2, CAL3, and CAL4) cause the basic processor to trap to location X'48' (for CAL1), X'49' (for CAL2), X'4A' (for CAL3), or X'4B' (for CAL4). Execution of the XPSD or PSS instruction in the trap location is as follows:

1. Store the current PSWs. The stored condition code bits are those that existed prior to the CALL instruction.

2. Store the general registers in PSS.

3. Load the new PSWs.

4. Modify the new PSWs.

   a. The R Field of the CALL instruction is logically ORed with the condition code register as loaded from memory.

   b. If bit 9 (AI) of XPSD or PSS contains a 1, the R field of the CALL instruction is added to the program counter. If AI contains a 0, the program counter remains unchanged from the value loaded from memory.

   Note: Return from a CALL trap will be to the trapping instruction + 1.

HARDWARE ERROR TRAP

A hardware error trap occurs when either a parity or a sequence check fault error is detected by a memory unit, basic processor, or any processor communicating with the basic processor, resulting in a basic processor trap to location X'4C'. The Trap Condition Code bits (TCCs) are set to X'0001' for all hardware fault conditions except general register and control register parity errors, where the TCCs are set to X'0000'.

To determine which of the possible detectable errors is responsible for the hardware error trap, the fault status registers of the various processors in the system must be polled with either the POLP or POLR instruction; the memory's status register must be read with the LMS instruction. The fault status register bit settings for processors and interfaces are given in Appendix C, Table C-1. The fault status register bit settings for the memory unit are given in Appendix C, Table C-2.

If the basic processor detects or receives a report of a hardware error, it attempts automatic retry of the current instruction. If retry is unsuccessful, the basic processor traps to location X'4C'. If retry is successful, the basic processor resumes execution of the next instruction in the program, the Processor Fault Interrupt (PFI) and the "successful instruction retry" bit (bit position 11) in the Basic Processor Fault Status Register are set to 1. There is automatic instruction

retry only for hardware errors that would otherwise result in a basic processor trap to location X'4C'. Automatic instruction retry is inhibited if:

1. The current instruction is being executed as a trap or interrupt instruction;

2. The Register Altered bit (bit position 60) of the current PSWs is set to 1 at the time of detection of the hardware error; or

3. The Retry Inhibit bit (bit position 0) in the basic processor control register is set to 1.

INSTRUCTION EXCEPTION TRAP

The instruction exception trap occurs whenever the basic processor detects a set of operations that are called for in an instruction but cannot be executed because of either a hardware restriction or a previous event.

The different conditions that cause the instruction exception trap are:

1. A processor-detected fault that occurs during the execution of an interrupt or trap entry sequence. An interrupt or trap entry sequence is defined as the sequence of events that consists of: (a) initiating an interrupt or trap; (b) accessing the instruction in the interrupt or trap location; and (c) executing that instruction, including the exchange of the program status words, if required. Note that instructions executed as a result of the interrupt or trap location are not considered part of the entry sequence.

2. An illegal instruction is found in the trap (not XPSD or PSS) or interrupt (not XPSD, PSS, MTB, MTH, MTW) location when executing a trap or interrupt sequence.

3. Bit positions 12-14 of the MOVE TO MEMORY CONTROL (MMC) instruction are interpreted as an illegal configuration. This is, any configuration other than 100, 010, 101, 001, or 011.

4. The set of operations, primarily doubleword and byte-string instructions, that yield an unpredictable result when an incorrect register is specified; this type of fault is called "invalid register designation" and includes the following instructions".[†]

   Register 0 Specified

   Edit Byte String (EBS)

   Odd Register Specified

   Add Doubleword (AD)

   Subtract Doubleword (SD)

   Floating Add Long (FAL)

   Floating Subtract Long (FSL)

   Floating Multiply Long (FML)

   Floating Divide Long (FDL)

   Translate Byte String (TBS)

   Translate and Test Byte String (TTBS)

   Edit Byte String (EBS)

   Move to Memory Control (MMC)

[†] "Invalid register designation" faults do not set the PDF flag.

TRAP CONDITION CODE

The Trap Condition Code (TCC) differentiates between the different fault types. Table 4 shows the settings of the TCC for the various faults that may be detected during a trap or interrupt entry sequence.

Table 4. TCC Setting for Instruction Exception Trap X'4D'

Fault Type	TCC
Trap or interrupt sequence and processor-detected fault.	1 1 1 1
Trap or interrupt sequence with invalid instruction.	1 1 0 0
MMC configuration invalid.	0 0 1 0
Invalid register designation.	0 0 0 1

POWER ON TRAP

Power On causes the basic processor to reset and then trap to location X'50'. This will occur only following restoration of power after an interruption of less than 500 milliseconds.

POWER OFF TRAP

Power Off occurs at interruptible point. As source power is going off, the basic processor traps to location X'51' and allows sufficient time for storage of information before the system becomes inoperable.

PROCESSOR DETECTED FAULT FLAG

The Processor Detected Fault (PDF) flag aids in solving a multiple error problem. Most traps occur because of a dynamic programming consideration (i.e., overflow, attempted division by zero, incorrect use of an instruction or address, etc.) and recovery is easily handled by another software

subroutine. However, with certain classes of errors, if a second error occurs while the basic processor is attempting to recover from the first error, unpredictable results occur. Included in this class of traps are the hardware error trap, some cases of the instruction exception trap, and the watchdog timer runout trap. Upon the first occurrence of this type of trap, the PDF flag is set.

When the PDF flag is set, the processor fault interrupt, the memory fault interrupt, and count pulse interrupts are automatically inhibited. The other interrupts may or may not be inhibited as specified by the program status words, which are loaded when the trap entry XPSD or PSS is executed. The PDF flag is normally reset by the last instruction of a trap routine, which is an LPSD or PLS instruction having bit 10 equal to 0 and bit 11 equal to 1.

If a second PDF is detected before the PDF flag is reset, the basic processor "hangs up" until the PDF flag is reset either by the operator entering the command for RESET BASIC PROCESSOR or RESET SYSTEM on the operator's console.

This reset will cause the following actions:

1. The processor fault status register is cleared.
2. The PDF flag is cleared and the processor fault interrupt generated flag is cleared.
3. The PSWs are cleared to zero except that the instruction address is set to location X'26'.
4. The basic processor will begin execution with the instruction contained in location X'26'.

REGISTER ALTERED BIT

Complete recoverability after a trap may require that no main memory location, no fast memory register, and no part (or flags) of the PSWs be changed when the trap occurs. If any of these registers or flags are changed, the Register Altered bit (60) of the old PSWs is set to 1 and is saved by the trap XPSD.

Changes to CC1-CC4 cause the Register Altered bit to be set only if the instruction requires these condition code bits as subsequent inputs.

Traps caused by conditions detected during operand fetch and store memory cycles, such as nonexistent memory, access protection violation, and memory parity error may or may not leave registers, memory, and PSWs unchanged, depending on when they occur during instruction execution. Generally, these traps are recoverable. This is done by checking for protection violations and nonexistent memory at the beginning of execution in case of a multiple operand access instruction, restoring the original register contents if execution cannot be completed because of a trap, and not loading the first word of the PSWs until a possible trap condition due to access of the second word could have been detected. Table 5 contains a list of instructions and indicates for these instructions what registers, memory locations, and bits of the PSWs, if any, have been changed when a trap due to an operand access memory cycle occurs.

Table 5. Registers Changed at Time of a Trap Due to an Operand Access

Instructions	Changes
AI, CI, LCFI, LI, MI	Immediate type, no operand access.
CAL1-CAL4, SF, S, WAIT, RD, WD, RIO, POLR, POLP, DSA	No operand access.
LRA	Has operand access but traps are suppressed; register bits and condition codes are set instead.
LB, LCF, LRP, CB LH, LAH, LCH, AH, SH, MH, DH, CH LW, LAW, LCW, AW, SW, MW, DW, CW	No operand store, registers and PSWs unchanged when trap due to operand fetch. CC1-4 may be changed but are not used as input to any of these instructions.
LD, LAD, LCD, AD, SD, CD, CLM, CLR EOR, OR, AND, LS, INT, CS FAS, FSS, FMS, FDS, FAL, FSL, FML, FDL	Registers and memory are preserved, condition codes may be changed but are not used as input to these instructions.
AWM, XW, STS, MTB, MTH, MTW STB, STCF, STH, STW, LAS	Memory will be altered and the Register Altered bit set.
EXU, BCR, BCS BAL, BDR, BIR	If the branch condition is true (always for EXU and BAL) and a trap occurs due to access of the indirect address or of the next (branched to or executed) instruction, the register used is left unchanged and the program address saved in the PSWs is the address of the branch or execute instruction.

Table 5. Registers Changed at Time of a Trap Due to an Operand Access (cont.)

Instructions	Changes
MBS, CBS, TBS, TTBS, EBS, MMC DA, DS, DL, DST, DC, DM, DD, PACK, UNPK, LM, STM, PLM, PSM, STD	Registers and memory may be changed and the Register Altered bit set.
CVA, CVS	If a trap occurs, the instruction will be aborted before altering registers. CC1-4 may be changed but not used as input to any of these instructions.
XPSD, LPSD, PSS, PLS	If a trap occurs due to storing the old PSWs or fetching the new PSWs, the instruction is aborted before changing the old PSWs.
SIO, TIO, TDV, HIO, AIO, RIO	If trap occurs, the instruction will be aborted without altering condition codes, registers, or memory.
*ANLZ	An indirect ANALYZE instruction executed in the master-protected mode will trap. No registers are altered.

3. INSTRUCTION REPERTOIRE

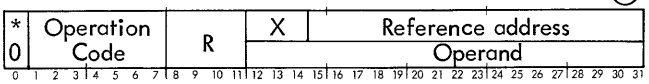
This chapter describes the instructions, grouped in the following functional classes:

1. Load and Store
2. Analyze and Interpret
3. Fixed-Point Arithmetic
4. Comparison
5. Logical
6. Shift
7. Conversion
8. Floating-Point Arithmetic
9. Decimal
10. Byte String
11. Push Down
12. Execute and Branch
13. Call
14. Control (privileged)
15. Input/Output (privileged)

Instructions are described in the following format:

MNEMONIC① INSTRUCTION NAME②
(Addressing Type③, Privileged④,
Interrupt Action⑤)

⑥



Description⑦

Affected⑧ Trap⑨

Symbolic Notation⑩

Condition Code Settings⑪

Trap Action⑫

Example⑬

1. MNEMONIC is the code used by Xerox assemblers to produce the instruction's basic operation code.
2. INSTRUCTION NAME is the instruction's descriptive title.
3. The instruction's addressing type is one of the following:

   a. Byte index alignment: the reference address field of the instruction (plus the displacement value) can be used to address a byte in main memory or in the current block of general registers.

   b. Halfword index alignment: the reference address field of the instruction (plus the displacement value) can be used to address a halfword in main memory or in the current block of general registers.

   c. Word index alignment: the reference address field of the instruction (plus the displacement value) can be used to address any word in main memory or in the current block of general registers.

   d. Doubleword index alignment: the reference address field of the instruction (plus the displacement value) can be used to address any doubleword in main memory or in the current block of general registers. The addressed doubleword is automatically located within doubleword storage boundaries. (The low order bit of the reference address is ignored.)

   e. Immediate operand: the instruction word contains an operand value used as part of the instruction execution. If indirect addressing is attempted with this type of instruction (i.e., bit 0 of the instruction word is a 1), the instruction is treated as a nonexistent instruction, and the basic processor unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40', the "nonallowed operation" trap. Indexing does not apply to this type of instruction.

   f. Immediate displacement: the instruction word contains an address displacement used as part of the instruction execution. If indirect addressing is attempted with this type of instruction, the basic processor treats the instruction as a nonexistent instruction, and it unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40'. Indexing does not apply to this type of instruction.

4. If the instruction is not executable while the basic processor is in the slave mode, it is labeled "privileged" If execution of a privileged instruction is attempted while the basic processor is in the slave mode, it unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40'.
5. If the instruction can be successfully resumed after its execution sequence has been interrupted by an interrupt acknowledgment, the instruction is labeled

"continue after interrupt". In the case of the "continue after interrupt" instructions, certain general registers contain intermediate results or control information that allows the instruction to continue properly.

6. Instruction format:

   a. Indirect addressing – If bit position 0 of the instruction format contains an asterisk (*), the instruction can use either indirect or direct addressing. If bit position 0 of the instruction format contains a 0, the instruction is of the immediate operand type, which is treated as a nonexistent instruction if indirect addressing is attempted (resulting in a trap to location X'40').

   b. Operation code – The operation code field (bit positions 1-7) of the instruction is shown in hexadecimal notation. For certain I/O instructions, the operation code field is extended and includes bit positions 15-17 of the instruction.

   c. R field – If the register address field (bit positions 8-11) of the instruction format contains the character "R", the instruction can specify any register in the current block of general registers as an operand source, result destination, or both; otherwise, the function of this field is determined by the instruction.

   d. X field – If the index register address field (bit positions 12-14) of the instruction format contains the character "X", the instruction specifies indexing with any one of registers 1 through 7 in the current block of general registers; otherwise, the function of this field is determined by the instruction.

   e. Reference address field – Normally, the address field (bit positions 15-31) of the instruction format is used as the reference address value for real, real extended, and virtual addresses (see Chapter 2). This reference address field is also used to address I/O systems (see I/O instructions later in this chapter and also Chapter 4). For immediate operand instructions, this field is augmented with the contents of the X field, as illustrated, to form a 20-bit operand.

   f. Value field – In some fixed-point arithmetic instructions, bit positions 12-31 of the instruction format contain the word "value". The field is treated as a 20-bit integer, with negative integers represented in two's complement form.

   g. Displacement field – In the byte string instructions bit positions 12-31 of the instruction format contain the byte "displacement". In the execution of the instruction, this field is used to modify the source address of an operand, the destination address of a result, or both.

   h. Reserved fields – In any format diagram that depicts system inputs (i.e., instruction, data word), a shaded area represents a field that is ignored by the basic processor (i.e., the content of the shaded field has no effect on instruction execution). It should not be used or must be coded with 0's to preclude conflict with possible future modifications.

      In any format diagram that depicts system outputs (i.e., general register, memory word modified by an instruction, or I/O status word), a shaded area represents a field whose content is indeterminate and must not be used (i.e., masked).

7. The description of the instruction defines the operations performed by the basic processor in response to the instruction configuration depicted by the instruction format diagram. Any instruction configuration that causes an unpredictable result is so specified in the description.

8. All programmable registers and storage areas that can be affected by the instruction are listed (symbolically) after the word "Affected". The instruction address portion of the program status words is considered to be affected only if a branch condition can occur as a result of the instruction execution, since the instruction address is incremented by 1 as part of every instruction execution.

9. All trap conditions that may be invoked by the execution of the instruction are listed after the word "Trap". Trap locations are summarized in the section "Trap System" in Chapter 2.

10. The symbolic notation presents the instruction operation as a series of generalized symbolic statements. The symbolic terms used in the notation are defined in the Appendix, "Glossary of Symbolic Terms".

11. Condition Code settings are given for each instruction that affects the condition code. A 0 or a 1 under any of columns 1, 2, 3, or 4 indicates that the instruction causes a 0 or 1 to be placed in CC1, CC2, CC3 or CC4, respectively, for the reasons given. If a hyphen (-) appears in columns 1, 2, 3, or 4, that portion of the condition code is not affected. For example, the following condition code settings are given for a comparison instruction:

1	2	3	4	Result of comparison
-	-	0	0	Equal.
-	-	0	1	Register operand is arithmetically less than effective operand.
-	-	1	0	Register operand is arithmetically greater than effective operand.
-	1	-	-	The logical product of the two operands is nonzero.
-	0	-	-	The logical product (AND) of the two operands is zero.

CC1 is unchanged by the instruction. CC2 indicates whether or not the two operands have 1's in corresponding bit positions, regardless of their arithmetic relationship. CC3 and CC4 are set according to the arithmetic relationship of the two operands, regardless of whether or not the two operands have 1's in corresponding bit positions. For example, if the register operand is arithmetically less than the effective operand, and the two operands both have 1's in at least one corresponding bit position, the condition code setting for the comparison instruction is:

1	2	3	4
-	1	0	1

The above statements about the condition code are valid only if no trap occurs before the successful completion of the instruction execution cycle. If a trap does occur during the instruction execution, the condition code is normally reset to the value it contained before the instruction was started and the register altered bit (bit 60 in PSWs) is set to 1 if a register has been altered. Then the appropriate trap location is activated.

12. Actions taken by the basic processor for those trap conditions that may be invoked by the execution of the instruction are described. The description includes the criteria for the trap condition, any controlling trap mask or inhibit bits, and the action taken by the basic processor.

    Note: To avoid unnecessary repetition, the three trap conditions that apply to all instructions (i.e., nonallowed operations, parity error, and watchdog timer runout) are not described for each instruction.

13. Some instruction descriptions provide one or more examples to illustrate the results of the instruction. These examples are intended only to show how the instructions operate, and not to demonstrate their full capability. Within the examples, hexadecimal notation is used to represent the contents of general registers and storage locations. Condition code settings are shown in binary notation. The character "x" is used to indicate irrelevant or ignored information.

    Note: In the following text, BP is used as an abbreviation for basic processor.

LOAD/STORE INSTRUCTIONS

The load/store instructions are as follows:

Instruction Name	Mnemonic
Load Immediate	LI
Load Byte	LB
Load Halfword	LH
Load Word	LW
Load Doubleword	LD
Load Complement Halfword	LCH
Load Absolute Halfword	LAH
Load Complement Word	LCW
Load Absolute Word	LAW
Load Complement Doubleword	LCD
Load Absolute Doubleword	LAD
Load Read Address (see "Control Instructions")	LRA
Load and Set	LAS
Load Memory Status (see "Control Instructions")	LMS
Load Selective	LS
Load Multiple	LM
Load Conditions and Floating Control Immediate	LCFI
Load Conditions and Floating Control	LCF
Load Virtual Address Word	LVAW
Exchange Word	XW
Store Byte	STB
Store Halfword	STH
Store Word	STW
Store Doubleword	STD
Store Selective	STS
Store Multiple	STM
Store Conditions and Floating Control	STCF

The load and store instructions operate with information fields of byte, halfword, word, and doubleword lengths. Load instructions load the information indicated into one or more of the general registers in the current register block. Load instructions do not affect the source of information; however, nearly all load instructions provide a condition code setting that indicates the following information about

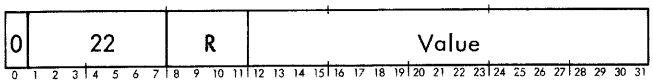
the contents of the affected general register(s) after the instruction is successfully completed:

Condition code settings:

1	2	3	4	Result
-	-	0	0	Zero – the result in the affected register(s) is all 0's.
-	-	0	1	Negative – register R contains a 1 in bit position 0.
-	-	1	0	Positive – register R contains a 0 in bit position 0, and at least one 1 appears in the remainder of the affected register(s) (or appeared during execution of the current instruction.)
-	0	-	-	No fixed-point overflow – the result in the affected register(s) is arithmetically correct.
-	1	-	-	Fixed-point overflow – the result in the affected register(s) is arithmetically incorrect.

Store instructions affect only that portion of memory storage that corresponds to the length of the information field specified by the operation code of the instruction; thus, register bytes are stored in memory byte locations, register halfwords in memory halfword locations, register words in memory word locations, and register doublewords in memory doubleword locations. Store instructions do not affect the contents of the general register specified by the R field of the instruction, unless the same register is also specified by the effective virtual address of the instruction.

LI LOAD IMMEDIATE
(Immediate operand)



LOAD IMMEDIATE extends the sign of the value field (bit position 12 of the instruction word) 12 bit positions to the left and then loads the 32-bit result into register R.

Affected: (R), CC3, CC4 Trap: Nonexistent instruction, if bit 0 is a 1.

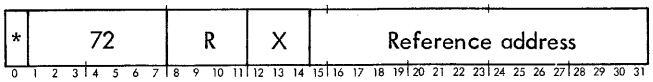
$(I)_{12-31SE} \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive

If LI is indirectly addressed, it is treated as a nonexistent instruction, in which case the BP unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the condition code unchanged.

LB LOAD BYTE
(Byte index alignment)



LOAD BYTE loads the effective byte into bit positions 24-31 of register R and clears bit positions 0-23 of the register to all 0's.

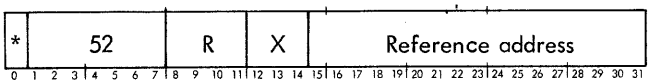
Affected: (R), CC3, CC4

$EB \longrightarrow R_{24-31}$; $0 \longrightarrow R_{0-23}$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	1	0	Nonzero

LH LOAD HALFWORD
(Halfword index alignment)



LOAD HALFWORD extends the sign of the effective halfword 16 bit positions to the left and then loads the 32-bit result into register R.

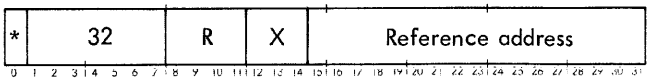
Affected: (R), CC3, CC4

$EH_{SE} \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive

LW LOAD WORD
(Word index alignment)



LOAD WORD loads the effective word into register R.

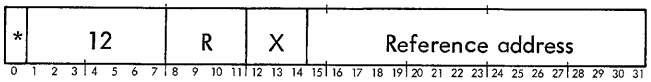
Affected: (R), CC3, CC4

$EW \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive

LD LOAD DOUBLEWORD
(Doubleword index alignment)



LOAD DOUBLEWORD loads the 32 low-order bits of the effective doubleword into register Ru1 and then loads the 32 high-order bits of the effective doubleword into register R.

If R is an odd value, the result in register R is the 32 high-order bits of the effective doubleword. The condition code settings are based on the effective doubleword, rather than the final result in register R (see example 3, below).

Affected: (R), (Ru1), CC3, CC4

$ED_{32-63} \longrightarrow Ru1$; $ED_{0-31} \longrightarrow R$

Condition code settings:

1	2	3	4	Effective doubleword
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive

Example 1, even R field value:

		Before execution	After execution
ED	=	X'0123456789ABCDEF'	X'0123456789ABCDEF'
(R)	=	xxxxxxxx	X'01234567'
(Ru1)	=	xxxxxxxx	X'89ABCDEF'
CC	=	xxxx	xx10

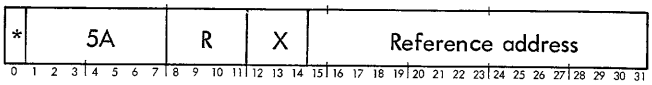
Example 2, odd R field value:

		Before execution	After execution
ED	=	X'0123456789ABCDEF'	X'0123456789ABCDEF'
(R)	=	xxxxxxxx	X'01234567'
CC	=	xxxx	xx10

Example 3, odd R field value:

		Before execution	After execution
ED	=	X'0000000012345678'	X'0000000012345678'
(R)	=	xxxxxxxx	X'00000000'
CC	=	xxxx	xx10

LCH LOAD COMPLEMENT HALFWORD
(Halfword index alignment)



LOAD COMPLEMENT HALFWORD extends the sign of the effective halfword 16 bit positions to the left and then loads the 32-bit two's complement of the result into register R. (Overflow cannot occur.)

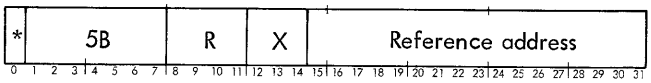
Affected: (R), CC3, CC4

$-\left[EH_{SE}\right] \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive

LAH LOAD ABSOLUTE HALFWORD
(Halfword index alignment)



If the effective halfword is positive, LOAD ABSOLUTE HALFWORD extends the sign of the effective halfword 16 bit positions to the left and then loads the 32-bit result in register R. If the effective halfword is negative, LAH extends the sign of the effective halfword 16 bit positions to the left and then loads the 32-bit two's complement of the result into register R. (Overflow cannot occur.)

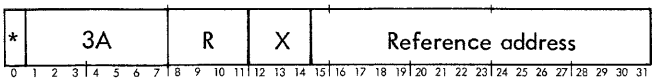
Affected: (R), CC3, CC4

$EH_{SE} \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	1	0	Nonzero

LCW LOAD COMPLEMENT WORD
(Word index alignment)



LOAD COMPLEMENT WORD loads the 32-bit two's complement of the effective word into register R. Fixed-point overflow occurs if the effective word is -2^{31} (X'80000000') in which case the result in register R is -2^{31} and CC2 is set to 1; otherwise, CC2 is reset to 0.

Affected: (R),CC2,CC3,CC4 Trap: Fixed-point overflow.

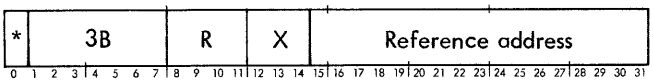
-EW ⟶ R

Condition code settings:

1	2	3	4	Result in R
-	0	0	0	Zero
-	-	0	1	Negative
-	0	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	0	1	Fixed-point overflow

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after execution of LOAD COMPLEMENT WORD; otherwise, the BP executes the next instruction in sequence.

LAW LOAD ABSOLUTE WORD
(Word index alignment)



If the effective word is positive, LOAD ABSOLUTE WORD loads the effective word into register R. If the effective word is negative, LAW loads the 32-bit two's complement of the effective word into register R. Fixed-point overflow occurs if the effective word is -2^{31} (X'80000000'), in which case the result in register R is -2^{31}, and CC2 is set to 1; otherwise, CC2 is reset to 0.

Affected: (R),CC2,CC3,CC4 Trap: Fixed-point overflow

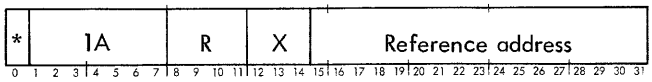
|EW| ⟶ R

Condition code settings:

1	2	3	4	Result in R
-	0	0	0	Zero
-	-	1	0	Nonzero
-	0	-	-	No fixed-point overflow
-	1	0	1	Fixed-point overflow (sign bit on)

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after execution of LOAD ABSOLUTE WORD; otherwise, the BP executes the next instruction in sequence.

LCD LOAD COMPLEMENT DOUBLEWORD
(Doubleword index alignment)



LOAD COMPLEMENT DOUBLEWORD forms the 64-bit two's complement of the effective doubleword, loads the 32 low-order bits of the result into register Ru1, and then loads the 32 high-order bits of the result into register R.

If R is an odd value, the result in register R is the 32 high-order bits of the two's complemented doubleword. The condition code settings are based on the two's complement of the effective doubleword, rather than the final result in register R.

Fixed-point overflow occurs if the effective doubleword is -2^{63} (X'8000000000000000'), in which case the result in registers R and Ru1 is -2^{63} and CC2 is set to 1; otherwise, CC2 is reset to 0.

Affected: (R),(Ru1),CC2, CC3,CC4 Trap: Fixed-point overflow

$[-ED]_{32-63} \longrightarrow$ Ru1; $[-ED]_{0-31} \longrightarrow$ R

Condition code settings:

1	2	3	4	Two's complement of effective doubleword
-	0	0	0	Zero
-	-	0	1	Negative
-	0	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	0	1	Fixed-point overflow

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after execution of LOAD COMPLEMENT DOUBLEWORD; otherwise, the BP executes the next instruction in sequence.

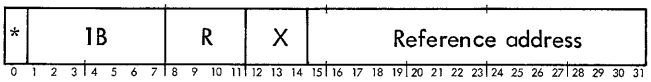
Example 1, even R field value:

	Before execution	After execution
ED	= X'0123456789ABCDEF'	X'1023456789ABCDEF'
(R)	= xxxxxxxx	X'FEDCBA98'
(Ru1)	= xxxxxxxx	X'76543211'
CC	= xxxx	x001

Example 2, odd R field value:

	Before execution	After execution
ED =	X'0123456789ABCDEF'	X'0123456789ABCDEF'
(R) =	xxxxxxxx	X'FEDCBA98'
CC =	xxxx	x001

LAD LOAD ABSOLUTE DOUBLEWORD
(Doubleword index alignment)



If the effective doubleword is positive, LOAD ABSOLUTE DOUBLEWORD loads the 32 low-order bits of the effective doubleword into register Ru1, and then loads the 32 high-order bits of the effective doubleword into register R. If R is an odd value, the result in register R is the 32 high-order bits of the effective doubleword. The condition code settings are based on the effective doubleword, rather than the final result in register R.

If the effective doubleword is negative, LAD forms the 64-bit two's complement of the effective doubleword, loads the 32 low-order bits of the two's complemented doubleword into register Ru1, and then loads the 32 high-order bits of the two's complemented doubleword into register R. If R is an odd value, the result in register R is the 32 high-order bits of the two's complemented doubleword. The condition code settings are based on the two's complement of the effective doubleword, rather than the final result in register R.

Fixed-point overflow occurs if the effective doubleword is -2^{63} (X'8000000000000000'), in which case the result in registers R and Ru1 is -2^{63} and CC2 is set to 1; otherwise, CC2 is reset to 0.

Affected: (R),(Ru1),CC2, CC3,CC4 Trap: Fixed-point overflow

$|ED|_{32-63} \longrightarrow Ru1$; $|ED|_{0-31} \longrightarrow R$

Condition code settings:

1	2	3	4	Absolute value of effective doubleword
-	0	0	0	Zero
-	-	1	0	Nonzero
-	0	-	-	No fixed-point overflow
-	1	0	1	Fixed-point overflow (sign bit on)

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after execution of LOAD ABSOLUTE DOUBLEWORD; otherwise, the BP executes the next instruction in sequence.

Example 1, even R field value:

	Before execution	After execution
ED =	X'0123456789ABCDEF'	X'0123456789ABCDEF'
(R) =	xxxxxxxx	X'01234567'
(Ru1) =	xxxxxxxx	X'89ABCDEF'
CC =	xxxx	x010

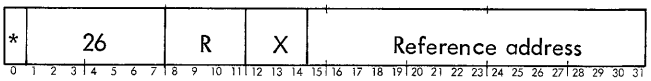
Example 2, even R field value:

	Before execution	After execution
ED =	X'FEDCBA9876543210'	X'FEDCBA9876543210'
(R) =	xxxxxxxx	X'01234567'
(Ru1) =	xxxxxxxx	X'89ABCDF0'
CC =	xxxx	x010

Example 3, odd R field value:

	Before execution	After execution
ED =	X'0123456789ABCDEF'	X'0123456789ABCDEF'
(R) =	xxxxxxxx	X'01234567'
CC =	xxxx	x010

LAS LOAD AND SET
(Word index alignment)



LOAD AND SET loads the effective word into R. If the effective address is equal to or greater than 16, a one is stored in the sign position of the effective location. If the effective address is equal to or less than 15 (effective location is a general register), the sign bit remains unchanged. This instruction is used to interlock multiple processors from the simultaneous execution of certain sections of code or from the simultaneous access to certain tables.

Affected: (R), CC3, CC4

$EW \longrightarrow R$

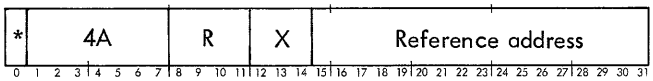
$1 \longrightarrow EW_0$, if $EA \geq 16$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive

Note: Write locks protect memory and traps are not inhibited during the execution of LAS.

LS LOAD SELECTIVE
(Word index alignment)



Register Ru1 contains a 32-bit mask. If R is an even value, LOAD SELECTIVE loads the effective word into register R in those bit positions selected by a 1 in corresponding bit positions of register Ru1. The contents of register R are not affected in those bit positions selected by a 0 in corresponding bit positions of register Ru1.

If R is an odd value, LS logically ANDs the contents of register R with the effective word and loads the result into register R. If corresponding bit positions of register R and the effective word both contain 1's, a 1 remains in register R; otherwise, a 0 is placed in the corresponding bit position of register R.

Affected: (R), CC3, CC4

If R is even, $[EW_n(Ru1)] \cup [(R)_n(\overline{Ru1})] \longrightarrow R$

If R is odd, $EW_n(R) \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero.
-	-	0	1	Bit 0 of register R is a 1.
-	-	1	0	Bit 0 of register R is a 0 and bit positions 1-31 of register R contain at least one 1.

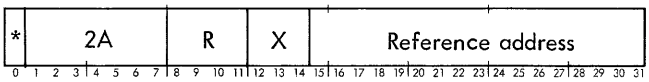
Example 1, even R field value:

		Before execution	After execution
EW	=	X'01234567'	X'01234567'
(Ru1)	=	X'FF00FF00'	X'FF00FF00'
(R)	=	xxxxxxxx	X'01xx45xx'
CC	=	xxxx	xx10

Example 2, odd R field value:

		Before execution	After execution
EW	=	X'89ABCDEF'	X'89ABCDEF'
(R)	=	X'F0F0F0F0'	X'80A0C0E0'
CC	=	xxxx	xx01

LM LOAD MULTIPLE
(Word index alignment)



LOAD MULTIPLE loads a sequential set of words into a sequential set of registers, the set of words to be loaded begins with the word pointed to by the effective address of LM, and the set of registers begins with register R. The set of registers is treated modulo 16 (i.e., the next register loaded after register 15 is register 0 in the current register block).

The number of words to be loaded into the general registers is determined by the setting of the condition code immediately before the execution of LM. (The desired value of the condition code can be set with LCF or LCFI.) An initial value of 0000 for the condition code causes 16 consecutive words to be loaded into the register block.

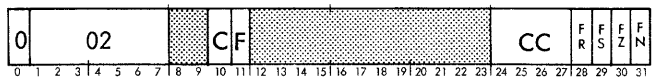
Affected: (R) to (R+CC-1)

$(EWL \longrightarrow R; (EWL+1) \longrightarrow R+1), \ldots, (EWL+CC-1) \longrightarrow R+CC-1$

The LM instruction may cause a trap if its operation extends into a page of memory that is protected by the access protection codes. A trap may also occur if the operation extends into a nonexistent memory region.

If the effective virtual address of the LM instruction is in the range 0 through 15, then the words to be loaded are taken from the general registers rather than from main memory. In this case the results will be unpredictable if any of the source registers are also used as destination registers.

LCFI LOAD CONDITIONS AND FLOATING CONTROL IMMEDIATE
(Immediate operand)



If bit position 10 of the instruction word contains a 1, LOAD CONDITIONS AND FLOATING CONTROL IMMEDIATE loads the contents of bit positions 24 through 27 of the instruction word into the condition code; however, if bit 10 is 0, the condition code is not affected.

If bit position 11 of the instruction word contains a 1, LCFI loads the contents of bit positions 28 through 31 of the instruction word into the floating round (FR), floating

significance (FS), floating zero (FZ), and floating normalize (FN) mode control bits, respectively (in the program status words); however, if bit 11 is 0, the FR, FS, FZ, and FN control bits are not affected. The functions of the floating-point control bits are described in the section "Floating-Point Arithmetic Instructions".

Affected: CC, FR, FS, FZ, FN Trap: Nonexistent instruction, if bit 0 is a 1.

If $(I)_{10} = 1$, $(I)_{24-27} \longrightarrow CC$

If $(I)_{10} = 0$, CC is not affected.

If $(I)_{11} = 1$, $(I)_{28-31} \longrightarrow FR, FR, FS, FZ, FN$

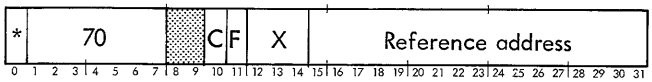
If $(I)_{11} = 0$, FR, FS, FZ, and FN not affected.

Condition code settings, if $(I)_{10} = 1$:

1	2	3	4
$(I)_{24}$	$(I)_{25}$	$(I)_{26}$	$(I)_{27}$

If LCFI is indirectly addressed, it is treated as a nonexistent instruction, in which case the computer unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the condition code unchanged.

LCF LOAD CONDITIONS AND FLOATING CONTROL
(Byte index alignment)



If bit position 10 of the instruction word contains a 1, LOAD CONDITIONS AND FLOATING CONTROL loads bits 0 through 3 of the effective byte into the condition code; however, if bit 10 is 0, the condition code is not affected.

If bit position 11 of the instruction word contains a 1, LCF loads bits 4 through 7 of the effective byte into the floating round (FR), floating significance (FS), floating zero (FZ), and floating normalize (FN) mode control bits, respectively; however, if bit 11 is 0, the FR, FS, FZ, and FN control bits are not affected. The functions of the floating-point mode control bits are described in the section "Floating-Point Arithmetic Instructions".

Affected: CC, FR, FS, FZ, FN

If $(I)_{10} = 1$, $EB_{0-3} \longrightarrow CC$

If $(I)_{10} = 0$, CC not affected

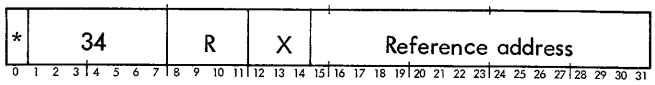
If $(I)_{11} = 1$, $EB_{4-7} \longrightarrow FR, FS, FZ, FN$

If $(I)_{11} = 0$, FR, FS, FZ, FN not affected

Condition code settings, if $(I)_{10} = 1$:

1	2	3	4
$(EB)_0$	$(EB)_1$	$(EB)_2$	$(EB)_3$

LVAW LOAD VIRTUAL ADDRESS WORD
(Word index alignment)



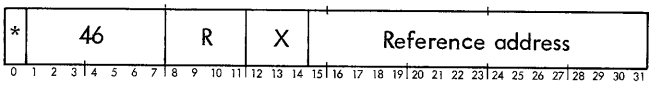
LOAD VIRTUAL ADDRESS WORD loads bit positions 15-31 of register R with the effective virtual word address of the instruction while bit positions 0-14 of register R are cleared to zero.

Affected: (R)

$EVA \longrightarrow R_{15-31}$, $0 \longrightarrow R_{0-14}$

Note: Condition code is not affected by LVAW.

XW EXCHANGE WORD
(Word index alignment)



EXCHANGE WORD exchanges the contents of register R with the contents of the effective word location.

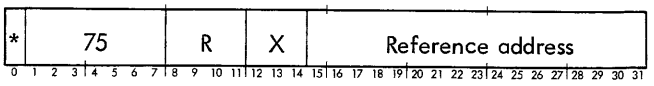
Affected: (R), (EWL), CC3, CC4

$(R) \longleftrightarrow (EWL)$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive

STB STORE BYTE
(Byte index alignment)

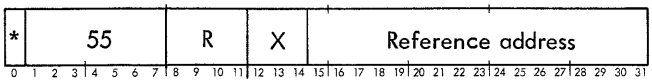


STORE BYTE stores the contents of bit positions 24-31 of register R into the effective byte location.

Affected: (EBL)

$(R)_{24-31} \longrightarrow EBL$

STH STORE HALFWORD
(Halfword index alignment)



STORE HALFWORD stores the contents of bit positions 16-31 of register R into the effective halfword location. If the information in register R exceeds halfword data limits, CC2 is set to 1; otherwise, CC2 is reset to 0.

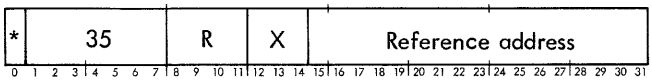
Affected: (EHL), CC2

$(R)_{16-31} \longrightarrow EHL$

Condition code settings:

1	2	3	4	Information in R
-	0	-	-	$(R)_{0-16}$ = all 0's or all 1's.
-	1	-	-	$(R)_{0-16} \neq$ all 0's or all 1's.

STW STORE WORD
(Word index alignment)

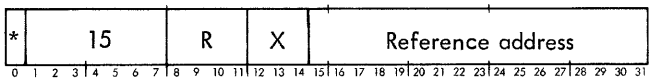


STORE WORD stores the contents of register R into the effective word location.

Affected: (EWL)

(R) $\longrightarrow$ EWL

STD STORE DOUBLEWORD
(Doubleword index alignment)



STORE DOUBLEWORD stores the contents of register R into the 32 high-order bit positions of the effective doubleword location and then stores the contents of register Ru1 into the 32 low-order bit positions of the effective doubleword location.

Affected: (EDL)

$(R) \longrightarrow EDL_{0-31}$; $(Ru1) \longrightarrow EDL_{32-63}$

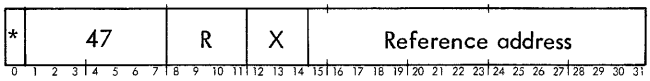
Example 1, even R field value:

	Before execution	After execution
(R) =	X'01234567'	X'01234567'
(Ru1) =	X'89ABCDEF'	X'89ABCDEF'
(EDL) =	xxxxxxxxxxxxxxxx	X'0123456789ABCDEF'

Example 2, odd R field value:

	Before execution	After execution
(R) =	X'89ABCDEF'	X'89ABCDEF'
(EDL) =	xxxxxxxxxxxxxxxx	X'89ABCDEF89ABCDEF'

STS STORE SELECTIVE
(Word index alignment)



Register Ru1 contains a 32-bit mask. If R is an even value, STORE SELECTIVE stores the contents of register R into the effective word location in those bit positions selected by a 1 in corresponding bit positions of register Ru1; the effective word remains unchanged in those bit positions selected by a 0 in corresponding bit positions of register Ru1.

If R is an odd value, STS logically inclusive ORs the contents of register R with the effective word and stores the result into the effective word location. The contents of register R are not affected.

Affected: (EWL)

If R is even, $[(R) \cap (Ru1)] \cup [EW \cap \overline{(Ru1)}] \longrightarrow EWL$

If R is odd, $(R) \cup EW \longrightarrow EWL$

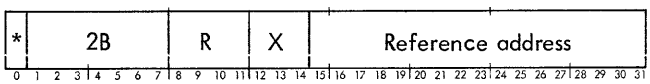
Example 1, even R field value:

	Before execution	After execution
(R) =	X'12345678'	X'12345678'
(Ru1) =	X'F0F0F0F0'	X'F0F0F0F0'
EW =	xxxxxxxx	X'1x3x5x7x'

Example 2, odd R field value:

	Before execution	After execution
(R) =	X'00FF00FF'	X'00FF00FF'
EW =	X'12345678'	C'12FF56FF'

STM STORE MULTIPLE
(Word index alignment)



STORE MULTIPLE stores the contents of a sequential set of registers into a sequential set of word locations. The set of locations begins with the location pointed to by the effective word address of STM, and the set of registers begins with register R. The set of registers is treated modulo 16 (i.e., the

next sequential register after register 15 is register 0). The number of registers to be stored is determined by the value of the condition code immediately before execution of STM. (The condition code can be set to the desired value before execution of STM with LCF or LCFI.) An initial value of 0000 for the condition code causes 16 general registers to be stored.

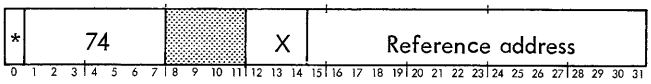
Affected: (EWL) to (EWL+CC-1)

(R) ⟶ EWL, (R+1) ⟶ EWL+1, . . . , (R+CC-1) ⟶ EWL+CC-1

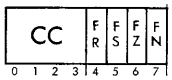
The STM instruction may cause a trap if its operation extends into a page of memory that is protected by the access protection codes or the write locks. A trap may also occur if the operation extends into a nonexistent memory region.

If the effective virtual address of the STM instruction is in the range 0 through 15, then the registers indicated by the R field of the STM instruction are stored in the general registers rather than main memory. In this case, the results will be unpredictable if any of the source registers are also used as destination registers.

STCF STORE CONDITIONS AND FLOATING CONTROL
(Byte index alignment)



STORE CONDITIONS AND FLOATING CONTROL stores the current condition code and the current value of the floating round (FR), floating significance (FS), floating zero (FZ), and floating normalize (FN) mode control bits of the program status words into the effective byte location as follows:

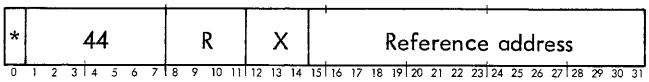


Affected: (EBL)

$(PSWs)_{0-7}$ ⟶ EBL

ANALYZE/INTERPRET INSTRUCTIONS

ANLZ ANALYZE
(Word index alignment)



ANALYZE evaluates the effective word as an instruction. The ANALYZE instruction always sets the condition codes to indicate the addressing type of the analyzed instruction (see condition code settings and Table 6). Except when the analyzed instruction is an immediate operand instruction, an effective virtual address for the analyzed instruction is also calculated and loaded into register R.

Table 6. ANALYZE Table for Operation Codes

X'n'	X'00'+n	X'20'+n	X'40'+n	X'60'+n
00	–	AI	TTBS	CBS
01	–	CI	TBS	MBS
02	LCFI ⑨tt	LI	–①tt	–
03	–	MI	–	EBS
04	CAL1	SF	ANLZ	BDR
05	CAL2	S	CS	BIR
06	CAL3	LAS	XW	AWM
07	CAL4	–	STS	EXU
08	PLW	CVS	EOR	BCR
09	PSW	CVA	OR	BCS
0A	PLM	LM ⑧tt	LS	BAL
0B	PSM	STM	AND	INT
0C	PLS†	LRA†	SIO†	RD†
0D	PSS†	LMS†	TIO†	WD†
0E	LPSD† ⑫tt	WAIT†	TDV†	AIO†
0F	XPSD†	LRP†	HIO†	MMC†
10	AD	SW	AH	LCF
11	CD	CW	CH	CB
12	LD	LW	LH	LB
13	MSP	MTW	MTH	MTB
14	–	LVAW	–	STCF
15	STD	STW	STH	STB
16	–	DW	DH ④tt	PACK ⓪tt
17	–	MW	MH	UNPK
18	SD	SW	SH	DS
19	CLM	CLR	–	DA
1A	LCD	LCW	LCH	DD
1B	LAD	LAW	LAH	DM
1C	FSL	FSS	–	DSA
1D	FAL	FAS	–	DC
1E	FDL	FDS	–	DL
1F	FML	FMS	–	DST

†Privileged instructions.

ttDecimal value of condition code settings when analyzed instruction calls for direct addressing. If analyzed instruction calls for indirect addressing, add 2 to the value shown.

The nonexistent instruction, the privileged instruction violation, and the unimplemented instruction trap conditions can never occur during execution of the ANLZ instruction. However, either the nonexistent memory address condition or the memory protection violation trap condition (or both) can occur as a result of any memory access initiated by the ANLZ instruction. If either of these trap conditions occurs, the instruction address stored by an XPSD in trap location X'40' is always the virtual address of the ANLZ instruction.

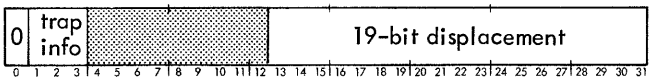
The detailed operation of ANALYZE is as follows:

1. The contents of the location pointed to by the effective virtual address of the ANLZ instruction is obtained. This effective word is the instruction to be analyzed. From a memory-protection viewpoint, the instruction (to be analyzed) is treated as an operand of the ANLZ instruction; that is, the analyzed instruction may be obtained from any memory area to which the program has read access.

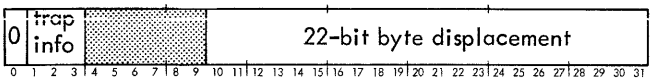
2. If the operation code portion of the effective word specifies an immediate-addressing instruction type, the condition code is set to indicate the addressing type, and instruction execution proceeds to the next instruction in sequence after ANLZ. The original contents of register R are not changed when the analyzed instruction is of the immediate-addressing type.

   If the operation code portion of the effective word specifies a reference-addressing instruction type, the condition code is set to indicate the addressing type of the analyzed instruction and the effective address of the analyzed instruction is computed (using all of the normal address computation rules). If bit 0 of the effective word is a 1, the contents of the memory location specified by bits 15-31 of the effective word are obtained and then used as a direct address. The nonallowed operation trap (memory protection violation or nonexistent memory address) can occur as a result of the memory access. Indexing is always performed (with an index register in the current register block) if bits 12-14 of the analyzed instruction are nonzero. During real extended addressing, the effective virtual address of the analyzed instruction is aligned as an integer displacement value and loaded into register R, according to the instruction addressing type, as follows:

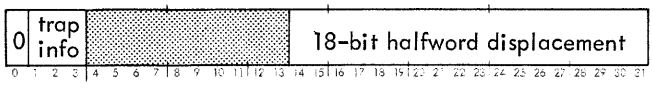
Byte Addressing: MA=0



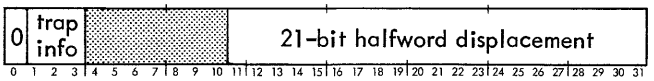
Byte Addressing: MA=1, MM=0



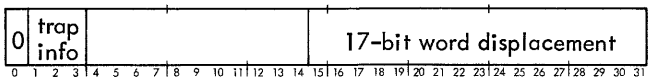
Halfword Addressing: MA=0



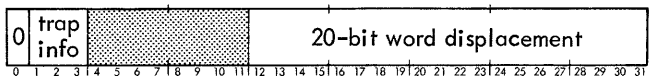
Halfword Addressing: MA=1, MM=0



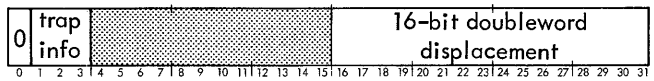
Word Addressing: MA=0



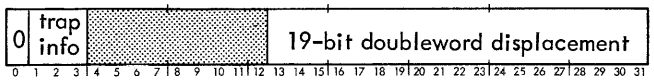
Word Addressing: MA=1, MM=0



Doubleword Addressing: MA=0



Doubleword Addressing: MA=1, MM=0



When the ANALYZE instruction is executed in the master-protected mode and a trap condition occurs, it traps only on an indirect ANALYZE. Otherwise, instead of trapping it completes its execution by storing in register R the address that would have caused the instruction to trap. Since the mode is master-protected, the access protection codes will apply to the interpretation of addresses. If a slave mode program is trapped because an instruction has referenced protected memory, the ANALYZE instruction in the master-protected mode can determine which address actually caused the trap.

To aid the interpreting program, when operating in the master-protected mode, the ANLZ instruction uses bits 1, 2, and 3 of register R to indicate which memory access initiated by the ANLZ would have trapped. The meaning of the possible codes in register R(1-3) is as follows:

Register R Bits

R1	R2	R3	Meaning
0	0	0	Successful generation of the effective virtual address of the analyzed instruction. The CCs are set to the addressing type of the analyzed instruction and R(10-31) contain the effective virtual address of the analyzed instruction aligned as an integer displacement value according to the instruction addressing type.
0	0	1	The indirect reference of the analyzed instruction would have trapped because it was either nonexistent, memory protected, or had a parity error. The CCs are set to the addressing type of the analyzed instruction and R(10-31) contain the virtual address of the indirect reference of the analyzed instruction aligned as a word displacement.

R1	R2	R3	Meaning
0	1	1	The effective virtual address of the ANLZ instruction would have trapped because it was either nonexistent, memory protected, or had a parity error. The CCs are indeterminate since the instruction to be analyzed may not have been fetched (nonexistent memory). R(10-31) contain the effective virtual address of the ANLZ instruction aligned as a word displacement.

If no trap condition occurs, ANLZ will execute normally and return the effective address of the instruction analyzed.

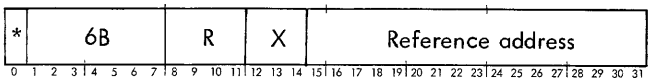
Table 6 shows the instruction set as a 4 by 32 matrix (arranged as a function of the operation code). This table also shows how the instruction set is divided into six groups as a function of the addressing type (delineated by heavy lines). For example, if the operation code of the analyzed instruction is either X'02', X'20', X'21', X'22', or X'23', then CC1 is set to 1, CC2 is set to 0, CC3 is set to 0 (when analyzed instruction specifies direct addressing), and CC4 is set to 1. The decimal equivalent of the condition code setting for this group of immediate, word addressing type of instructions is shown as a 9 within a circle. The decimal equivalents of the condition code settings for the other five groups are shown in the same manner. If the analyzed instruction calls for indirect addressing, CC3 is always set to a 1 and the decimal value of the condition code setting shown in Table 6 should be increased by 2.

Affected: (R), CC

Condition code settings:

1	2	3	4	Instruction addressing type
0	0	-	0	Byte
0	0	-	1	Immediate, byte
0	1	-	0	Halfword
1	0	-	0	Word
1	0	-	1	Immediate, word
1	1	-	0	Doubleword
-	-	0	-	Direct addressing ($EW_0 = 0$)
-	-	1	-	Indirect addressing ($EW_0 = 1$)

INT INTERPRET
(Word index alignment)



INTERPRET loads bits 0-3 of the effective word into the condition code, loads bits 16-31 of the effective word into bit positions 16-31 of register Ru1 (and loads 0's into bit positions 0-15 of register Ru1, loads bits 4-15 of the effective word into bit positions 20-31 of register R (and clears the remaining bits of register R). If R is an odd value, INT loads bits 0-3 of the effective word into the condition code, loads bits 16-31 of the effective word into bit positions 16-31 of register R, and loads 0's into bit positions 0-15 of register R (bits 4-15 of the effective word are ignored in this case).

Affected: (R), (Ru1), CC

$EW_{0-3} \longrightarrow CC$

$EW_{4-15} \longrightarrow R_{20-31}; 0 \longrightarrow R_{0-19}$

$EW_{16-31} \longrightarrow Ru1_{16-31}; 0 \longrightarrow Ru1_{0-15}$

Condition code settings:

1	2	3	4
$(EW)_0$	$(EW)_1$	$(EW)_2$	$(EW)_3$

Example 1, even R field value:

	Before execution	After execution
EW =	X'12345678'	X'12345678'
(R) =	xxxxxxxx	X'00000234'
(Ru1) =	xxxxxxxx	X'00005678'
CC =	xxxx	0001

FIXED-POINT ARITHMETIC INSTRUCTIONS

The fixed-point arithmetic instructions are:

Instruction Name	Mnemonic
Add Immediate	AI
Add Halfword	AH
Add Word	AW
Add Doubleword	AD
Subtract Halfword	SH
Subtract Word	SW
Subtract Doubleword	SD
Multiply Immediate	MI
Multiply Halfword	MH
Multiply Word	MW
Divide Halfword	DH

Instruction Name	Mnemonic
Divide Word	DW
Add Word to Memory	AWM
Modify and Test Byte	MTB
Modify and Test Halfword	MTH
Modify and Test Word	MTW

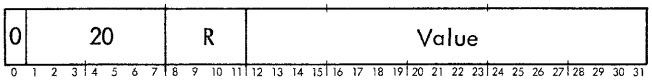
The fixed-point arithmetic instruction set performs binary addition, subtraction, multiplication, and division with integer operands that may be data, addresses, index values, or counts. One operand may be either in the instruction word itself or may be in one or two of the current general registers; the second operand may be either in main memory or in one or two of the current general registers. For most of these instructions, both operands may be in the same general register, thus permitting the doubling, squaring, or clearing the contents of a register by using a reference address value equal to the R field value.

All fixed-point arithmetic instructions provide a condition code setting that indicates the following information about the result of the operation called for by the instruction:

Condition code settings:

1	2	3	4	Result
-	-	0	0	Zero – the result in the specified general register(s) is all zeros.
-	-	0	1	Negative – the instruction has produced a fixed-point negative result.
-	-	1	0	Positive – the instruction has produced a fixed-point positive result.
-	0	-	-	Fixed-point overflow has not occurred during execution of an add, subtract, or divide instruction, and the result is correct.
-	1	-	-	Fixed-point overflow has occurred during execution of an add, subtract, or divide instruction. For addition and subtraction, the incorrect result is loaded into the designated register(s). For a divide instruction, the designated register(s), and CC1, CC3, and CC4 are not affected.
0	-	-	-	No carry – for an add or subtract instruction, there was no carry of a 1-bit out of the high-order (sign) bit position of the result.
1	-	-	-	Carry – for an add or subtract instruction, there was a 1-bit carry out of the sign bit position of the result. (Subtracting zero will always produce carry.)

AI ADD IMMEDIATE
(Immediate operand)



The value field (bit positions 12-31 of the instruction word) is treated as a 20-bit, two's complement integer. ADD IMMEDIATE extends the sign of the value field (bit position 12 of the instruction word) 12 bit positions to the left, adds the resulting 32-bit value to the contents of register R, and loads the sum into register R.

Affected: (R), CC

Trap: Fixed-point overflow, or nonexistent instruction if bit 0 is a 1.

$(R) + (I)_{12\text{-}31SE} \longrightarrow R$

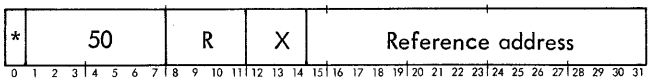
Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from bit position 0
1	-	-	-	Carry from bit position 0

If AI is indirectly addressed, it is treated as a nonexistent instruction, in which case the BP unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the condition code unchanged.

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after loading the sum into register R; otherwise, the BP executes the next instruction in sequence.

AH ADD HALFWORD
(Halfword index alignment)



ADD HALFWORD extends the sign of the effective halfword 16 bit positions to the left (to form a 32-bit word in which bit positions 0-15 contain the sign of the effective halfword), adds the 32-bit result to the contents of register R, and loads the sum into register R.

Affected: (R), CC

Trap: Fixed-point overflow

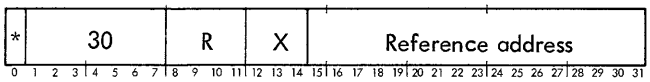
$(R) + EH_{SE} \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from bit position 0
1	-	-	-	Carry from bit position 0

If CC2 is set to 1 and the fixed-point arithmetic trap mask is 1, the BP traps to location X'43' after loading the sum into register R; otherwise, the BP executes the next instruction in sequence.

AW ADD WORD
(Word index alignment)



ADD WORD adds the effective word to the contents of register R and loads the sum into register R.

Affected: (R), CC Trap: Fixed-point overflow

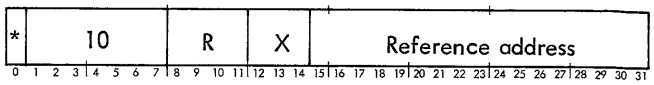
(R) + EW ⟶ R

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from bit position 0
1	-	-	-	Carry from bit position 0

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after loading the sum into register R; otherwise, the BP executes the next instruction in sequence.

AD ADD DOUBLEWORD
(Doubleword index alignment)



ADD DOUBLEWORD adds the effective doubleword to the contents of registers R and Ru1 (treated as a single, 64-bit register); loads the 32 low-order bits of the sum into register Ru1 and then loads the 32 high-order bits of the sum into register R. R must be an even value; if R is an odd value, the BP traps with the contents in register R unchanged.

Affected: (R), (Ru1), CC Trap: Fixed-point overflow, instruction exception

(R, Ru1) + ED ⟶ R, Ru1

Condition code settings:

1	2	3	4	Result in R, Ru1
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from bit position 0
1	-	-	-	Carry from bit position 0

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after loading the sum into registers R and Ru1; otherwise, the BP executes the next instruction in sequence.

The R field of the AD instruction must be an even value for proper operation of the instruction; if the R field of AD is an odd value, the instruction traps to location X'4D', instruction exception trap.

Example 1, even R field value:

		Before execution	After execution
ED	=	X'33333333EEEEEEEE'	X'33333333EEEEEEEE'
(R)	=	X'11111111'	X'44444445'
(Ru1)	=	X'33333333'	X'22222221'
CC	=	xxxx	0010

SH SUBTRACT HALFWORD
(Halfword index alignment)

SUBTRACT HALFWORD extends the sign of the effective halfword 16 bit positions to the left (to form a 32-bit word in which bit positions 0-15 contain the sign of the effective halfword), forms the two's complement of the resulting word, adds the complemented word to the contents of register R, and loads the sum into register R.

Affected: (R), CC Trap: Fixed-point overflow

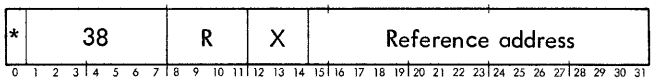
$-EH_{SE} + (R) \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from bit position 0
1	-	-	-	Carry from bit position 0

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after loading the sum into register R; otherwise, the BP executes the next instruction in sequence.

SW SUBTRACT WORD
(Word index alignment)



SUBTRACT WORD forms the two's complement of the effective word, adds that complement to the contents of register R, and loads the sum into register R.

Affected: (R), CC Trap: Fixed-point overflow

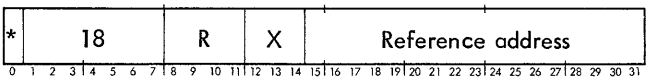
$-EW + (R) \longrightarrow R$

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from bit position 0
1	-	-	-	Carry from bit position 0

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after loading the sum into register R; otherwise, the BP executes the next instruction in sequence.

SD SUBTRACT DOUBLEWORD
(Doubleword index alignment)



SUBTRACT DOUBLEWORD forms the 64-bit two's complement of the effective doubleword, adds the complemented doubleword to the contents of registers R and Ru1 (treated as a single, 64-bit register), loads the 32 low-order bits of the sum into register Ru1 and loads the 32 high-order bits of the sum into register R.

Affected: (R), (Ru1), CC Trap: Fixed-point overflow, instruction exception

$-ED + (R, Ru1) \longrightarrow R, Ru1$

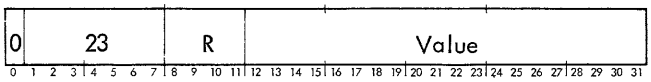
Condition code settings:

1	2	3	4	Result in R, Ru1
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from bit position 0
1	-	-	-	Carry from bit position 0

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after the result is loaded into registers R and Ru1; otherwise, the BP executes the next instruction in sequence.

The R field of the SD instruction must be an even value for proper operation of the instruction; if the R field of SD is an odd value, the instruction traps to location X'4D', instruction exception trap; the contents in register R remain unchanged.

MI MULTIPLY IMMEDIATE
(Immediate operand)



The value field (bit positions 12-31 of the instruction word) is treated as a 20-bit, two's complement integer. MULTIPLY IMMEDIATE extends the sign of the value field (bit position 12) of the instruction word 12 bit positions to the left and multiplies the resulting 32-bit value by the contents of register Ru1, then loads the 32 high-order bits of the product into register R, and then loads the 32 low-order bits of the product into register Ru1.

If R is an odd value, the result in register R is the 32 low-order bits of the product. Thus, in order to generate a 64-bit product, the R field of the instruction must be even and the multiplicand must be in register R+1. The condition code settings are based on the 64-bit product formed during instruction execution, rather than on the final contents of register R. Overflow cannot occur.

Affected: (R), (Ru1), CC2, CC3, CC4

Trap: Nonexistent instruction if bit 0 is a 1.

$(Ru1) \times (I)_{12\text{-}31SE} \longrightarrow R, Ru1$

Condition code settings:

1	2	3	4	64-bit product
-	-	0	0	Zero.
-	-	0	1	Negative.
-	-	1	0	Positive
-	0	-	-	Result is correct, as represented in register Ru1.
-	1	-	-	Result is not correctly representable in register Ru1 alone.

If MI is indirectly addressed, it is treated as a nonexistent instruction, in which case the BP unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R, register Ru1, and the condition code unchanged; otherwise, the BP executes the next instruction in sequence.

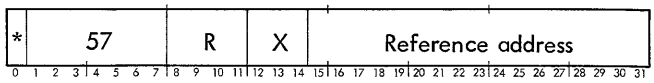
Example 1, even R field value:

		Before execution	After execution
$(I)_{12\text{-}31}$	=	X'70000'	X'70000'
(R)	=	xxxxxxxx	X'00007000'
(Ru1)	=	X'10001000'	X'70000000'
CC	=	xxxx	x110

Example 2, odd R field value:

		Before execution	After execution
$(I)_{12\text{-}31}$	=	X'01234'	X'01234'
(R)	=	X'00030002'	X'369C2468'
CC	=	xxxx	x010

MH MULTIPLY HALFWORD
(Halfword index alignment)



MULTIPLY HALFWORD multiplies the contents of bit positions 16-31 of register R by the effective halfword (with both halfwords treated as signed, two's complement integers) and stores the product in register Ru1 (overflow cannot occur). If R is an even value, the original multiplier in register R is preserved, allowing repetitive halfword multiplication with a constant multiplier; however, if R is an odd value, the product is loaded into the same register. Overflow cannot occur.

Affected: (Ru1), CC3, CC4

$(R)_{16\text{-}31} \times EH \longrightarrow Ru1$

Condition code settings:

1	2	3	4	Result in Ru1
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive

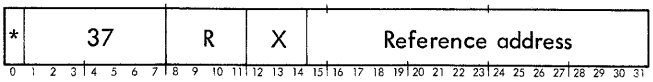
Example 1, even R field value:

		Before execution	After execution
EH	=	X'FFFF'	X'FFFF'
(R)	=	X'xxxx000A'	X'xxxx000A'
(Ru1)	=	xxxxxxxx	X'FFFFFFF6'
CC	=	xxxx	xx01

Example 2, odd R field value:

		Before execution	After execution
EH	=	X'FFFF'	X'FFFF'
(R)	=	X'xxxx000A'	X'FFFFFFF6'
CC	=	xxxx	xx01

MW MULTIPLY WORD
(Word index alignment)



MULTIPLY WORD multiplies the contents of register Ru1 by the effective word, loads the 32 high-order bits of the product into register R and then loads the 32 low-order bits of the product into register Ru1 (overflow cannot occur).

If R is odd value, the result in register R is the 32 low-order bits of the product. Thus, in order to generate a 64-bit product, the R field of the instruction must be even and the multiplicand must be in register R+1. The condition code settings are based on the 64-bit product formed during instruction execution, rather than on the final contents of register R.

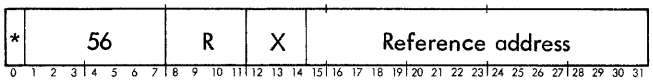
Affected: (R), (Ru1), CC

(Ru1) x EW ⟶ R, Ru1

Condition code settings:

1	2	3	4	64-bit product
-	-	0	0	Zero.
-	-	0	1	Negative.
-	-	1	0	Positive.
-	0	-	-	Result is correct, as represented in register Ru1.
-	1	0	0	Result is not correctly representable in register Ru1 alone.

DH DIVIDE HALFWORD
(Halfword index alignment)



DIVIDE HALFWORD divides the contents of register R (treated as a 32-bit fixed-point integer) by the effective halfword and loads the quotient into register R. If the absolute value of the quotient cannot be correctly represented in 32 bits, fixed-point overflow occurs; in which case CC2 is set to 1 and the contents of register R, and CC1, CC3, and CC4 are unchanged.

Affected: (R), CC2, CC3, CC4 Trap: Fixed-point overflow

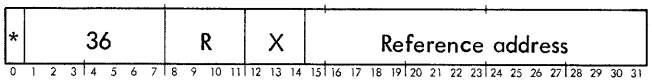
(R) ÷ EH ⟶ R

Condition code settings:

1	2	3	4	Result in R
-	0	0	0	Zero quotient, no overflow.
-	0	0	1	Negative quotient, no overflow.
-	0	1	0	Positive quotient, no overflow.
-	1	-	-	Fixed-point overflow.

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' with the contents of register R, CC1, CC3, and CC4 unchanged.

DW DIVIDE WORD
(Word index alignment)



DIVIDE WORD divides the contents of registers R and Ru1 (treated as a 64-bit fixed-point integer) by the effective word, loads the integer remainder into register R and then loads the integer quotient into register Ru1. If a nonzero remainder occurs, the remainder has the same sign as the dividend (original contents of register R). If R is an odd value, DW forms a 64-bit register operand by extending the sign of the contents of register R 32 bit positions to the left, then divides the 64-bit register operand by the effective word, and loads the quotient into register R. In this case, the remainder is lost and only the contents of register R are affected.

If the absolute value of the quotient cannot be correctly represented in 32 bits, fixed-point overflow occurs; in which case CC2 is set to 1 and the contents of register R, register Ru1, CC1, CC3, and CC4 remain unchanged; otherwise, CC2 is reset to 0, CC3 and CC4 reflect the quotient in register Ru1, and CC1 is unchanged.

Affected: (R), (Ru1), CC2 CC3, CC4 Trap: Fixed-point overflow

(R, Ru1) ÷ EW ⟶ R (remainder), Ru1 (quotient)

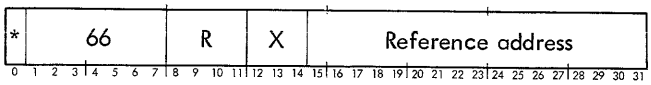
Condition code settings:

1	2	3	4	Result in Ru1
-	0	0	0	Zero quotient, no overflow.
-	0	0	1	Negative quotient, no overflow.
-	0	1	0	Positive quotient, no overflow.
-	1	-	-	Fixed-point overflow.

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' with the

original contents of register R, register Ru1, CC1, CC3, and CC4 unchanged; otherwise, the BP executes the next instruction in sequence.

AWM ADD WORD TO MEMORY[†]
(Word index alignment)



ADD WORD TO MEMORY adds the contents of register R to the effective word and stores the sum in the effective word location. The sum is stored regardless of whether or not overflow occurs.

Affected: (EWL), CC Trap: Fixed-point overflow

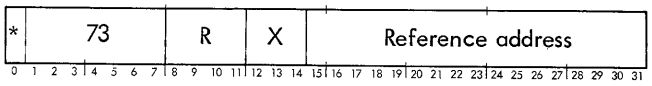
EW + (R) ⟶ EWL

Condition code settings:

1	2	3	4	Result in EWL
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from bit position 0
1	-	-	-	Carry from bit position 0

If CC2 is set to 1 and fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after the result is stored in the effective word location; otherwise, the BP executes the next instruction in sequence.

MTB MODIFY AND TEST BYTE[†]
(Byte index alignment)



If the value of the R field is nonzero, the high-order bit of the R field (bit position 8 of the instruction word) is extended 4 bit positions to the left, to form a byte with bit positions 0-4 of that byte equal to the high-order bit of the R field. This byte is added to the effective byte and then (if no memory protection violation occurs) the sum is stored in the effective byte location and the condition code is set according to the value of the resultant byte. This process allows modification of a byte by any number in the range -8 through +7, followed by a test.

If the value of the R field is zero, the effective byte is tested for being a zero or nonzero value. The condition code is set according to the result of the test, but the effective byte is not affected. A memory write-protection violation cannot occur in this case; however, a memory read-protection violation can occur.

Affected: CC if $(I)_{8-11} \neq 0$
(EBL) and CC if $(I)_{8-11} \neq 0$

If $(I)_{8-11} \neq 0$, EB + $(I)_{8-11SE}$ ⟶ EBL and set CC

If $(I)_{8-11} = 0$, test byte and set CC

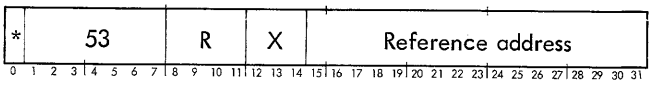
Condition code settings:

1	2	3	4	Result in EBL
-	0	0	0	Zero
-	0	1	0	Nonzero
0	-	-	-	No carry from byte
1	-	-	-	Carry from byte

If MTB is executed in an interrupt or trap location, the condition code is not affected and a 20-bit reference address is used, as described under "Interrupt and Trap Entry Addressing", Chapter 2.

Note: All "Modify and Test" instructions in interrupt locations other than Counter 4 use real, or real extended, addressing mode. Counter 4 uses virtual addressing mode.

MTH MODIFY AND TEST HALFWORD[†]
(Halfword index alignment)



If the value of the R field is nonzero, the high-order bit of the R field (bit position 8 of the instruction word) is extended 12 bit positions to the left, to form a halfword with bit positions 0-11 of that halfword equal to the high-order bit of the R field. This halfword is added to the effective halfword and then (if no memory protection violation occurs) the sum is stored in the effective halfword location and the condition code is set according to the value of the resultant halfword. The sum is stored regardless of whether or not overflow occurs. This process allows modification of a halfword by any number in the range -8 through +7, followed by a test.

[†]This instruction requires two memory references to the same location for its execution. To preclude other processors from accessing the effective location during this time, the memory unit containing the effective location is reserved (not accessible to other processors) until the instruction is completed.

If the value of the R field is zero, the effective halfword is tested for being a zero, negative, or positive value. The condition code is set, according to the result of the test, but the effective halfword is not affected. A memory write-protection violation cannot occur in this case; however, a memory read-protection violation can occur.

Affected: CC if $(I)_{8-11} = 0$; Trap: Fixed-point overflow (EHL) and CC if $(I)_{8-11} \neq 0$

If $(I)_{8-11} = 0$, test halfword and set CC

If $(I)_{8-11} \neq 0$, EH + $(I)_{8-11SE}$ ⟶ EHL and set CC

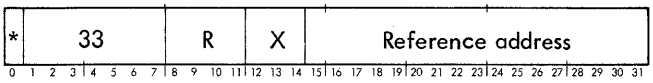
Condition code settings:

1	2	3	4	Result in EHL
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from halfword
1	-	-	-	Carry from halfword

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after the result is stored in the effective halfword location; otherwise, the BP executes the next instruction in sequence.

If MTH is executed in an interrupt or trap location, the condition code is not affected and a 20-bit reference address is used, as described under "Interrupt and Trap Entry Addressing", Chapter 2.

MTW MODIFY AND TEST WORD[t]
(Word index alignment)



If the value of the R field is nonzero, the high-order bit of the R field (bit position 8 of the instruction word) is extended 28 bit positions to the left, to form a word with bit positions 0-27 of that word equal to the high-order bit of the R field. This word is added to the effective word and then (if no memory protection violation occurs) the sum is stored in the effective word location and condition code is set according to the value of the resultant word. The sum is stored regardless of whether or not overflow occurs. This process allows modification of a word by any number in the range -8 through +7, followed by a test.

If the value of the R field is zero, the effective word is tested for being a zero, negative, or positive value. The condition code is set according to the result of the test, but the effective word is not affected. A memory write-protection violation cannot occur in this case; however, a memory read-protection violation can occur.

Affected: CC if $(I)_{8-11} = 0$; Trap: Fixed-point overflow (EWL) and CC if $(I)_{8-11} \neq 0$

If $(I)_{8-11} = 0$, test word and set CC

If $(I)_{8-11} \neq 0$, EW + I_{8-11SE} ⟶ EWL and set CC

Condition code settings:

1	2	3	4	Result in EWL
-	-	0	0	Zero
-	-	0	1	Negative
-	-	1	0	Positive
-	0	-	-	No fixed-point overflow
-	1	-	-	Fixed-point overflow
0	-	-	-	No carry from word
1	-	-	-	Carry from word

If CC2 is set to 1 and the fixed-point arithmetic trap mask (AM) is a 1, the BP traps to location X'43' after the result is stored in the effective word location; otherwise, the BP executes the next instruction in sequence.

If MTW is executed in an interrupt or trap location, the condition code is not affected and a 20-bit reference address is used, as described under "Interrupt and Trap Entry Addressing", Chapter 2.

COMPARISON INSTRUCTIONS

The comparison instructions are:

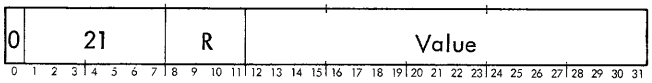
Instruction Name	Mnemonic
Compare Immediate	CI
Compare Byte	CB

[t]This instruction requires two memory references to the same location for its execution. To preclude other processors from accessing the effective location during this time, the memory unit containing the effective location is reserved (not accessible to other processors) until the instruction is completed.

Instruction Name	Mnemonic
Compare Halfword	CH
Compare Word	CW
Compare Doubleword	CD
Compare Selective	CS
Compare With Limits in Register	CLR
Compare With Limits in Memory	CLM

All comparison instructions produce a condition code setting which is indicative of the results of the comparison, without affecting the effective operand in memory and without affecting the contents of the designated register.

CI COMPARE IMMEDIATE
(Immediate operand)



COMPARE IMMEDIATE extends the sign of the value field (bit position 12) of the instruction word 12 bit positions to the left, compares the 32-bit result with the contents of register R (with both operands treated as signed fixed-point quantities), and then sets the condition code according to the results of the comparison.

Affected: CC2, CC3, CC4 Trap: Nonexistent instruction if bit 0 is a 1.

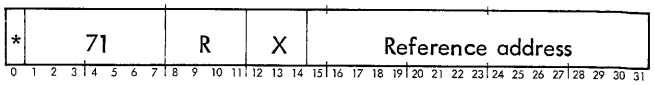
$(R) : (I)_{12-31SE}$

Condition code settings:

1	2	3	4	Result of Comparison
-	-	0	0	Equal.
-	-	0	1	Register value less than immediate value.
-	-	1	0	Register value greater than immediate value.
-	0	-	-	No 1-bits compare, $(R) \cap (I)_{12-32SE} = 0$.
-	1	-	-	One or more 1-bits compare, $(R) \cap (I)_{12-32SE} \neq 0$.

If CI is indirectly addressed, it is treated as a nonexistent instruction, in which case the basic processor unconditionally aborts execution of the instruction (at the time of operation code decoding) and then traps to location X'40' with the condition code unchanged.

CB COMPARE BYTE
(Byte index alignment)



COMPARE BYTE compares the contents of bit positions 24-31 of register R with the effective byte (with both bytes treated as positive integer magnitudes) and sets the condition code according to the results of the comparison.

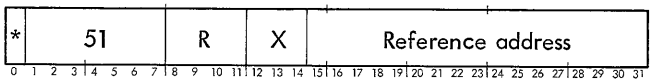
Affected: CC2, CC3, CC4

$(R)_{24-31} : EB$

Condition code settings:

1	2	3	4	Result of Comparison
-	-	0	0	Equal.
-	-	0	1	Register byte less than effective byte.
-	-	1	0	Register byte greater than effective byte.
-	0	-	-	No 1-bits compare, $(R)_{24-31} \cap EB = 0$.
-	1	-	-	One or more 1-bits compare, $(R)_{24-31} \cap EB \neq 0$.

CH COMPARE HALFWORD
(Halfword index alignment)



COMPARE HALFWORD extends the sign of the effective halfword 16 bit positions to the left, then compares the resultant 32-bit word with the contents of register R (with both words treated as signed, fixed-point quantities) and sets the condition code according to the results of the comparison.

Affected: CC2, CC3, CC4

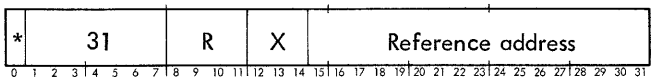
$(R) : EH_{SE}$

Condition code settings:

1	2	3	4	Result of Comparison
-	-	0	0	Equal.
-	-	0	1	Register word less than effective halfword with sign extended.
-	-	1	0	Register word greater than effective halfword with sign extended.

1	2	3	4	Result of Comparison
-	0	-	-	No 1-bits compare, (R) n EH_{SE} = 0.
-	1	-	-	One or more 1-bits compare, (R) n $EH_{SE} \neq 0$.

CW COMPARE WORD
(Word index alignment)



COMPARE WORD compares the contents of register R with the effective word, with both words treated as signed fixed-point quantities, and sets the condition code according to the results of the comparison.

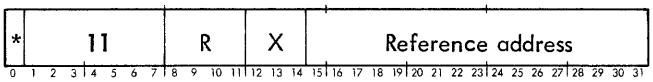
Affected: CC2, CC3, CC4

(R) : EW

Condition code settings:

1	2	3	4	Result of Comparison
-	-	0	0	Equal.
-	-	0	1	Register word less than effective word.
-	-	1	0	Register word greater than effective word.
-	0	-	-	No 1-bits compare, (R) n EW = 0.
-	1	-	-	One or more 1-bits compare, (R) n EW $\neq$ 0.

CD COMPARE DOUBLEWORD



COMPARE DOUBLEWORD compares the effective doubleword with the contents of registers R and Ru1 (with both doublewords treated as signed, fixed-point quantities) and sets the condition code according to the results of the comparison. If the R field of CD is an odd value, CD forms a 64-bit register operand (by duplicating the contents of register R for both the 32 high-order bits and the 32 low-order bits) and compares the effective doubleword with the 64-bit register operand. The condition code settings are based on the 64-bit comparison.

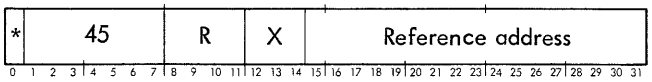
Affected: CC3, CC4

(R, Ru1) : ED

Condition code settings:

1	2	3	4	Result of Comparison
-	-	0	0	Equal.
-	-	0	1	Register doubleword less than effective doubleword.
-	-	1	0	Register doubleword greater than effective doubleword.

CS COMPARE SELECTIVE



COMPARE SELECTIVE compares the contents of register R with the effective word in only those bit positions selected by a 1 in corresponding bit positions of register Ru1 (mask). The contents of register R and the effective word are ignored in those bit positions designated by a 0 in corresponding bit positions of register Ru1. The selected contents of register R and the effective word are treated as positive integer magnitudes, and the condition code is set according to the result of the comparison. If the R field of CS is an odd value; CS compares the contents of register R with the logical product (AND) of the effective word and the contents of register R.

Affected: CC3, CC4

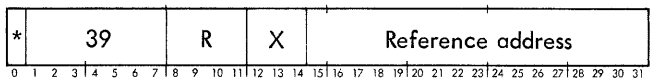
If R is even: (R) n (Ru1) : EW n (Ru1)

If R is odd: (R) : EW n (R)

Condition code settings:

1	2	3	4	Results of Comparison under Mask in Ru1
-	-	0	0	Equal.
-	-	0	1	Register word less than effective word.
-	-	1	0	Register word greater than effective word. (if R is even).

CLR COMPARE WITH LIMITS IN REGISTERS
(Word index alignment)



COMPARE WITH LIMITS IN REGISTERS simultaneously compares the effective word with the contents of register R and with the contents of register Ru1 (with all three words treated as signed fixed-point quantities), and sets the condition code according to the results of the comparisons.

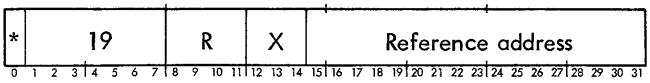
Affected: CC

(R) : EW, (Ru1) : EW

Condition code settings:

1	2	3	4	Result of Comparison
-	-	0	0	Contents of R equal to effective word.
-	-	0	1	Contents of R less than effective word.
-	-	1	0	Contents of R greater than effective word.
0	0	-	-	Contents of Ru1 equal to effective word.
0	1	-	-	Contents of Ru1 less than effective word.
1	0	-	-	Contents of Ru1 greater than effective word.

CLM COMPARE WITH LIMITS IN MEMORY
(Doubleword index alignment)



COMPARE WITH LIMITS IN MEMORY simultaneously compares the contents of register R with the 32 high-order bits of the effective doubleword and with the 32 low-order bits of the effective doubleword, with all three words treated as 32-bit signed quantities, and sets the condition code according to the results of the comparisons.

Affected: CC

(R) : ED_{0-31}; (R) : ED_{32-63}

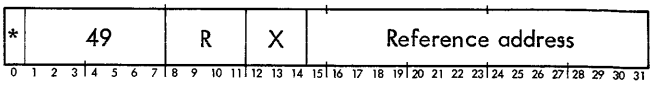
Condition code settings:

1	2	3	4	Result of Comparison
-	-	0	0	Contents of R equal to most significant word, (R) = ED_{0-31}.
-	-	0	1	Contents of R less than most significant word, (R) < ED_{0-31}.
-	-	1	0	Contents of R greater than most significant word, (R) > ED_{0-31}.
0	0	-	-	Contents of R equal to least significant word, (R) = ED_{32-63}.
0	1	-	-	Contents of R less than least significant word, (R) < ED_{32-63}.
1	0	-	-	Contents of R greater than least significant word, (R) > ED_{32-63}.

LOGICAL INSTRUCTIONS

All logical operations are performed bit by corresponding bit between two operands; one operand is in register R and the other operand is the effective word. The result of the logical operation is loaded into register R.

OR OR WORD
(Word index alignment)



OR WORD logically ORs the effective word into register R. If corresponding bits of register R and the effective word are both 0, a 0 remains in register R; otherwise, a 1 is placed in the corresponding bit position of register R. The effective word is not affected.

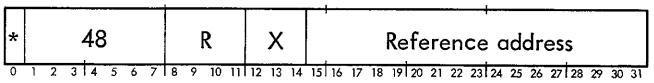
Affected: (R), CC3, CC4

(R) u EW ⟶ R, where 0 u 0 = 0, 0 u 1 = 1, 1 u 0 = 1, 1 u 1 = 1

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero.
-	-	0	1	Bit 0 of register R is a 1.
-	-	1	0	Bit 0 of register R is a 0 and bit positions 1-31 of register R contain at least one 1.

EOR EXCLUSIVE OR WORD
(Word index alignment)



EXCLUSIVE OR WORD logically exclusive ORs the effective word into register R. If corresponding bits of register R and the effective word are different, a 1 is placed in the corresponding bit position of register R; if the contents of the corresponding bit positions are alike, a 0 is placed in the corresponding bit position of register R. The effective word is not affected.

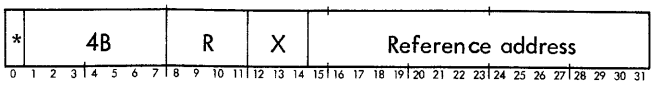
Affected: (R), CC3, CC4

(R) ⓤ EW⟶R

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero.
-	-	0	1	Bit 0 of register R is a 1.
-	-	1	0	Bit 0 of register R is a 0 and bit positions 1-31 of register R contain at least one 1.

AND AND WORD
(Word index alignment)



AND WORD logically ANDs the effective word into register R. If corresponding bits of register R and the effective word are both 1, a 1 remains in register R; otherwise, a 0 is placed in the corresponding bit position of register R. The effective word is not affected.

Affected: (R), CC3, CC4

(R) n EW ⟶ R

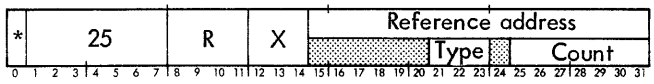
Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero.
-	-	0	1	Bit 0 of register R is a 1.
-	-	1	0	Bit 0 of register R is a 0 and bit positions 1-31 of register R contain at least one 1.

SHIFT INSTRUCTIONS

The instruction format for logical, circular, arithmetic, and searching shift operations is:

S SHIFT
(Word index alignment)



If neither indirect addressing nor indexing is called for in the instruction SHIFT, bit positions 21-23 of the reference address field determine the type, and bit positions 25-31 determine the direction and amount of the shift.

If only indirect addressing is called for in the instruction, bits 15-31 of the instruction are used to access the indirect word and then bits 21-23 and 25-31 of the indirect word determine the type, direction, and amount of the shift.

If only indexing is called for in the instruction, bits 21-23 of the instruction word determine the type of shift; the direction and amount of shift are determined by bits 25-31 of the instruction plus bits 25-31 of the specified index register.

If both indirect addressing and indexing are called for in the instruction, bits 15-31 of the instruction are used to access the indirect word and then bits 21-23 of the indirect word determine the type of shift; the direction and amount of the shift are determined by bits 25-31 of the indirect word plus bits 25-31 of the specified index register.

The effective address does not reference memory. Bit positions 15-20 and 24 of the effective virtual address are ignored. Bit positions 21, 22, and 23 of the effective virtual address determine the type of shift, as follows:

21	22	23	Shift Type
0	0	0	Logical, single register
0	0	1	Logical, double register
0	1	0	Circular, single register
0	1	1	Circular, double register
1	0	0	Arithmetic, single register
1	0	1	Arithmetic, double register
1	1	0	Searching, single register
1	1	1	Searching, double register

Bit positions 25 through 31 of the effective virtual address are a shift count that determines the direction and amount of the shift. The shift count (C) is treated as a 7-bit signed binary integer, with the high-order bit (bit position 25) as the sign (negative integers are represented in two's complement form). A positive shift count causes a left shift of C bit positions. A negative shift count causes a right shift of |C| bit positions. The value of C is within the range: $-64 \leq C \leq +63$.

All double-register shift operations require an even value for the R field of the instruction, and treat registers R and Ru1 as a 64-bit register with the high-order bit (bit position 0 of register R) as the sign for the entire register. If the R field of SHIFT is an odd value and a double-register shift operation is specified, a register doubleword is formed by duplicating the contents of register R for both the 32 high-order bits and the 32 low-order bits of the doubleword. The shift operation is then performed and the 32 high-order bits of the result are loaded into register R.

Overflow occurs (on left shifts only) whenever the value of the sign bit (bit position 0 of register R) changes. At the completion of logical left, circular left, arithmetic left, and searching left shifts, the condition code is set as follows:

1	2	3	4	Result of Shift
0	-	-	-	Even number of 1's shifted off left end of register R.
1	-	-	-	Odd number of 1's shifted off left end of register R†.

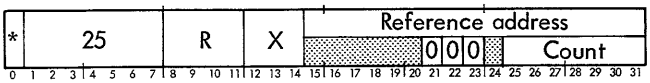
†Not applicable for searching shift.

1	2	3	4	Result of Shift
-	0	-	-	No overflow on left shift.
-	1	-	-	Overflow on left shift.
-	-	-	1	Searching shift terminated with R_0 equal to 1.

At the completion of right shifts, the condition code is set as follows:

1	2	3	4
0	0	-	-

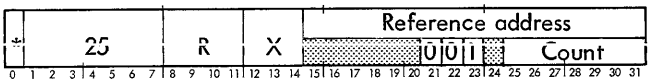
Logical Shift, Single Register



If the shift count, C, is positive, the contents of register R are shifted left C places, the 0's copied into vacated bit positions on the right. (Bits shifted past R_0 are lost.) If C is negative, the contents of register R are shifted right |C| places, with 0's copied into vacated bit positions on the left. (Bits shifted past R_{31} are lost.)

Affected: (R), CC1, CC2

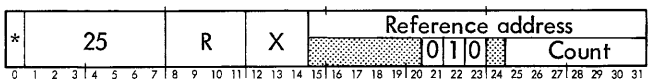
Logical Shift, Double Register



If the shift count, C, is positive, the contents of registers R and Ru1 are shifted left C places, with 0's copied into vacated bit positions on the right. Bits shifted past bit position 0 of register Ru1 are copied into bit position 31 of register R. (Bits shifted past R_0 are lost.) If C is negative, the contents of registers R and Ru1 are shifted right |C| places with 0's copied into vacated bit positions on the left. Bits shifted past bit position 31 of register R are copied into bit position 0 of register Ru1. (Bits shifted past $Ru1_{31}$ are lost.)

Affected: (R), (Ru1), CC1, CC2

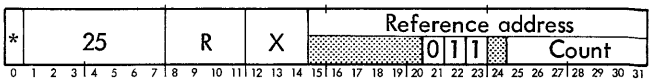
Circular Shift, Single Register



If the shift count, C, is positive, the contents of register R are shifted left C places. Bits shifted past bit position 0 are copied into bit position 31. (No bits are lost.) If C is negative, the contents of register R are shifted right |C| places. Bits shifted past bit position 31 are copied into bit position 0. (No bits are lost.)

Affected: (R), CC1, CC2

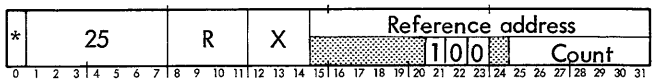
Circular Shift, Double Register



If the shift count, C, is positive, the contents of registers R and Ru1 are shifted left C places. Bits shifted past bit position 0 of register R are copied into bit position 31 of register Ru1. (No bits are lost.) If C is negative, the contents of registers R and Ru1 are shifted right |C| places. Bits shifted past bit position 31 of register Ru1 are copied into bit position 0 of register R. (No bits are lost.)

Affected: (R), (Ru1), CC1, CC2

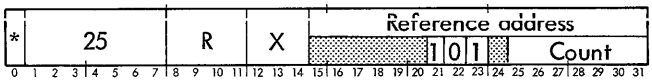
Arithmetic Shift, Single Register



If the shift count, C, is positive, the contents of register R are shifted left C places, with 0's copied into vacated bit positions on the right. (Bits shifted past R_0 are lost.) If C is negative, the contents of register R are shifted right |C| places, with the contents of bit position 0 copied into vacated bit positions on the left. (Bits shifted past R_{31} are lost.)

Affected: (R), CC1, CC2

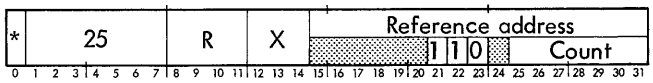
Arithmetic Shift, Double Register



If the shift count, C, is positive, the contents of register R and Ru1 are shifted left C places, with 0's copied into vacated bit positions on the right. Bits shifted past bit position 0 of register Ru1 are copied into bit position 31 of register R. (Bits shifted past R_0 are lost.) If C is negative, the contents of registers R and Ru1 are shifted right |C| places, with the contents of bit position 0 of register R copied into vacated bit positions on the left. Bits shifted past bit position 31 of register R are copied into bit position 0 of register Ru1. (Bits shifted past $Ru1_{31}$ are lost.)

Affected: (R), (Ru1), CC1, CC2

Searching Shift, Single Register



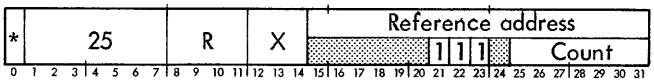
The searching shift is circular in either direction. If the shift count, C, is positive, the contents of register R are shifted left C bit positions or until a 1 appears in bit position 0. If C is negative, the contents are shifted right |C| positions or until a 1 appears in bit position 0. When the shift is terminated, the remaining count is stored in register 1, which is dedicated to the searching shift instruction.

Bits 0-24 of register 1 are cleared and the remaining count is loaded into bits 25-31. If the initial contents of bit 0 is equal to 1, then no bits are shifted by the instruction. In this case the original count in the instruction is stored in register 1.

Searching shift causing a change in bit position 0 causes CC2 to be set to 1. If bit position 0 is not changed during a searching shift, CC2 is cleared. CC4 is set to 1 if the shift is terminated with a 1 in bit position 0.

Affected: (R), (R1), CC2, CC4

Searching Shift, Double Register



The searching shift is circular in either direction. If the shift count, C, is positive, the contents of registers R and Ru1 are shifted left C bit positions or until a 1 appears in bit position 0 of register R. If C is negative, the contents are shifted right |C| positions or until a 1 appears in bit position 0. When the shift is terminated, the remaining count is stored in register 1, which is dedicated to the searching shift instruction. Bits 0-24 of register 1 are cleared and the remaining count is loaded into bits 25-31.

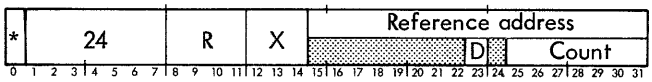
Searching shift causing a change in bit position 0 causes CC2 to be set to 1. If bit position 0 is not changed during a searching shift, CC2 is cleared. CC4 is set to 1 if the shift is terminated with a 1 in bit position 0.

Affected: (R), (Ru1), (R1), CC2, CC4

FLOATING-POINT SHIFT

Floating-point numbers are defined in the "Floating-Point Arithmetic Instructions" section. The format for the floating-point shift instruction is:

SF SHIFT FLOATING
(Word index alignment)



If direct addressing and no indexing is called for in the instruction SHIFT FLOATING, bit position 23 of the reference address field determines the type (long or short format) of shift, and bit positions 25-31 determine the direction and amount of the shift.

If indirect addressing and no indexing is called for in the instruction, bit positions 15-31 of the instruction are used to access the indirect word and then bit positions 23 and 25-31 of the indirect word determine the type, direction, and amount of the shift.

If direct addressing and indexing are called for in the instruction, bit 23 of the reference address (not affected by subsequent indexing) determines the type of shift. Bits 25-31 of the reference address plus bits 25-31 of the specified indexed register determine the direction and amount of the shift.

If indirect addressing and indexing are called for in the instruction, bits 15-31 of the reference address are used to access the indirect word. Bit 23 of the indirect word (not affected by subsequent indexing) determines the type of shift. Bits 25-31 of the indirect address plus bits 25-31 of the specified index register determine the direction and amount of the shift.

The shift count, C, in bit positions 25-31 of the effective virtual address determines the amount and direction of the shift. The shift count is treated as a 7-bit signed binary integer, with the high-order bit (bit position 25) as the sign (negative integers are represented in two's complement form).

The absolute value of the shift count determines the number of hexadecimal digit positions the floating-point number is to be shifted. If the shift count is positive, the floating-point number is shifted left; if the count is negative, the number is shifted right.

SHIFT FLOATING loads the floating-point number from the register(s) specified by the R field of the instruction into a set of internal registers. If the number is negative, it is two's complemented. A record of the original sign is retained. The floating-point number is then separated into a characteristic and a fraction, and CC1 and CC2 are both reset to 0's.

A positive shift count produces the following left shift operations:

1. If the fraction is normalized (i.e., is less than 1 and is equal to or greater than 1/16), or the fraction is all 0's, CC1 is set to 1.

2. If the fraction field is all 0's, the entire floating-point number is set to all 0's ("true" zero), regardless of the sign and the characteristic of the original number.

3. If the fraction is not normalized, the fraction field is shifted 1 hexadecimal digit position (4 bit positions) to the left and the characteristic field is decremented by 1. Vacated digit positions at the right of the fraction are filled with hexadecimal 0's.

   If the characteristic field underflows (i.e., is all 1's as the result of being decremented), CC2 is set to 1. However, if the characteristic field does not underflow, the shift process (shift fraction, and decrement characteristic) continues until the fraction is normalized, until the characteristic field underflows, or until the fraction is shifted left C hexadecimal digit positions, whichever occurs first. (Any two, or all three, of the terminating conditions can occur simultaneously.)

4. At the completion of the left shift operation, the floating-point result is loaded back into the general register(s). If the number was originally negative, the two's complement of the resultant number is loaded into the general register(s).

5. The condition code settings following a floating-point left shift are as follows:

1	2	3	4	Result
-	-	0	0	"True" zero (all 0's).
-	-	0	1	Negative.
-	-	1	0	Positive.
0	0	-	-	C digits shifted (fraction unnormalized, no characteristic underflow).
1	-	-	-	Fraction normalized (includes "true" zero).
-	1	-	-	Characteristic underflow.

A negative shift count produces the following right shift operations (again assuming that negative numbers are two's complemented before and after the shift operation):

1. The fraction field is shifted 1 hexadecimal digit position to the right and the characteristic field is incremented by 1. Vacated digit positions at the left are filled with hexadecimal 0's.

2. If the characteristic field overflows (i.e., is all 0's as the result of being incremented), CC2 is set to 1. However, if the characteristic field does not overflow, the shift process (shift fraction, and increment characteristic) continues until the characteristic field overflows or until the fraction is shifted right |C| hexadecimal digit positions, whichever occurs first. (Both terminating conditions can occur simultaneously.)

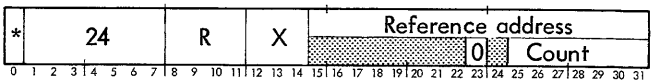
3. If the resultant fraction field is all 0's, the entire floating-point number is set to all 0's ("true" zero), regardless of the sign and the characteristic of the original number.

4. At the completion of the right shift operation, the floating-point result is loaded back into the general register(s). If the number was originally negative, the two's complement of the resultant number is loaded into the general register(s).

5. The condition code settings following a floating-point right shift are as follows:

1	2	3	4	Result
-	-	0	0	"True" zero (all zeros).
-	-	0	1	Negative.
-	-	1	0	Positive.
0	0	-	-	|C| digits shifted (no characteristic overflow).
0	1	-	-	Characteristic overflow.

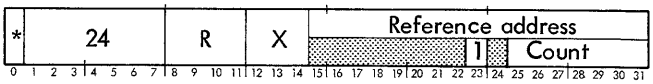
Floating Shift, Single Register



The short-format floating-point number in register R is shifted according to the rules established above for floating-point shift operations.

Affected: (R), CC

Floating Shift, Double Register



The long-format floating-point number in registers R and Ru1 is shifted according to the rules established above for floating-point shift operations. (If the R field of the instruction word is an odd value, a long-format floating-point number is generated by duplicating the contents of register R, and the 32 high-order bits of the result are loaded into register R.)

Affected: (R), (Ru1), CC

CONVERSION INSTRUCTIONS

The conversion instructions are:

Instruction Name	Mnemonic
Convert by Addition	CVA
Convert by Subtraction	CVS

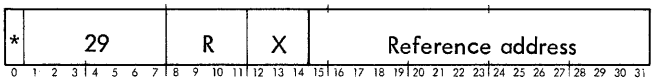
These two conversion instructions can be used to accomplish bidirectional translation between binary code and any other weighted binary code, such as BCD.

The effective addresses of the instructions CONVERT BY ADDITION and CONVERT BY SUBTRACTION each point to the starting location of a conversion table of 32 words, containing weighted values for each bit position of register Ru1. The 32 words of the conversion table are considered to be 32-bit positive quantities, and are referred

to as conversion values. The intermediate results of these instructions are accumulated in internal basic processor registers until the instruction is completed; the result is then loaded into the appropriate general register. Both instructions use a counter (n) that is set to 0 at the beginning of the instruction execution and is incremented by 1 with each iteration, until a total of 32 iterations has been performed.

If a memory parity or protection violation trap occurs during the execution of either instruction, the instruction sequence is aborted (without having changed the contents of register R or Ru1) and may be restarted (at the beginning of the instruction sequence) after the trap routine is processed.

CVA CONVERT BY ADDITION
(Word index alignment)



CONVERT BY ADDITION initially clears the internal A register and sets an internal counter (n) to 0. If bit position n of register Ru1 contains a 1, CVA adds the nth conversion value (contents of the word location pointed to by the effective address plus n) to the contents of the A register, accumulates the sum in the A register, and increments n by 1. If bit position n of register Ru1 contains a 0, CVA only increments n. If n is less than 32 after being incremented, the next bit position of register Ru1 is examined, and the addition process continues through n equal to 31; the result is then loaded into register R. If, on any iteration, the sum has exceeded the value $2^{32}-1$, CC1 is set to 1; otherwise, CC1 is reset to 0.

Affected: (R), CC1, CC3, CC4

$0 \longrightarrow A,\ 0 \longrightarrow n$

If $(Ru1)_n = 1$, then $(EWL + n) + (A) \longrightarrow A,\ n + 1 \longrightarrow n$

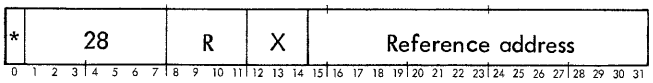
If $(Ru1)_n = 0$, then $n + 1 \longrightarrow n$

If $n < 32$, repeat; otherwise, $(A) \longrightarrow R$ and continue to next instruction.

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	Zero.
-	-	0	1	Bit 0 of register R is a 1.
-	-	1	0	Bit 0 of register R is a 0 and bit positions 1-31 of register R contain at least one 1.
0	-	-	-	Sum is correct (less than 2^{32}).
1	-	-	-	Sum is greater than $2^{32}-1$.

CVS CONVERT BY SUBTRACTION
(Word index alignment)



CONVERT BY SUBTRACTION loads the internal A register with the contents of register R, clears the internal B register, and sets an internal counter (n) to 0. All conversion values are considered to be 32-bit positive quantities. If the nth conversion value (the contents of the word location pointed to by the effective address plus n) is equal to or less than the current contents of the A register, CVS increments n by 1, adds the two's complement of the nth conversion value to the contents of the A register, stores the sum in the A register, and stores a 1 in bit position n of the B register. If the nth conversion value is greater than the current contents of the A register, CVS only increments n by 1. If n is less than 32 after being incremented, the next conversion value is compared and the process continues through n equal to 31; the remainder in the A register is loaded into register R, and the converted quantity in the B register is loaded into register Ru1.

Affected: (R), (Ru1), CC3, CC4

$(R) \longrightarrow A,\ 0 \longrightarrow B,\ 0 \longrightarrow n$

If $(EWL + n) \leq (A)$ then $A - (EWL + n) \longrightarrow A,$

$1 \longrightarrow B_n, n + 1 \longrightarrow n$

If $(EWL + n) > (A)$ then $n + 1 \longrightarrow n$

If $n < 32$, repeat; otherwise, $(A) \longrightarrow R,\ (B) \longrightarrow$ Ru1 and continue to the next instruction.

Condition code settings:

1	2	3	4	Result in Ru1
-	-	0	0	Zero.
-	-	0	1	Bit 0 of register Ru1 is a 1.
-	-	1	0	Bit 0 of register Ru1 is a 0 and bit positions 1-31 of register Ru1 contain at least one 1.

FLOATING-POINT ARITHMETIC INSTRUCTIONS

The floating-point arithmetic instructions are:

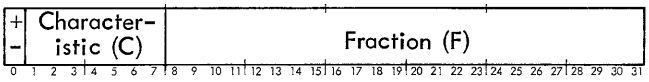
Instruction Name	Mnemonic
Floating Add Short	FAS
Floating Add Long	FAL
Floating Subtract Short	FSS

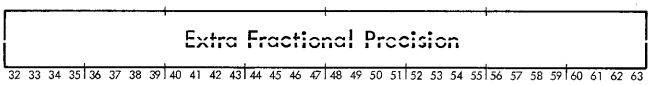
Instruction Name	Mnemonic
Floating Subtract Long	FSL
Floating Multiply Short	FMS
Floating Multiply Long	FML
Floating Divide Short	FDS
Floating Divide Long	FDL

FLOATING-POINT NUMBERS

Two number formats are accommodated for floating-point arithmetic: short and long. A short-format floating-point number consists of a sign (bit 0), a biased†, base 16 exponent, which is called a characteristic (bits 1-7), and a six-digit hexadecimal fraction (bits 8-31). A long-format floating-point number consists of a short-format floating-point number followed by an additional eight hexadecimal digits of fractional significance, and occupies a double-word memory location or an even-odd pair of general registers.

A floating-point number (N) has the following format:





A floating-point number (N) has the following formal definition:

1. $N = F \times 16^{C-64}$ where $F = 0$ or

   $16^{-6} \leq |F| \leq 1$ (short format) or

   $16^{-14} \leq |F| \leq 1$ (long format)

   and $0 \leq C \leq 127$.

2. A positive floating-point number with a fraction of zero and a characteristic of zero is a "true" zero. A positive floating-point number with a fraction of zero and a nonzero characteristic is an "abnormal" zero. For floating-point multiplication and division, an abnormal zero is treated as a true zero. However, for addition and subtraction, an abnormal zero is treated the same as any nonzero operand.

3. A positive floating-point number is normalized if and only if the fraction is contained in the interval

   $1/16 \leq F < 1$

4. A negative floating-point number is the two's complement of its positive representation.

5. A negative floating-point number is normalized if and only if its two's complement is a normalized positive number.

By this definition, a floating-point number of the form

1xxx xxxx 1111 0000 ... 0000

is normalized, and a floating-point number of the form

1xxx xxxx 0000 0000 ... 0000

is illegal and, whenever generated by floating-point instructions, is converted to the form

1yyy yyyy 1111 0000 ... 0000

where yy ... y is 1 less than xx ... x. Table 7 contains examples of floating-point numbers.

Modes of Operation

There are four mode control bits that are used to qualify floating-point operations. These mode control bits are identified as FR (floating round), FS (floating significance), FZ (floating zero), and FN (floating normalize); they are contained in bit positions 4, 5, 6, and 7, respectively, of the program status words ($PSWs_{4-7}$).

The floating-point mode is established by setting the four floating-point mode control bits. This can be performed by any of the following instructions:

Instruction Name	Mnemonic
Load Conditions and Floating Control	LCF
Load Conditions and Floating Control Immediate	LCFI
Load Program Status Words	LPSD
Exchange Program Status Words	XPSD

The floating-point mode control bits are stored by executing either of the following instructions:

Instruction Name	Mnemonic
Store Conditions and Floating Control	STCF
Exchange Program Status Words	XPSD

† The bias value of 40_{16} is added to the exponent for the purpose of making it possible to compare the absolute magnitude of two numbers, i.e., without reference to a sign bit. This manipulation effectively removes the sign bit, making each characteristic a 7-bit positive number.

Table 7. Floating-Point Number Representation

Decimal Number	Short Floating-Point Format ±	C		F						Hexadecimal Value	
$+(16^{+63})(1-2^{-24})$	0	111	1111	1111	1111	1111	1111	1111	1111	7F	FFFFFF
$+(16^{+3})(5/16)$	0	100	0011	0101	0000	0000	0000	0000	0000	43	500000
$+(16^{-3})(209/256)$	0	011	1101	1101	0001	0000	0000	0000	0000	3D	D10000
$+(16^{-63})(2047/4096)$	0	000	0001	0111	1111	1111	0000	0000	0000	01	7FF000
$+(16^{-64})(1/16)$	0	000	0000	0001	0000	0000	0000	0000	0000	00	100000
0 (called true zero)	0	000	0000	0000	0000	0000	0000	0000	0000	00	000000
$-(16^{-64})(1/16)$	1	111	1111	1111	0000	0000	0000	0000	0000	FF	F00000
$-(16^{-63})(2047/4096)$	1	111	1110	1000	0000	0001	0000	0000	0000	FE	801000
$-(16^{-3})(209/256)$	1	100	0010	0010	1111	0000	0000	0000	0000	C2	2F0000
$-(16^{+3})(5/16)$	1	011	1100	1011	0000	0000	0000	0000	0000	BC	B00000
$-(16^{+63})(1-2^{24})$	1	000	0000	0000	0000	0000	0000	0000	0001	80	000001
Special Case											
$-(16^{e})(1)$	1	$\overline{e}$		0000	0000	0000	0000	0000	0000		
is changed to											
$-(16^{e+1})(1/16)$	1	$\overline{e+1}$		1111	0000	0000	0000	0000	0000		
whenever generated as the result of a floating-point instruction.											

FLOATING-POINT ADD AND SUBTRACT

The floating round (FR), floating normalize (FN), floating zero (FZ), and floating significance (FS) mode control bits determine the operation of floating-point addition and subtraction (if characteristic overflow does not occur) as follows:

FR Floating round:

Note: The floating round facility is available only in the hardware floating-point. In the absence of this feature, the floating-point subroutines offer only truncation; hence, to guarantee hardware and software identical results, FR (bit 4 of PSWs) must be zero.

FR = 0 No rounding specified (truncation).

FR = 1 The results of additions and subtractions are to be rounded. Each value associated with the operation (i.e., augend, addend, and intermediate result of an add) is extended by the hardware to include one guard digit. (Short-format values are extended into bit positions 32-35 and long-format values are extended into bit positions 64-67.) Contents of guard digits may be affected during prealignment, computation, or postnormalization. Rounding is performed by evaluating the guard digit of the intermediate result after any required postnormalization. If the value of the guard digit is 0-7, the other digits are not modified. If the value of the guard digit is 8-F, the value contained within the other digits is incremented by one.

The following table shows the possible cases:

Postnormalization

Pre-alignment (exponents ≠)	Scale Answer Left	Scale Answer Right	Guard Digit Action
0	0	0	(Guard digit = 0.)
0	0	1	Round on guard digit. ①
0	1	0	Guard digit left shifted into low end of register. ②
0	1	1	Not possible.
1	0	0	Round on guard digit. ①
1	0	1	Round on guard digit. ①
1	1	0	Guard digit left shifted into low end of register. ②
1	1	1	Not possible.

Notes: ①Increment fraction if guard digit ≥ 8.

②Contents of guard digit become zero on first left shift.

Normally, there is no time penalty for the rounding operation. However, if the intermediate value is .FFFFFF and the guard digit is 8-F after postnormalization, a right alignment is done after rounding. (See the example below.)

Example

.FFFFFFF	(intermediate result before rounding)
1.000000	(result after rounding and truncating – not valid)
.100000	(result after postrounding alignment)

FN Floating normalize:

FN = 0 The results of additions and subtractions are to be postnormalized. If characteristic underflow occurs, if the result is zero, or if more than two postnormalization hexadecimal shifts are required, the settings for FZ and FS determine the resultant action. If none of the above conditions occurs, the condition code is set to 0010 if the result is positive, or to 0001 if the result is negative.

FN = 1 Inhibit postnormalization of the result of additions and subtractions. The settings of FZ and FS have no effect on the instruction operation. If the result is zero, the result is set to "true" zero and the condition code is set to 0000. If the result is positive, the condition code is set to 0010. If the result is negative, the condition code is set to 0001.

FZ Floating zero: (applies only if FN = 0)

FZ = 0 If the final result of an addition or subtraction operation cannot be expressed in normalized form because of the characteristic being reduced below zero, underflow has occurred, in which case the result is set equal to "true" zero and the condition code is set to 1100. (Exception: if a trap results from significance checking with FS = 1 and FZ = 0, an underflow generated in the process of postnormalizing is ignored.)

FZ = 1 Characteristic underflow causes the basic processor to trap to location X'44' with the contents of the general registers unchanged. If the result is positive, the condition code is set to 1110. If the result is negative, the condition code is set to 1101.

FS Floating significance: (applies only if FN = 0)

FS = 0 Inhibit significance trap. If the result of an addition or subtraction is zero, the result is set equal to "true" zero, the condition code is set to 1000, and the basic processor executes the next instruction in sequence. If more than two hexadecimal places of postnormalization shifting are required and characteristic underflow does not occur, the condition code is set to 1010 if the result is positive, or to 1001 if the result is negative; then, the basic processor executes the next instruction in sequence. (Exception: if characteristic underflow occurs with FS = 0, FZ determines the resultant action.)

FS = 1 The basic processor traps to location X'44' if more than two hexadecimal places of postnormalization shifting are required or if the result is zero. The condition code is set to 1000 if the result is zero, to 1010 if the result is positive, or to 1001 if the result is negative; however, the contents of the general registers are not changed. (Exception: if a trap results from characteristic underflow with FZ = 1, the results of significance testing are ignored.)

If characteristic overflow occurs, the basic processor always traps to location X'44' with the general registers unchanged and the condition code set to 0110 if the result is positive, or to 0101 if the result is negative.

FLOATING-POINT MULTIPLY AND DIVIDE

The floating round (FR) and floating zero (FZ) mode control bits determine the operation of floating-point multiplication and division (if characteristic overflow does not occur and division by zero is not attempted) as follows:

FR Floating round:

FR = 0 No rounding specified.

FR = 1 The results of floating multiplication and division instructions are to be rounded. For multiply or divide operations, a normalized product or quotient is produced, appended by a guard digit. This will be an absolute value.

Note: The example above (under "Floating-Point Add and Subtract") is not possible for multiply and divide. Therefore, there is never a time penalty for rounding.

FZ Floating zero:

FZ = 0 If the final result of a multiplication or division operation cannot be expressed in normalized form because of the characteristic being reduced below zero, underflow has occurred. If underflow occurs, the result is set equal to "true" zero and the condition code is set to 1100. If underflow does not occur, the condition code is set to 0010 if the result is positive, to 0001 if the result is negative, or to 0000 if the result is zero.

FZ = 1 Underflow causes the basic processor to trap to location X'44' with the contents of the general registers unchanged. The condition code is set to 1110 if the result is positive, or to 1101 if the result is negative. If underflow does not occur, the resultant action is the same as that for FZ = 0.

If the divisor is zero in a floating-point division, the basic processor always traps to location X'44' with the general registers unchanged and the condition code set to 0100. If characteristic overflow occurs, the basic processor always traps to location X'44' with the general registers unchanged and the condition code set to 0110 if the result is positive, or to 0101 if the result is negative.

CONDITION CODES FOR FLOATING-POINT INSTRUCTIONS

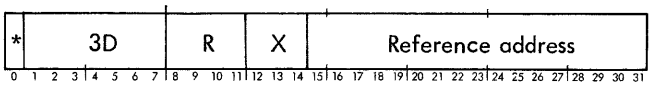
The condition code settings for floating-point instructions are summarized in Table 8. The following provisions apply to all floating-point instructions:

1. Undeflow and overflow detection apply to the final characteristic, not to any "intermediate" value.

2. If a floating-point operation results in a trap, the original contents of all general registers remain unchanged.

3. All shifting, truncation, and rounding are performed on absolute magnitudes. If the fraction is negative, then the two's complement is formed after shifting or truncation.

FAS FLOATING ADD SHORT
(Word index alignment)



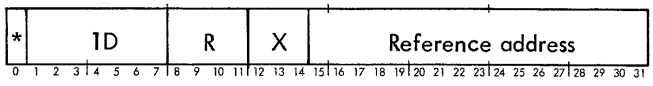
The effective word and the contents of register R are loaded into a set of internal registers and a low-order hexadecimal zero (guard digit) is appended to both fractions, extending them to seven hexadecimal digits each. FAS then forms the floating-point sum of the two numbers. (See "FR Floating round" under "Floating-Point Add and Subtract", if rounding applies.) If no floating-point arithmetic fault occurs, the sum is loaded into register R as a short-format floating-point number.

Affected: (R), CC

Trap: Floating-point arithmetic fault

(R) + EW ⟶ R

FAL FLOATING ADD LONG
(Doubleword index alignment)



The effective doubleword and contents of registers R and Ru1 are loaded into a set of internal registers.

The operation of FAL is identical to that of FLOATING ADD SHORT (FAS) except that the fractions to be added are each 14 hexadecimal digits long, guard digits are appended to the fractions only if rounding is specified, and R must be an even value for correct results. If no floating-point arithmetic fault occurs, the sum is loaded into registers R and Ru1 as a long-format floating-point number.

Affected: (R), (Ru1), CC

Trap: Floating-point arithmetic fault, instruction exception

(R, Ru1) + ED ⟶ R, Ru1

Table 8. Condition Code Settings for Floating-Point Instructions

Condition Code 1 2 3 4	Meaning If No Trap to Location X'44'	Meaning If Trap to Location X'44' Occurs
0 0 0 0	A x 0, 0/A, or -A + A① with FN=1 (Normal results)	*②
0 0 0 1	N < 0 (Normal results)	*
0 0 1 0	N > 0 (Normal results)	*
0 1 0 0	*②	Divide by zero (Always trapped)
0 1 0 1	*	Overflow, N < 0 (Always trapped)
0 1 1 0	*	Overflow, N > 0 (Always trapped)
③ 1 0 0 0	-A + A① (FS=0, FN=0, and no underflow)	-A + A
③ 1 0 0 1	N < 0 (> 2 Postnormalizing shifts; FS=0, FN=0, and no underflow)	N < 0 (> 2 Postnormalizing shifts; FS=1, FN=0, and no underflow with FZ=1)
③ 1 0 1 0	N > 0 (> 2 Postnormalizing shifts; FS=0, FN=0, and no underflow)	N > 0 (> 2 Postnormalizing shifts; FS=1, FN=0, and no underflow with FZ=1)
1 1 0 0	Underflow with FZ=0 and no trap by FS=1①	*
1 1 0 1	*	Underflow, N < 0 (FZ=1)
1 1 1 0	*	Underflow, N > 0 (FZ=1)

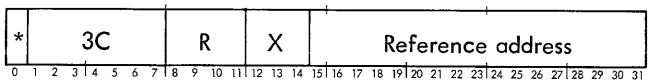
Notes: ① Result set to "true" zero

② "*" indicates impossible configurations

③ Applies to add and subtract only where FN=0

The R field of the FAL instruction must be an even value for proper operation of the instruction; if the R field of FAL is an odd value, the instruction traps to location X'4D', instruction exception trap.

FSS FLOATING SUBTRACT SHORT
(Word index alignment)



The effective word and the contents of register R are loaded into a set of internal registers.

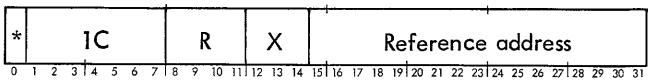
FLOATING SUBTRACT SHORT forms the two's complement of the effective word and then operates identically to FLOATING ADD SHORT (FAS). If no floating-point arithmetic fault occurs, the difference is loaded into register R as a short-format floating-point number.

Affected: (R), CC

Trap: Floating-point arithmetic fault

(R) - EW ⟶ R

FSL FLOATING SUBTRACT LONG
(Doubleword index alignment)



The effective doubleword and the contents of registers R and Ru1 are loaded into a set of internal registers.

FLOATING SUBTRACT LONG forms the two's complement of the effective doubleword and then operates identically to FLOATING ADD LONG (FAL). If no floating-point arithmetic fault occurs, the difference is loaded into registers R and Ru1 as a long-format floating-point number.

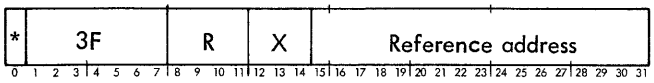
Affected: (R), (Ru1), CC

Trap: Floating-point arithmetic fault, instruction exception

(R, Ru1) - ED ⟶ R, Ru1

The R field of the FSL instruction must be an even value for proper operation of the instruction; if the R field of FSL is an odd value, the instruction traps to location X'4D', instruction exception trap.

FMS FLOATING MULTIPLY SHORT
(Word index alignment)

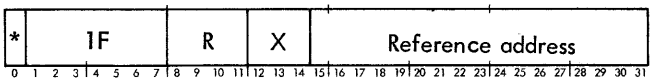


The effective word (multiplier) and the contents of register R (multiplicand) are loaded into a set of internal registers, and both numbers are then prenormalized (if necessary). A normalized 6-digit product is produced, appended by a guard digit. If FR equals 1, and the guard digit contains 8 or greater, the fraction is incremented. If no floating-point arithmetic fault occurs, the product is loaded into register R as a short-format floating-point number.

Affected: (R), CC
(R) x EW ⟶ R

Trap: Floating-point arithmetic fault

FML FLOATING MULTIPLY LONG
(Doubleword index alignment)



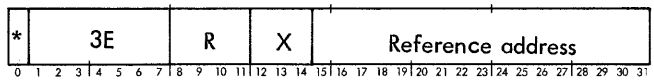
The effective doubleword (multiplier) and the contents of registers R and Ru1 (multiplicand) are loaded into a set of internal registers. (FLOATING MULTIPLY LONG then operates identically to FLOATING MULTIPLY SHORT (FMS), except that the operands are each 14 hexadecimal digits long. R must be an even value for correct results. If no floating-point arithmetic fault occurs, the product is loaded into registers R and Ru1 as a long-format floating-point number.

Affected: (R), (Ru1), CC
(R, Ru1) x ED ⟶ R, Ru1

Trap: Floating-point arithmetic fault, instruction exception

The R field of the FML instruction must be an even value for proper operation of the instruction; if the R field of FML is an odd value, the instruction traps to location X'4D', instruction exception trap.

FDS FLOATING DIVIDE SHORT
(Word index alignment)

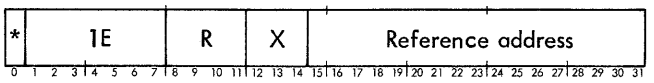


The effective word (divisor) and the contents of register R (dividend) are loaded into a set of internal registers and both numbers are then prenormalized (if necessary). A normalized 6-digit quotient is produced, appended by a guard digit. If FR equals 1, and the guard digit contains 8 or greater, the fraction is incremented. If no floating-point arithmetic fault occurs, the quotient is loaded into register R as a short-format floating-point number.

Affected: (R), CC
(R) ÷ EW ⟶ R

Trap: Floating-point arithmetic fault

FDL FLOATING DIVIDE LONG
(Doubleword index alignment)



The effective doubleword (divisor) and the contents of registers R and Ru1 (dividend) are loaded into a set of internal registers. FLOATING DIVIDE LONG then operates identically to FLOATING DIVIDE SHORT (FDS), except that the operands are each 14 hexadecimal digits long. R must be an even value for correct results. If no floating-point arithmetic fault occurs, the quotient is loaded into registers R and Ru1 as a long-format floating-point number.

Affected: (R), (Ru1), CC
(R, Ru1) ÷ ED ⟶ R, Ru1

Trap: Floating-point arithmetic fault, instruction exception

The R field of the FDL instruction must be an even value for proper operation of the instruction; if the R field of FDL is an odd value, the instruction traps to location X'4D' instruction exception trap.

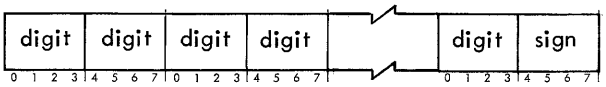
DECIMAL INSTRUCTIONS

The following instructions comprise the decimal instruction set:

Instruction Name	Mnemonic
Decimal Load	DL
Decimal Store	DST
Decimal Add	DA
Decimal Subtract	DS
Decimal Multiply	DM
Decimal Divide	DD
Decimal Compare	DC
Decimal Shift Arithmetic	DSA
Pack Decimal Digits	PACK
Unpack Decimal Digits	UNPK
Edit Byte String (described under "Byte-String Instructions")	EBS

PACKED DECIMAL NUMBERS

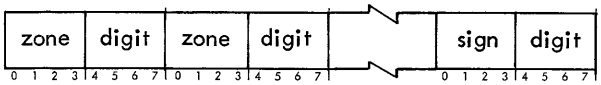
All decimal arithmetic instructions operate on packed decimal numbers, each consisting of from 1 to 31 decimal digits[†] (in absolute form) plus a decimal sign. A decimal digit is a 4-bit code in the range 0000 through 1001, where 0000 = 0, 0001 = 1, 0010 = 2, 0011 = 3, 0100 = 4, 0101 = 5, 0110 = 6, 0111 = 7, 1000 = 8, and 1001 = 9. A positive decimal sign is a 4-bit code of the form: 1010(X'A'), 1100(X'C'), 1110(X'E'), or 1111(X'F'). A negative decimal sign is a 4-bit code of the form: 1011(X'B'), or 1101(X'D'). However, the decimal sign codes generated for the result of a decimal instruction are: 1100 (X'C') for positive results, and 1101 (X'D') for negative results. The format of packed decimal numbers is:



For the decimal arithmetic instructions, a packed decimal number must occupy an integral number (1 through 16) of consecutive bytes. Thus, a decimal number must contain an odd number of decimal digits, the high-order digit (zero or nonzero) of the number must be in bit positions 0-3 of the first byte, the decimal sign must be in bit positions 4-7 of the last byte, and all decimal digits and the decimal sign must be 4-bit codes of the form described above.

ZONED DECIMAL NUMBERS

In zoned decimal format, a single decimal digit is contained within bit positions 4-7 of a byte, and bit positions 0-3 of the byte are referred to as the "zone" of the decimal digit. A zoned decimal number consists of from 1 to 31 bytes, with the decimal sign appearing as the zone for the last byte, as follows:



The sign format is EBCDIC and the zones are 1111.

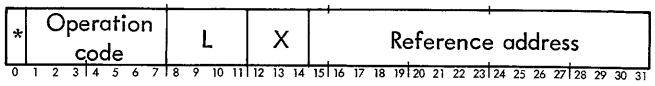
A decimal number can be converted from zoned to packed format by means of the instruction PACK DECIMAL DIGITS. A decimal number can be converted from packed to zoned format by means of the instruction UNPACK DECIMAL DIGITS.

DECIMAL ACCUMULATOR

All decimal arithmetic instructions imply the use of registers 12 through 15 of the current register block as the decimal accumulator, and registers 12 through 15 are treated as a single 16-byte register. The entire decimal accumulator is used in every decimal arithmetic instruction.

DECIMAL INSTRUCTION FORMAT

The general format of a decimal instruction is as follows:



The indirect address bit (position 0), the operation code (positions 1-7), the index field (12-14), and the reference address field (15-31) all have the same functions for the decimal instructions as they do for any other byte-addressing instruction. However, bit positions 8-11 of the instruction word do not refer to a general register; instead, the contents of this field (designated by the character "L") designate the length, in bytes, of a packed decimal number. (If L = 0, a length of 16 bytes is assumed.)

ILLEGAL DIGIT AND SIGN DETECTION

Prior to executing any decimal instruction, the basic processor checks all decimal operands for the presence of illegal decimal digits or illegal decimal signs. For all decimal arithmetic instructions, an illegal decimal digit is a sign code (i.e., in the range X'A' through X'F') that appears anywhere except in bit positions 4-7 of the least significant byte (the sign position) of the packed decimal number; an illegal decimal sign is a digit code (i.e., in the range X'0' through X'9') that appears in the sign position of the packed decimal number.

For the instructions DECIMAL MULTIPLY and DECIMAL DIVIDE, the illegal sign and digit check also includes a check for an illegal L field in the instruction. Illegal L fields are X'0' and the range X'9' to X'F'.

For the DECIMAL MULTIPLY instruction, only registers R14 and R15 are checked for illegal digits. The original contents of R12 and R13 are ignored and are presumed to be zeros.

If an illegal digit or sign is detected, the basic processor unconditionally aborts the execution of the instruction (at the time that the illegal digit or sign is detected), sets CC1 to 1 and resets CC2 to 0. If the decimal arithmetic fault trap mask (bit position 10 of the program status words) is a 0, the basic processor then executes the next instruction in sequence; however, if the decimal arithmetic fault trap mask is a 1, the basic processor traps to location X'45'. In either case, the contents of the decimal accumulator, the effective decimal operand, CC3, and CC4 remain unchanged.

[†]Except EDIT BYTE STRING (EBS), which has no limit on the size of numbers.

OVERFLOW DETECTION

Arithmetic overflow can occur during execution of the following decimal instructions:

DECIMAL ADD. Overflow occurs when the sum of the two decimal numbers exceeds the 31-digit capacity of the decimal accumulator ($+10^{31} - 1$ to $-10^{31} + 1$).

DECIMAL SUBTRACT. Overflow occurs when the difference between the two decimal numbers exceeds the 31-digit capacity of the decimal accumulator.

DECIMAL DIVIDE. Overflow occurs either when the divisor is zero, or when the dividend is greater than 14 digits in length and the absolute value of the significant digits to the left of the 15th digit position (counting from the right) is greater than or equal to the absolute value of the divisor.

If arithmetic overflow occurs during execution of DECIMAL ADD, DECIMAL SUBTRACT, or DECIMAL DIVIDE, the basic processor unconditionally aborts execution of the instruction (at the time of overflow detection), resets CC1 to 0, and sets CC2 to 1. Then, if the decimal arithmetic fault trap mask ($PSWs_{10}$) is a 1, the basic processor traps to location X'45'; if the decimal arithmetic fault trap mask is a 0, the basic processor executes the next instruction in sequence. In either case, the contents of the decimal accumulator, memory storage, CC3, and CC4 remain unchanged.

DECIMAL INSTRUCTION NOMENCLATURE

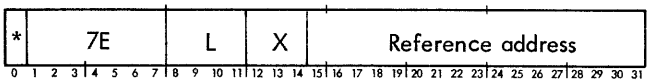
For the purpose of abbreviating the instruction descriptions to follow, the symbolic term "DECA" is used to represent the decimal accumulator, and the symbolic term "EDO" is used to represent the effective decimal operand of the instruction. For the instructions DECIMAL LOAD, DECIMAL ADD, DECIMAL SUBTRACT, DECIMAL MULTIPLY, DECIMAL DIVIDE, and DECIMAL COMPARE, the effective decimal operand is a packed decimal number that is "L" bytes in length, where L is the numeric value of bit positions 8-11 of the instruction word, and a value of 0 for L designates 16 bytes. The effective byte addresses of these instructions point to the byte location that contains the most significant byte (high-order digits) of the decimal number, and the effective byte address plus L-1 (where L = 0 = 16) points to the least significant byte (low-order digit and sign) of the decimal number. Thus, for these instructions, the effective decimal operand (EDO) is the contents of the byte string that begins with the effective byte location, is L bytes in length, and ends with the effective byte location plus L-1.

CONDITION CODE SETTINGS

All decimal instructions provide condition code settings, using CC1 to indicate whether or not an illegal digit or sign has been detected, and CC2 to indicate whether or not overflow has occurred. Most (but not all) of the decimal instructions provide condition code settings, using CC3 and CC4 to indicate whether the decimal number in the decimal accumulator is zero, negative, or positive, as follows:

CC3	CC4	Result in DECA
0	0	Zero – the decimal accumulator contains a positive or negative decimal sign code in the four low-order bit positions; the remainder of the decimal accumulator contains all 0's.
0	1	Negative – the decimal accumulator contains a negative decimal sign code in the four low-order bit positions; the remainder of the decimal accumulator contains at least one nonzero decimal digit.
1	0	Positive – the decimal accumulator contains a positive decimal sign code in the four low-order bit positions; the remainder of the decimal accumulator contains at least one nonzero decimal digit.

DL DECIMAL LOAD
(Byte index alignment)



If no illegal digit or sign is detected in the effective decimal operand, DECIMAL LOAD expands the effective decimal operand to 16 bytes (31 digits + sign) by appending high-order 0's, and then loads the expanded decimal number into the decimal accumulator. If the result in the decimal accumulator is zero, the converted sign remains unchanged.

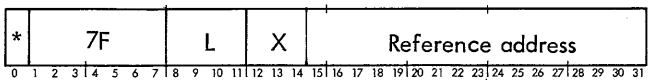
Affected: (DECA), CC Trap: Decimal arithmetic

(EBL to EBL + L - 1) ⟶ DECA

Condition code settings:

1	2	3	4	Result in DECA	
1	0	-	-	Illegal digit or sign detected, instruction aborted	
0	0	0	0	Zero	No illegal digit or illegal sign detected, instruction completed
0	0	0	1	Negative	
0	0	1	0	Positive	

DST DECIMAL STORE
(Byte index alignment)



If no illegal digit or sign is detected in the decimal accumulator, DECIMAL STORE stores the low-order L bytes of the decimal accumulator into memory from the effective byte location to the effective byte location plus L-1. If the decimal accumulator contains more significant information than is actually stored (i.e., at least one non-zero digit was not stored), CC2 is set to 1; otherwise, CC2 is reset to 0. If the result in memory is zero, the converted sign remains unchanged.

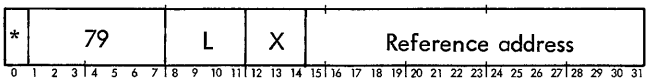
Affected: (EBL to EBL + L-1), CC1, CC2 Trap: Decimal arithmetic

(DECA) low-order bytes ⟶ EBL to EBL + L -1

Condition code settings:

1	2	3	4	Result of DST	
1	0	-	-	Illegal digit or sign detected, instruction aborted	
0	0	-	-	All significant information stored	No illegal digit or illegal sign detected, instruction completed
0	1	-	-	Some significant information not stored	

DA DECIMAL ADD
(Byte index alignment)



If no illegal digit or sign is detected in the effective decimal operand or in the decimal accumulator, DECIMAL ADD algebraically adds the decimal number to the contents of the decimal accumulator. If the result in the decimal accumulator is zero, the resulting sign is forced to the positive form.

Overflow occurs if the sum exceeds the capacity of the decimal accumulator (i.e., if the absolute value of the sum is equal to or greater than 10^{31}), in which case CC1 is reset to 0, CC2 is set to 1, and the instruction aborted with the previous contents of the decimal accumulator, CC3 and CC4 unchanged.

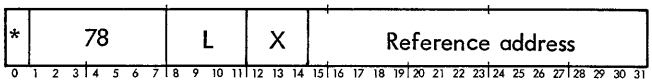
Affected: (DECA), CC Trap: Decimal arithmetic

(DECA) + EDO ⟶ DECA

Condition code settings:

1	2	3	4	Result in DECA	
1	0	-	-	Illegal digit or sign detected	Instruction aborted
0	1	-	-	Overflow	
0	0	0	0	Zero	No illegal digit or sign detected, no overflow, instruction completed
0	0	0	1	Negative	
0	0	1	0	Positive	

DS DECIMAL SUBTRACT
(Byte index alignment)



If no illegal digit or sign is detected in the effective decimal operand or in the decimal accumulator, DECIMAL SUBTRACT algebraically subtracts the decimal number from the contents of the decimal accumulator, and then loads the difference into the decimal accumulator. If the result in the decimal accumulator is zero, the resulting sign is forced to the positive form.

Overflow occurs if the difference exceeds the capacity of the decimal accumulator (i.e., if the absolute value of the difference is equal to or greater than 10^{31}), in which case CC1 is reset to 0, CC2 is set to 1, and the instruction is aborted with the contents of the previous decimal accumulator, CC3 and CC4 unchanged.

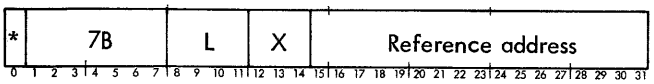
Affected: (DECA), CC Trap: Decimal arithmetic

(DECA) – EDO ⟶ DECA

Condition code settings:

1	2	3	4	Result in DECA	
1	0	-	-	Illegal digit or sign detected	Instruction aborted
0	1	-	-	Overflow	
0	0	0	0	Zero	No illegal digit or sign detected, no overflow, instruction completed
0	0	0	1	Negative	
0	0	1	0	Positive	

DM DECIMAL MULTIPLY
(Byte index alignment, continue after interrupt)



If no illegal digit or sign is detected in the effective decimal operand or decimal accumulator, DECIMAL MULTIPLY multiplies the effective decimal operand (multiplicand) by the contents of the decimal accumulator registers R14 and R15 (multiplier) and then loads the product into the entire decimal accumulator. If the result in the decimal accumulator is zero, the resulting sign is forced to the positive form.

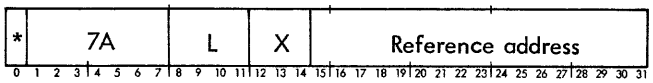
Affected: (DECA), CC Trap: Decimal arithmetic

(DECA) x EDO ⟶ DECA

Condition code settings:

1	2	3	4	Result in DECA	
1	0	-	-	Illegal digit or sign detected, instruction aborted	
0	0	0	0	Zero	No illegal digit or sign detected, instruction completed
0	0	0	1	Negative	
0	0	1	0	Positive	

DD DECIMAL DIVIDE
(Byte index alignment, continue after interrupt)



If there is no illegal digit or sign in the effective decimal operand and if there is at least one decimal sign in the decimal accumulator, DECIMAL DIVIDE divides the contents of the decimal accumulator (dividend) by the effective decimal operand (divisor). Then, if no overflow has occurred, the basic processor loads the quotient (15 decimal digits plus sign) into the eight low-order bytes of the decimal accumulator (registers 14 and 15), and loads the remainder (also 15 decimal digits plus sign) into the eight high-order bytes of the decimal accumulator (registers 12 and 13). The sign of the remainder is the same as that of the original dividend. If the quotient is zero, the sign of the quotient is forced to the positive form.

Overflow occurs if any of the following conditions are not satisfied before the initial execution of DECIMAL DIVIDE:

1. The divisor must not be zero.
2. If the length of the dividend is greater than 15 decimal digits, the absolute value of the significant digits to the left of the 15th digit position (i.e., those digits in registers 12 and 13) must be less than the absolute value of the divisor.

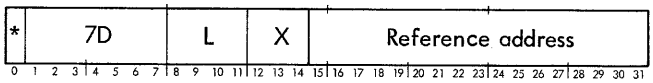
Affected: (DECA), CC Trap: Decimal arithmetic

(DECA) ÷ EDO ⟶ DECA

Condition code settings:

1	2	3	4	Result in DECA	
1	0	-	-	Illegal digit or sign detected	Instruction aborted
0	1	-	-	Overflow	
0	0	0	0	Zero quotient	No illegal digit or sign detected, no overflow, instruction completed
0	0	0	1	Negative quotient	
0	0	1	0	Positive quotient	

DC DECIMAL COMPARE
(Byte index alignment)



If there is no illegal digit or sign in the effective decimal operand or in the decimal accumulator, DECIMAL COMPARE expands the effective decimal operand to 16 bytes (31 digits plus sign) by appending high-order 0's, algebraically compares the expanded decimal number to the contents of the entire decimal accumulator, and sets CC3 and CC4 according to the result of the comparison (a positive zero compares equal to a negative zero).

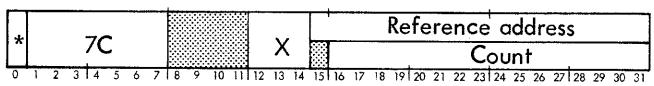
Affected: CC Trap: Decimal arithmetic

(DECA) : EDO

Condition code settings:

1	2	3	4	Result of comparison	
1	0	-	-	Illegal digit or sign detected, instruction aborted	
0	0	0	0	(DECA) equals EDO	No illegal digit or sign detected, instruction completed
0	0	0	1	(DECA) less than EDO	
0	0	1	0	(DECA) greater than EDO	

DSA DECIMAL SHIFT ARITHMETIC
(Byte index alignment)



If no illegal digit or sign is detected in the decimal accumulator, DECIMAL SHIFT ARITHMETIC arithmetically shifts the contents of the decimal accumulator (excluding the decimal sign), with the direction and amount of the shift determined by the effective virtual address of the instruction. If the result in the decimal accumulator is zero, the resulting sign remains unchanged.

If no indirect addressing or indexing is used with DSA, the shift count C is the contents of bit positions 16-31 of the instruction word. If only indirect addressing is used with DSA, the shift count is the contents of bit positions 16-31 of the word pointed to by the indirect address in the instruction word. If indexing only is used with DSA, the shift count is the contents of bit positions 16-31 of the instruction word plus the contents of bit positions 14-29 of the designated index register (bits 0-13, 30, and 31 of the index are ignored). If indirect addressing and indexing are both used with DSA, the shift count is the sum of the contents of bit positions 16-31 of the word pointed to by the indirect address and the contents of bit positions 14-29 of the designated index register.

The shift count, C, is treated as a 16-bit signed binary integer, with negative integers in two's complement form. If the shift count is positive, the contents of the decimal accumulator are shifted left C decimal digit positions; if the shift count is negative, the contents of the decimal accumulator are shifted right -C decimal digit positions. In either case, the decimal sign is not shifted, vacated decimal digit positions are filled with 0's, and any digits shifted out of the decimal accumulator are lost. Although the range of possible values for C is $2^{-15} \leq C \leq 215-1$. a shift count greater than +31 or less than -31 is interpreted as a shift count of exactly +31 or -31.

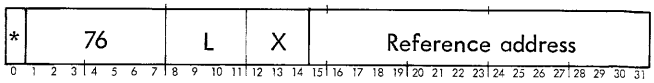
If any nonzero decimal digit is shifted out of the decimal accumulator during a left shift, CC2 is set to 1; otherwise, CC2 is reset to 0. CC2 is unconditionally reset to 0 at the completion of a right shift.

Affected: (DECA), CC Trap: Decimal arithmetic

Condition code settings:

1	2	3	4	Result in DECA	
1	0	-	-	Illegal digit or sign detected, instruction aborted	
0	-	0	0	Zero	No illegal digit or sign detected, instruction completed
0	-	0	1	Negative	
0	-	1	0	Positive	
0	0	-	-	Right shift or no nonzero digit shifted out of DECA on left shift	
0	1	-	-	One or more nonzero digit(s) shifted out of DECA on left shift	

PACK PACK DECIMAL DIGITS
(Byte index alignment)



PACK DECIMAL DIGITS converts the effective decimal operand (assumed to be in zoned format) into a packed decimal number and, if necessary, appends sufficient high-order 0's to produce a decimal number that is 16 bytes (31 decimal digits plus sign) in length. The zone (bits 0-3) of the low-order digit of the effective decimal operand is used to select the sign code for the packed decimal number; all other zones are ignored in formatting the packed decimal number. If no illegal digit or sign appears in the packed decimal number, it is then loaded into the decimal accumulator. If the result in the decimal accumulator is zero, the resulting sign remains unchanged.

The L field of this instruction specifies the length, in bytes, of the resultant packed decimal number in the decimal accumulator; therefore, the length of the effective decimal operand is 2L-1 bytes (where L = 0 implies a length of 31 bytes for the effective decimal operand).

Affected: (DECA), CC Trap: Decimal arithmetic

packed (EBL to EBL + 2L - 2) ⟶ DECA

Condition code settings:

1	2	3	4	Result in DECA	
1	0	-	-	Illegal digit or sign detected, instruction aborted	
0	0	0	0	Zero	No illegal digit or sign detected, instruction completed
0	0	0	1	Negative	
0	0	1	0	Positive	

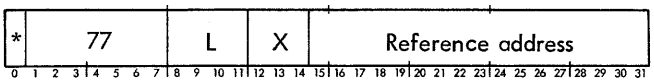
Example 1, L = 6:

		Before execution	After execution
EDO	=	X'F0F1F2F3 F4F5F6F7 F8F9F0'	X'F0F1F2F3 F4F5F6F7 F8F9F0'
(DECA)	=	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx	X'00000000 00000000 00000123 4567890C'
CC	=	xxxx	0010

Example 2, L = 6:

		Before execution	After execution
EDO	=	X'000938F7 E655B483 02F1B0'	X'000938F7 E655B483 02F1B0'
(DECA)	=	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx	X'00000000 00000000 00000987 6543210D'
CC	=	xxxx	0001

UNPK UNPACK DECIMAL DIGITS
(Byte index alignment, continue after interrupt)



If no illegal digit or sign is detected in the decimal accumulator (assumed to be in packed decimal format), UNPACK DECIMAL DIGITS converts the contents of the low-order L bytes of the decimal accumulator to zoned decimal format and stores the result, as a byte string, from the effective byte location to the effective byte location plus 2L-2. The contents of the four low-order bit positions of the decimal accumulator are used to select the sign code for the last digit of the string; for all other digits, the zones are 1111 (X'F'). The contents of the decimal accumulator remain unchanged, and only 2L-1 bytes of memory are altered. If the decimal accumulator contains more significant information than is actually unpacked and stored, CC2 is set to 1; otherwise, CC2 is reset to 0. If the result in memory is zero, the resulting sign remains unchanged.

Affected: (EBL to EBL+2L-2), CC1, CC2 Trap: Decimal arithmetic

zoned (DECA) ⟶ EBL to EBL + 2L -2

Condition code settings:

1	2	3	4	Result of UNPK	
1	0	-	-	Illegal digit or sign detected, instruction aborted	
0	0	-	-	All significant information zoned and stored	No illegal digit or sign detected, instruction completed
0	1	-	-	Some significant information not zoned and stored	

Example 1, L = 10:

		Before execution	After execution
(DECA)	=	X'00000000 00000001 23456789 0123456D'	X'00000000 00000001 23456789 0123456D'
EDO	=	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxx	X'F0F0F0F1 F2F3F4F5 F6F7F8F9 F0F1F2F3 F4F5D6'
CC	=	xxxx	00xx

Example 2, L = 8:

		Before execution	After execution
(DECA)	=	X'00000000 23000000 10001234 0012345C'	X'00000000 23000000 10001234 0012345C'
EDO	=	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxx	X'F1F0F0F0 F1F2F3F4 F0F0F1F2 F3F4C5'
CC	=	xxxx	01xx

Example 3, L = 4:

		Before execution	After execution
(DECA)	=	X'00001001 00001002 00001003 0001004F'	X'00001001 00001002 00001003 0001004F'
EDO	=	xxxxxxxx xxxxxxxx	X'F0F0F0F1 F0F0C4'
CC	=	xxxx	01xx

BYTE-STRING INSTRUCTIONS

Five instructions provide for the manipulation of strings of consecutive bytes. The byte-string instructions and their mnemonic codes are as follows:

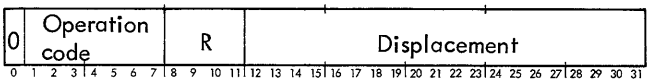
Instruction Name	Mnemonic
Move Byte String	MBS
Compare Byte String	CBS

Instruction Name	Mnemonic
Translate Byte String	TBS
Translate and Test Byte String	TTBS
Edit Byte String	EBS

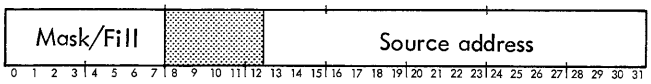
These instructions are in the immediate byte operand class and are memory-to-memory operations. These operations are under the control of information that must be loaded into certain general registers before the instruction is executed. Except for the MOVE BYTE STRING instruction, which proceeds four bytes at a time under certain conditions, a byte string instruction proceeds one byte at a time and may be interrupted after any individual byte operation. Upon return, execution resumes from the point of interruption.

The general format for the information in the instruction word and in the general registers is as follows:

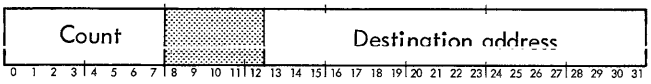
Instruction word:



Contents of register R:



Contents of register Ru1:



Designation	Function
Operation	The 7-bit operation code of the instruction. (If any byte-string instruction is indirectly addressed, the basic processor traps to location X'40' at the time of operation code decoding.)
R	The 4-bit field that identifies register R of the current general register block.
Displacement	A 20-bit field that contains a signed byte displacement value, used to form an effective byte address. The displacement value is right-justified in the 20-bit field, and negative values are in two's complement form.
Mask/Fill	An 8-bit field used only with TRANSLATE AND TEST BYTE STRING and EDIT BYTE STRING. The purpose of this field is explained in the detailed discussion of the TTBS and EBS instructions.
Source Address	A 19-bit field that normally contains the byte address of the first (most significant) byte of the source byte string operand. The effective source address is the source address in register R plus the displacement value in the instruction word.
Count	An 8-bit field that contains the true count (from 0 to 255) of the number of bytes involved in the operation. This field is decremented by 1 as each byte in the destination byte string is processed. A 0 count means "no operation" with respect to the registers and main memory.
Destination Address	A 19-bit field that contains the byte address of the first (most significant) byte of the destination byte-string operand. This field is incremented by 1 as each byte in the destination byte string is processed.

In any byte-string instruction, any portion of register R or Ru1 that is not explicitly defined (i.e., bit positions 8-12), should be coded with zeros for real and virtual addressing.

Since the value Ru1 is obtained by performing a logical inclusive OR with the value 0001 and the value of the R field of the instruction word, the two control registers are R and R+1 if R is even. However, if R is an odd value, register R contains an address value that functions both as a source operand address and as a destination operand address. Also, if register 0 is designated in any byte-string instruction (except for TRANSLATE AND TEST BYTE STRING and EDIT BYTE STRING), its contents are ignored and a zero source address value is obtained. Thus, the following three cases exist for most byte-string instructions, depending on whether the value of the R field of the instruction word is even and nonzero, odd, or zero:

Case I, R is even and nonzero

The effective source address is the address in register R plus the displacement in the instruction word; the destination address is the address in register R + 1, but without the displacement added.

Case II, R is odd

The effective source address is the address in register R plus the displacement in the instruction word; the destination address is also the address in register R, but without the displacement added.

Case III, R is zero

The effective source address is the displacement value in the instruction word; the destination address is the address in register 1. In this case, the source byte-string operand is always a single byte.

In the descriptions of the byte-string instructions, the following abbreviations and terms are used:

D — Displacement, $(I)_{12-31}$.

SA — Source address, $(R)_{13-31}$

ESA — Effective source address, $\left[(R)_{13-31}+(I)_{12-31}\right]_{13-31}$

The contents of bit positions 13-31 of register R are added (right aligned) to the contents of bit positions 12-31 of the instruction word; the 19 low-order bits of the result are used as the effective source address.

C — Count, $(Ru1)_{0-7}$

DA — Destination address, $(Ru1)_{13-31}$

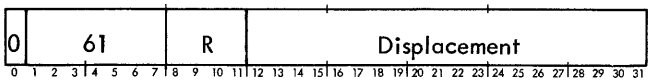
SBS — Source byte string, the byte string that begins with the byte location pointed to by the 19-bit effective source address and is C bytes in length (if R is 0).

DBS — Destination byte string, the byte string that begins with the byte location pointed to by the destination address and is always C bytes in length.

TRAPS BY BYTE-STRING INSTRUCTIONS

Byte-string instructions cause a trap if either of the addressed byte strings come from memory pages that are protected by either access protection or write locks. A trap also occurs if either byte string is fully or partly contained within memory pages that are physically not present. A check for these access trap conditions is made prior to initiation of any byte relocation or general register change. These tests are performed for MOVE BYTE STRING and TRANSLATE BYTE STRING. The source and destination locations are tested for MOVE BYTE STRING; only the destination location is tested for TRANSLATE BYTE STRING, since there is no assurance that the translate table will be accessed in its entirety in the course of execution. If an access protection violation were to occur in trying to reach a byte in the translate table or decimal digit strings during the course of execution, then the instruction would trap and result in a partially executed condition. However, if the destination byte string does overlap the translation table, the registers would be restored in such a manner that the instruction could be restarted after the protection violation had been corrected. When a trap occurs resulting in a partially executed instruction, the Register Altered indicator will be set.

MBS MOVE BYTE STRING
(Immediate Displacement, continue after interrupt)



MOVE BYTE STRING copies the contents of the source byte string (left to right) into the destination byte string. The previous contents of the destination byte string are destroyed, but the contents of the source byte string are not affected unless the destination byte string overlaps the source byte string.

When the destination byte string overlaps the source byte string, the resulting destination byte string contains one or more repetitions of bytes from the source byte string. Thus, if a destination byte string of C bytes begins with the kth byte of a source byte string (numbering from 1), the first k-1 bytes of the source byte string are duplicated in the destination byte string x number of times, where $x = C/(k-1)$. For example, if the destination byte string begins with the second byte of the source byte string, the first byte of the source byte string is duplicated throughout the destination byte string.

If both byte strings begin with the same byte (i.e., k = 1) and the R field of MBS is nonzero, the destination byte string is read and replaced into the same memory locations. However, if both byte strings begin with the same byte and the R field of MBS is zero, the first byte of the byte string is duplicated throughout the remainder of the byte string (see "Case III", below).

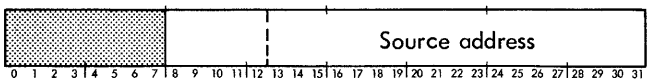
Affected: (DBS), (R), (Ru1)

(SBS) ⟶ DBS

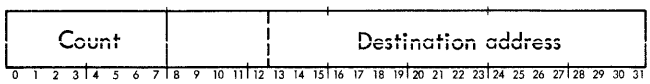
If MBS is indirectly addressed, it is treated as a nonexistent instruction. The basic processor unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the destination byte string unchanged. See "Traps by Byte String Instructions" (in this section) for other trap conditions.

Case I, even, nonzero R field (Ru1=R+1)

Contents of register R:



Contents of register R+1:

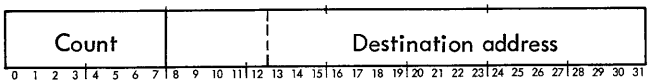


The source byte string begins with the byte location pointed to by the source address in register R plus the displacement in MBS; the destination byte string begins with the byte location pointed to by the destination address in register R+1.

Both byte strings are C bytes in length. When the instruction is completed, the destination and source addresses are each incremented by C, and C is set to zero.

Case II, odd R field (Ru1=R)

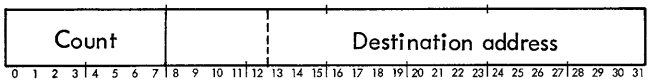
Contents of register R:



The source byte string begins with the byte location pointed to by the address in register R plus the displacement in MBS; the destination byte string begins with the byte location pointed to by the destination address in register R. Both byte strings are C bytes in length. When the instruction is completed, the destination address is incremented by C, and C is set to zero.

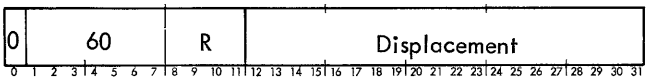
Case III, zero R field (Ru1=1)

Contents of register 1:



The source byte string consists of a single byte, the contents of the byte location pointed to by the displacement in MBS; the destination byte string begins with the byte location pointed to by the destination address in register 1 and is C bytes in length. In this case, the source byte is duplicated throughout the destination byte string. When the instruction is completed, the destination address is incremented by C, and C is set to zero.

CBS COMPARE BYTE STRING
(Immediate displacement, continue after interrupt)



COMPARE BYTE STRING compares, as magnitudes, the contents of the source byte string with the contents of the destination byte string, byte by corresponding byte, beginning with the first byte of each string. The comparison continues until the specified number of bytes have been compared or until an inequality is found. When CBS is terminated, CC3 and CC4 are set to indicate the result of the last comparison. If the CBS instruction terminates due to inequality, the count in register Ru1 is one greater than the number of bytes remaining to be compared; the source address in register R and the destination address in register Ru1 indicate the locations of the unequal bytes.

Affected: (R), (Ru1), CC3, CC4

(SBS) : (DBS)

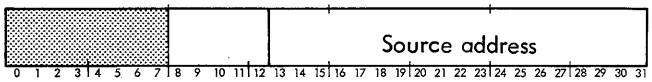
Condition code settings:

1	2	3	4	Result of CBS
-	-	0	0	Source byte string equals destination byte string or initial byte count is equal to zero.
-	-	0	1	Source byte string less than destination byte string.
-	-	1	0	Source byte string greater than destination byte string.

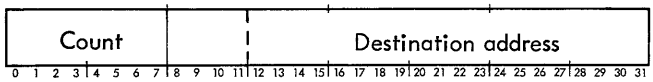
If CBS is indirectly addressed, it is treated as a nonexistent instruction. The basic processor unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the destination byte string unchanged. See "Traps By Byte String Instructions" (in this section) for other trap conditions.

Case I, even, nonzero R field (Ru1=R+1)

Contents of register R:



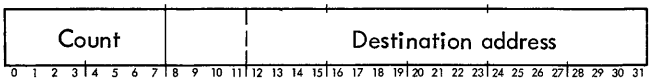
Contents of register R+1:



The source byte string begins with the byte location pointed to by the source address in register R plus the displacement in CBS; the destination byte string begins with the byte location pointed to by the destination address in register R+1. Both byte strings are C bytes in length.

Case II, odd R field (Ru1=R)

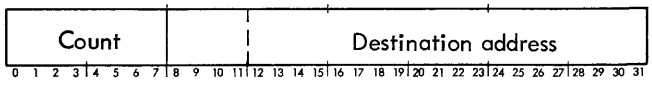
Contents of register R:



The source byte string begins with the byte location pointed to by the address in register R plus the displacement in CBS; the destination byte string begins with the byte location pointed to by the destination address in register R. Both byte strings are C bytes in length.

Case III, zero R field (Ru1=1)

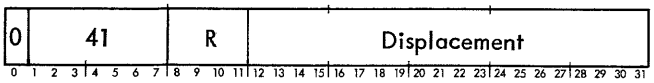
Contents of register 1:



The source byte string consists of a single byte, the contents of the location pointed to by the displacement in CBS;

the destination byte string begins with the byte location pointed to by the destination address in register 1 and is C bytes in length. In this case, the source byte is compared with each byte of the destination byte string until an inequality is found.

TBS TRANSLATE BYTE STRING
(Immediate displacement, continue after interrupt)



TRANSLATE BYTE STRING replaces each byte of the destination byte string with a source byte located in a translation table. The destination byte string begins with the byte location pointed to by the destination address in register Ru1, and is C bytes in length. The translation table consists of up to 256 consecutive byte locations, with the first byte location of the table pointed to by the displacement in TBS plus the source address in register R. A source byte is defined as that which is in the byte location pointed to by the 19 low-order bits of the sum of the following values.

1. The displacement in bit positions 12-31 of the TBS instruction.

2. The current contents of bit positions 13-31 of register R (source address).

3. The numeric value of the current destination byte, the 8-bit contents of the byte location pointed to by the current destination address in bit positions 13-31 of register (Ru1).

Affected: (DBS), (Ru1) Trap: Instruction exception

translated (DBS) ⟶ DBS

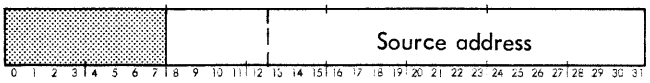
The R field of the TBS instruction must be an even value for proper operation of the instruction; if the R field of TBS is an odd value, the instruction traps to location X'4D', instruction exception trap.

If TBS is indirectly addressed, it is treated as a nonexistent instruction. The basic processor unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the destination byte string unchanged.

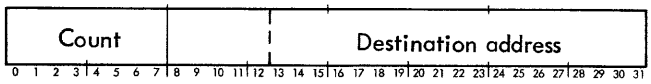
See "Traps By Byte String Instructions" (in this section) for other trap conditions. Note that the check for access trap conditions is done only for the source byte string.

Case I, even, nonzero R field (Ru1=R+1)

Contents of register R:



Contents of register R+1:



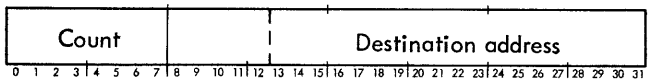
The destination byte string begins with the byte location pointed to by the destination address in register R + 1 and is C bytes in length. The source byte string (translation table) begins with the byte location pointed to by the displacement in TBS plus the source address in register R. When the instruction is completed, the destination address is incremented by C, C is set to zero, and the source address remains unchanged.

Case II, odd R field (Ru1=R)

Because of the interruptible nature of TRANSLATE BYTE STRING, the instruction traps with the contents of register R unchanged when an odd-numbered general register is specified by the R field of the instruction word.

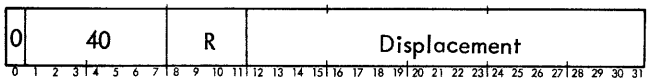
Case III, zero R field (Ru1=1)

Contents of register 1:



The destination byte string begins with the byte location pointed to by the destination address in register 1 and is C bytes in length. The source byte string (translation table) begins with the location pointed to by the displacement in TBS. When the instruction is completed, the destination address is incremented by C and C is set to zero.

TTBS TRANSLATE AND TEST BYTE STRING
(Immediate displacement, continue after interrupt)



TRANSLATE AND TEST BYTE STRING compares the mask in bit positions 0-7 of register R with source bytes in a byte translation table. The destination byte string begins with the byte location pointed to by the destination address in register Ru1, and is C bytes in length. The byte translation table and the translation bytes themselves are identical to that described for the instruction TRANSLATE BYTE STRING. The destination byte string is examined (without being changed) until a translation byte (source byte) is found that contains a 1 in any of the bit positions selected by a 1 in the mask. When such a translation byte is found, TTBS replaces the mask with the logical product (AND) of the translation byte and the mask, and terminates with CC4 set to 1.

If the TTBS instruction terminates due to the above condition, the count (C) in register Ru1 is one greater than the number of bytes remaining to be compared and the destination address in register Ru1 indicates the location

of the destination byte that caused the instruction to terminate. If no translation byte is found that satisfies the above condition after the specified number of destination bytes have been compared, TTBS terminates with CC4 reset to 0. In no case does the TTBS instruction change the source byte string.

Affected: (R), (Ru1), CC4 Trap: Instruction exception

If translated (SBS) n mask $\neq 0$, translated (SBS) n mask ⟶ mask and stop

If translated (SBS) n mask = 0, continue

Condition code settings:

1	2	3	4	Result of TTBS
-	-	-	0	Translation bytes and the mask do not compare ones any place.
-	-	-	1	The last translation byte compared with the mask contained at least one 1 corresponding to a 1 in the mask.

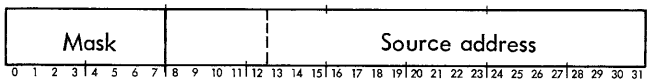
The R field of the TTBS instruction must be an even value for proper operation of the instruction; if the R field of TTBS is an odd value, the instruction traps to location X'4D', instruction exception trap.

If TTBS is indirectly addressed, it is treated as a nonexistent instruction. The basic processor unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the destination byte string unchanged.

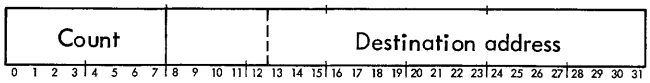
See "Traps By Byte String Instructions" (in this section) for other trap conditions. Note that the check for access trap conditions is done only for the source byte string.

Case I, even, nonzero R field (Ru1=R+1)

Contents of register R:



Contents of register R+1:



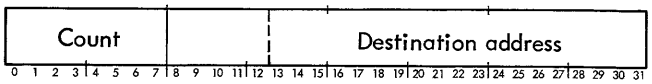
The destination byte string begins with the byte location pointed to by the destination address in register R + 1 and is C bytes in length. The source byte string (translation table) begins with the byte location pointed to by the displacement in TTBS plus the source address in register R.

Case II, odd R field

Because of the interruptible nature of TRANSLATE AND TEST BYTE STRING the instruction traps with the contents of register R unchanged when an odd-numbered general register is specified by the R field of the instruction word.

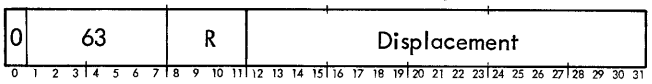
Case III, zero R field (Ru1=1)

Contents of register 1:



The destination byte string begins with the byte location pointed to by the destination address in register 1 and is C bytes in length. The source byte string (translation table) begins with the location pointed to by the displacement in TTBS. In this case, the instruction automatically provides a mask of eight 1's. (This is an exception to the general rule, used in the other byte-string instructions, the register 0 provides all 0's as its contents.)

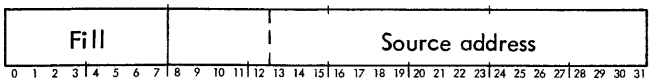
EBS EDIT BYTE STRING
(Immediate displacement, continue after interrupt)



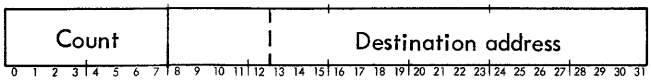
EDIT BYTE STRING converts a decimal information field from packed decimal format to zoned decimal EBCDIC format, under control of the editing pattern in the destination byte string, and replaces the destination byte string with the edited, zoned result. (See "Decimal Instructions", "Packed Decimal Numbers", and "Zoned Decimal Numbers" for a description of packed and zoned decimal formats.) EBS proceeds one byte at a time, starting with the first (most significant) byte of the editing pattern, and continues until all bytes in the editing pattern have been processed. The fill character, contained in bit position 0-7 of register R, replaces the pattern byte under specified conditions. More than one decimal number field can be edited by a single EBS instruction if the pattern in memory is, in fact, a series of patterns corresponding to a series of number fields. In such cases, however, after the EBS instruction is completed, the condition code indicates the result of the last decimal number field processed and register 1 contains the byte address (or the byte address plus 1) of the last significance indicator in the edited destination byte string. (This allows the insertion of a floating dollar sign, etc., with a subsequent instruction.)

R must be an even value (excluding 0) for proper operation of the instruction; if R is an odd value or equal to zero, the basic processor traps to location X'4D', instruction exception trap, with the contents in register R unchanged.

Contents of register R:



Contents of register R+1:



The destination byte string is an editing pattern that begins in the byte location pointed to by the destination address in register R + 1, and is C bytes in length. The decimal information field, which must be in packed decimal format, begins with the byte location pointed to by the displacement in EBS plus the source address in register R. The decimal information field must contain legal decimal digit and sign codes (packed format) and must begin with a decimal digit.

The destination byte string (the editing pattern) may contain any 8-bit codes desired. However, four byte codes in the editing pattern have special meanings. These codes are as follows:

Binary value	Function	Abbreviation
0010 0000 (X'20')	Digit selector	ds
0010 0001 (X'21')	Significance start	ss
0010 0010 (X'22')	Field separation	fs
0010 0011 (X'23')	Immediate significance start	si

Before executing EBS, the condition code should be set to 0000 if the high-order digit of the decimal number is in the left half of a byte, and should be set to 0100 if the high-order digit is in the right half of a byte.

The editing operation performed on each pattern byte of the destination byte string is determined by the following conditions:

1. The pattern byte obtained from the destination byte string.
2. The decimal digit obtained from the decimal number field.
3. The current state of the condition code.

Depending upon various combinations of these conditions, the instruction EDIT BYTE STRING performs one (and only one) of the following actions with the pattern byte and the decimal digit:

1. The fill character (contents of bit positions 0-7 of register R) or a blank character replaces the byte in the destination byte string.
2. The decimal digit is expanded to zoned decimal format and replaces the pattern byte in the destination byte string.
3. The pattern byte remains unchanged.

In general, the normal editing process is as follows:

1. Each byte of the destination byte string is replaced by a fill character until significance is present, either in the destination byte string or in the decimal information field. Significance is indicated by any of the following:
   a. The pattern byte is X'23' (immediate significance start), which begins significance with the current decimal digit.
   b. The pattern byte is X'21' (significance start), which begins significance with the following pattern byte.
   c. The current decimal digit is nonzero, which begins significance with the current pattern byte.
2. After significance is encountered, each pattern byte that is X'20' (digit selector), X'21' (significance start), X'22' (field separator), or X'23' (immediate significance start) is replaced by a zoned decimal number from the decimal field and all other pattern bytes are unchanged. This process continues until any of the following conditions occurs:
   a. A positive sign is encountered in the decimal field, in which case subsequent pattern bytes are replaced by blank characters until significance is again present, until a field separator is encountered, or until the destination byte string is entirely processed, whichever occurs first.
   b. A negative sign is encountered in the decimal field, in which case subsequent pattern bytes are unchanged until significance is again present, until a field separator is encountered, or until the destination byte string is entirely processed, whichever occurs first.
   c. A pattern byte of X'22' (field separator) is encountered, in which case the field separator is replaced by a fill character; subsequent pattern bytes are replaced by the fill character until significance is

again present, until a positive or negative sign is encountered, or until the destination byte string is entirely processed, whichever occurs first.

d. The destination byte string is entirely processed, in which case the basic processor executes the next instruction in sequence.

Detailed operation of EDIT BYTE STRING follows. The explanation is necessarily quite detailed due to the high degree of flexibility inherent in EBS. Condition code settings are made continuously during the editing process and these settings help determine how each subsequent pattern byte will be edited. The summary of condition code settings given later in this section will help clarify the following discussion:

1. If the count in bit position 0-7 of register R+1 is a nonzero, a pattern byte is obtained from the destination byte string; if the count in register R+1 is 0, the basic processor executes the next instruction in sequence.

2. If the pattern byte is a digit selector (X'20'), a significance start (X'21'), or immediate significance start (X'23'), a digit is accessed from the decimal information field as follows:

   a. A decimal byte is obtained from the byte location pointed to by the displacement in EBS plus the source address in register R.

   b. If bits 0-3 of the decimal byte are a sign code, the basic processor automatically aborts execution of EBS and traps to location X'45', with the contents of register R, register R+1, the condition code, and the destination byte string unchanged from their current contents.

   c. If CC2 is currently set to 0, the digit to be used for editing is the left digit (bits 0-3) of the decimal byte; however, if CC2 is currently set to 1, the digit to be used is the right digit (bits 4-7) of the decimal byte. In either case, CC3 is set to 1 if the digit is nonzero. If CC2 is set to 1 and the right digit (bits 4-7) of the decimal byte is a sign code, the basic processor automatically aborts execution of EBS and traps to location X'45', as described above.

   d. One of the following editing actions is performed:

Conditions	Action	Mark
Pattern byte=SI(X'23')	Expand digit to zoned format, store in pattern byte location, and set CC4 to 1 (start significance).	Mode 1
Pattern byte=SS(X'21') CC4=1	Expand digit to zoned format and store in pattern byte location	None
Pattern byte=SS(X'21') CC4=1 (cont.)	(because CC4=1 means significance already encountered).	
Pattern byte=SS CC4=0 nonzero digit	Expand digit to zoned format, store in pattern byte location (because nonzero digit begins significance), and set CC4 to 1.	Mode 1
Pattern byte=SS CC4=0 digit=0	Store fill character in pattern byte location (because significance starts with next pattern byte) and set CC4 to 1.	Mode 2
Pattern byte=DS(X'20') CC4=1	Expand digit to zoned format, and store digit in pattern byte location.	None
Pattern byte=DS CC4=0 nonzero digit	Expand digit to zoned format, store digit in pattern byte location, and set CC4 to 1 to signal significance.	Mode 1
Pattern byte=DS CC4=0 digit=0	Store fill character in pattern byte location (because significance not encountered yet).	None

   e. If CC2 is currently reset to 0 and if bits 4-7 of the decimal byte are a positive decimal sign code, CC1 is set to 1, CC4 is reset to 0, and the source address in register R is incremented by 1. If CC2 is currently reset to 0 and if bits 4-7 of the decimal byte are a negative decimal sign code, CC1 and CC4 are both set to 1, and the source address is incremented by 1. Otherwise, CC2 is added to the source address and then CC2 is inverted.

   f. If marking is invoked at set d, above, one of the two following marking operations are performed:

      Mode 1: Load bits 13-31 of register R+1 into bit positions 13-31 of register 1; bit positions 0-12 of register are unpredictable.

      Mode 2: Load bits 13-31 of register R+1 into bit positions 13-31 of register 1 and then increment the contents of register 1 by 1; bit positions 0-12 of register 1 are unpredictable.

      If marking is not applicable (i.e., significance has not been encountered), the contents of register 1 are not affected.

3. If the pattern byte is a field separator (X'22'), the fill character is stored in the pattern byte location. CC1, CC3, and CC4 are all reset to 0's, and CC2 remains unchanged.

4. If the pattern byte is not a digit selector, significance start, immediate significance start, or field separator, one of the following actions are performed:

Conditions	Action
CC1=0 and CC4=0	Store fill character in pattern byte location.
CC1=1 and CC4=0	Store blank character (X'40') in pattern byte location.
CC4=1	None (pattern byte remains unchanged).

5. Increment the destination address in register Ru1 and decrement the count in register Ru1. If the count is still nonzero, process the next pattern byte as above; otherwise, execute the next instruction in sequence.

Affected: (R), (Ru1) (register 1), (DBS), CC

Traps: Nonexistent instruction, decimal arithmetic, instruction exception

edited (SBS) ⟶ DBS

Condition code settings:

1	2	3	4	Result of EBS
0	-	-	0	Significance is not present, no sign digit has been encountered.
0	-	-	1	Significance is present, no sign digit has been encountered.
1	-	-	0	A positive sign has been encountered.
1	-	-	1	A negative sign has been encountered.
-	0	-	-	Next digit to be processed is left digit of byte.
-	1	-	-	Next digit to be processed is right digit of byte.
-	-	0	-	No nonzero digit has been encountered.
-	-	1	-	A nonzero digit has been encountered.

If EBS is indirectly addressed, it is treated as a nonexistent instruction. The basic processor unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R, register Ru1, register 1, the destination byte string, and the condition code unchanged.

The R field of the EBS instruction must be an even value (excluding 0) for proper operation of the instruction; if the R field is an odd value or equal to zero, the instruction traps to location X'4D', instruction exception trap.

If an illegal digit or sign is detected in the decimal information field, the basic processor unconditionally aborts execution of the instruction (at the time the illegal digit or sign is encountered) and traps to location X'45' with the contents of register R, register Ru1, register 1, the destination byte string, and the condition code containing the results of the last editing operation performed before the illegal digit or sign was encountered.

See "Traps By Byte-String Instructions" (in this section) for other trap conditions.

In the following examples, the hexadecimal codes for the digit selector (X'20'), the significance start (X'21'), the field separation (X'22'), and the immediate significance start (X'23') are represented by the character groups ds, ss, fs, and si, respectively. Also, the symbol ♭ is used to represent the character blank (X'40'). Note that code X'5C' represents the * symbol.

Example 1, before execution:

The instruction word is

X'63600000'

The contents of register 6 are:

X'5C000100'

The contents of register 7 are:

X'0C001000'

The contents of the decimal information field beginning at byte location X'100' are:

00 00 00 0+

The contents of the destination byte string beginning at byte location X'1000' are:

ds ds , ds ds ss . ds ds ♭ C R

The condition code is:

0000

Example 1, after execution:

The instruction word is unchanged.

The new contents of register 6 are:

X'5C000104'

The new contents of register 7 are:

X'0000100C'

The contents of the decimal information field are unchanged.

The new contents of the destination byte string are:

******.00bbb

The new condition code is:

1000

The contents of register 1 are:

X'xxx01006'

By subsequent programming, a floating dollar sign can be inserted in front of the first significant character of the edited byte string by using the contents of register 1, minus 1, as the address of the byte location where the dollar sign is to be inserted.

Example 2, before execution:

The initial conditions are identical to example 1, except that the contents of the decimal information field are:

06 54 32 1-

Example 2, after execution:

The instruction word and the decimal field are unchanged.

The new contents of registers 6 and 7 are identical to those given for example 1.

The new contents of the destination byte string are:

*6,543.21bCR

The new condition code is:

1011

The new contents of register 1 are:

X'xxx01001'

Example 3, before execution:

The initial conditions are identical to example 1, except that the contents of the decimal field are:

00 54 32 1+

Example 3, after execution:

The instruction word and the decimal field are unchanged.

The new contents of registers 6 and 7 are identical to that given for example 1.

The new contents of the destination byte string are:

***543.21bbb

The new condition code is:

1010

The new contents of register 1 are:

X'xxx01003'

Example 4, before execution:

The instruction word is:

X'63400100'

The contents of register 4 are:

X'7B001000'

The contents of register 5 are:

X'19002000'

The contents of the decimal information field beginning at byte location X'1100' are:

06 12 50 0+ 01 23 4+ 03 5-

The contents of the destination byte string beginning at byte location X'2000' are:

A ds ds si . ds ds ds fs B ds ds ss . ds ds C fs D

si ds ds END

The condition code is:

0100

Example 4, after execution:

The instruction word is unchanged.

The new contents of register 4 are:

X'7B001009'

The new contents of register 5 are:

X'00002019'

The decimal information field is unchanged.

The new contents of the destination byte string are:

```
#612.500###12.346##035END
```

The new condition code is:

1011

The new contents of register 1 are:

X'xxx02013'

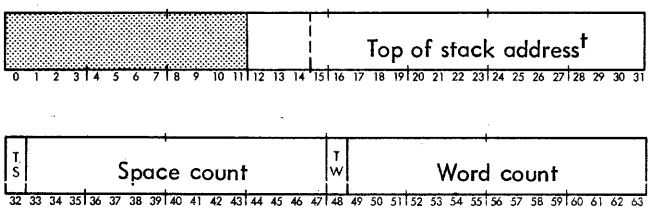
PUSH-DOWN INSTRUCTIONS (NON-PRIVILEGED)

The term "push-down processing" refers to the programming technique (used extensively in recursive routines) of storing the context of a calculation in memory, proceeding with a new set of information, and then activating the previously stored information. Typically, this process involves a reserved area of memory (stack) into which operands are pushed (stored) and from which operands are pulled (loaded) on a last-in, first-out basis. The basic processor provides for simplified and efficient programming of push-down processing by means of the following non-privileged instructions:

Instruction Name	Mnemonic
Push Word	PSW
Pull Word	PLW
Push Multiple	PSM
Pull Multiple	PLM
Modify Stack Pointer	MSP

STACK POINTER DOUBLEWORD (SPD)

Each non-privileged push-down instruction operates with respect to a memory stack that is defined by a doubleword located at effective address of the instruction. This doubleword, referred to as a stack pointer doubleword (SPD), has the following structure:



†For real extended mode of addressing this is a 20-bit field (12-31); for real and virtual addressing modes it is a 17-bit field (15-31).

Bit positions 15 through 31 of the SPD contain a 17-bit address field† that points to the location of the word currently at the top (highest-numbered address) of the operand stack. In a push operation, the top-of-stack address is incremented by 1 and then an operand in a general register is pushed (stored) into that location, thus becoming the contents of the new top of the stack; the contents of the previous top of the stack remain unchanged. In a pull operation, the contents of the current top of the stack are pulled (loaded) into a general register and then the top-of-stack address is decremented by 1; the contents of the stack remain unchanged.

Bit positions 33 through 47 of the SPD, referred to as the space count, contain a 15-bit count (0 to 32,767) of the number of word locations currently available in the region of memory allocated to the stack. Bit positions 49 through 63 of the SPD, referred to as the word count, contain a 15-bit count (0 to 32,767) of the number of words currently in the stack. In a push operation, the space count is decremented by 1 and the word count is incremented by 1; in a pull operation, the space count is incremented by 1 and the word count is decremented by 1. At the beginning of all non-privileged push-down instructions, the space count and the word count are each tested to determine whether the instruction would cause either count field to be incremented above the upper limit of $2^{15}-1$ (32,767), or to be decremented below the lower limit of 0. If execution of the push-down instruction would cause either count limit to be exceeded, the basic processor unconditionally aborts execution of the instruction, with the stack, the stack pointer doubleword, and the contents of general registers unchanged. Ordinarily, the basic processor traps to location X'42' after aborting a push-down instruction because of impending stack limit overflow or underflow, and with the condition code unchanged from the value it contained before execution of the instruction.

However, this trap action can be selectively inhibited by setting either (or both) of the trap inhibit bits in the SPD to 1.

Bit position 32 of the SPD, referred to as the trap-on-space (TS) inhibit bit, determines whether the basic processor will trap to location X'42' as a result of impending overflow or underflow of the space count (SPD_{33-47}), as follows:

TS	Space count overflow/underflow action
0	If the execution of a pull instruction would cause the space count to exceed $2^{15}-1$, or if the execution of a push instruction would cause the space count to be less than 0, the basic processor traps to location X'42' with the condition code unchanged.
1	Instead of trapping to location X'42', the basic processor sets CC1 to 1 and then executes the next instruction in sequence.

Bit position 48 of the SPD, referred to as the trap-on-word (TW) inhibit bit, determines whether the basic processor

traps to location X'42' as a result of impending overflow or underflow of the word count (SPD_{49-63}), as follows:

TW	Word count overflow/underflow action
0	If the execution of a push instruction would cause the word count to exceed $2^{15}-1$, or if the execution of a pull instruction would cause the word count to be less than 0, the basic processor traps to location X'42' with the condition code unchanged.
1	Instead of trapping to location X'42', the basic processor sets CC3 to 1 and then executes the next instruction in sequence.

PUSH-DOWN CONDITION CODE SETTINGS

If the execution of a push-down instruction is attempted and the basic processor traps to location X'42', the condition code remains unchanged from the value it contained immediately before the instruction was executed.

If the execution of a push-down instruction is attempted and the instruction is aborted because of impending stack limit overflow or underflow (or both) but the push-down stack limit trap is inhibited by one (or both) of the inhibits (TS and TW), then, CC1 or CC3 is set to 1 (or both are set to 1's) to indicate the reason for aborting the push-down instruction, as follows:

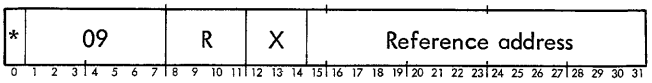
1	2	3	4	Reason for abort
0	-	1	-	Impending overflow of word count on a push operation or impending underflow of word count on a pull operation. The push-down stack limit trap was inhibited by the TW bit (SPD_{48}).
1	-	0	-	Impending overflow of space count on a pull operation or impending underflow of space count on a push operation. The push-down stack limit trap was inhibited by the TS bit (SPD_{32}).
1	-	1	-	Impending overflow of word count and underflow of space count on a push operation or impending overflow of space count and underflow of word count on a pull operation. The push-down stack limit trap was inhibited by both the TW and the TS bits.

If a push-down instruction is successfully executed, CC1 and CC3 are reset to 0 at the completion of the instruction. Also, CC2 and CC4 are independently set to indicate the current status of the space count and the word count, respectively, as follows:

1	2	3	4	Status of space and word counts
-	0	-	0	The current space count and the current word count are both greater than zero.
-	0	-	1	The current space count is greater than zero, but the current word count is zero, indicating that the stack is now empty. If the next operation on the stack is a pull instruction, the instruction will be aborted.
-	1	-	0	The current word count is greater than zero, but the current space count is zero, indicating that the stack is now full. If the next operation on the stack is a push instruction, the instruction will be aborted.

If the basic processor does not trap to location X'42' as a result of impending stack limit overflow/underflow, CC2 and CC4 indicate the status of the space and word counts at the termination of the push-down instruction, regardless of whether the space and word counts were actually modified by the instruction. In the following descriptions of the push-down instructions, condition code settings given are only those that can be produced by the instruction, provided that the basic processor does not trap to location X'42'.

PSW PUSH WORD
(Doubleword index alignment)



PUSH WORD stores the contents of register R into the push-down stack defined by the stack pointer doubleword located at the effective doubleword address of PSW. If the push operation can be successfully performed, the instruction operates as follows:

1. The current top-of-stack address (SPD_{15-31})[†] is incremented by 1 to point to the new top-of-stack location.
2. The contents of register R are stored in the location pointed to by the new top-of-stack address.

[†]For real extended mode of addressing this is a 20-bit field (12-31); for real and virtual addressing modes it is a 17-bit field (15-31).

3. The space count (SPD_{33-47}) is decremented by 1 and the word count (SPD_{49-63}) is incremented by 1.

4. The condition code is set to reflect the new status of the space count.

Affected: (SPD), (TSA+1), CC Trap: Push-down stack limit

$(SPD)_{15-31} + 1 \longrightarrow SPD_{15-31}$[t]

$(R) \longrightarrow (SPD_{15-31})$[t]

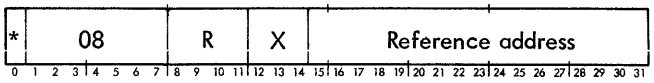
$(SPD)_{33-47} - 1 \longrightarrow SPD_{33-47}$

$(SPD)_{49-63} + 1 \longrightarrow SPD_{49-63}$

Condition code settings:

1	2	3	4	Result of PSW	
0	0	0	0	Space count is greater than 0.	Instruction completed
0	1	0	0	Space count is now 0.	
0	0	1	0	Word count = $2^{15}-1$, TW = 1.	Instruction aborted
1	1	0	0	Space count = 0, TS = 1.	
1	1	0	1	Space count = 0, word count = 0, TS = 1.	
1	1	1	0	Word count = $2^{15}-1$, space count = 0, TW = 1, and TS = 1.	

PLW PULL WORD
(Doubleword index alignment)



PULL WORD loads register R with the word currently at the top of the push-down stack defined by the stack pointer doubleword located at the effective doubleword address of PLW. If the pull operation can be performed successfully, the instruction operates as follows:

1. Register R is loaded with the contents of the location pointed to by the current top-of-stack address (SPD_{15-31})[t].

2. The current top-of-stack address is decremented by 1, to point to the new top-of-stack location.

[t]For real extended mode of addressing this is a 20-bit field (12-31); for real and virtual addressing modes it is a 17-bit field (15-31).

3. The space count (SPD_{33-47}) is incremented by 1 and the word count (SPD_{49-63}) is decremented by 1.

4. The condition code is set to reflect the status of the new word count.

Affected: (SPD), (R), CC Trap: Push-down stack limit

$(SPD)_{15-31} \longrightarrow R$; $(SPD)_{15-31} - 1 \longrightarrow SPD_{15-31}$[t]

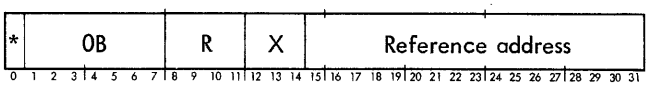
$(SPD)_{33-47} + 1 \longrightarrow SPD_{33-47}$;

$(SPD)_{49-63} - 1 \longrightarrow SPD_{49-63}$

Condition code settings:

1	2	3	4	Result of PLW	
0	0	0	0	Word count is greater than 0.	Instruction completed
0	0	0	1	Word count is now 0.	
0	0	1	1	Word count = 0, TW = 1.	Instruction aborted
0	1	1	1	Space count = 0, word count = 0, TW = 1.	
1	0	0	0	Space count = $2^{15}-1$, TS = 1.	
1	0	1	1	Space count = $2^{15}-1$, word count = 0, TS = 1, and TW = 1.	

PSM PUSH MULTIPLE
(Doubleword index alignment)



PUSH MULTIPLE stores the contents of a sequential set of general registers into the push-down stack defined by the stack pointer doubleword located at the effective doubleword address of PSM. The condition code must contain a count of the number of registers to be pushed into the stack. (An initial value of 0000 for the condition code specifies that all 16 general registers are to be pushed into the stack.) The registers are treated as a circular set (with register 0 following register 15) and the first register to be pushed into the stack is register R. The last register to be pushed in to the stack is register R + CC -1, and the contents of this register become the contents of the new top-of-stack location.

If there is sufficient space in the stack for all of the specified registers, PSM operates as follows:

1. The contents of registers R to R = CC - 1 are stored in ascending sequence, beginning with the location tion pointed to by the current top-of-stack address $(SPD_{15-31})^t$ plus 1 and ending with the current top-of-stack address plus CC.

2. The current top-of-stack address is incremented by the value of CC, to point to the new top-of-stack location.

3. The space count (SPD_{33-47}) is decremented by the value of CC and the word count is incremented by the value of CC.

4. The condition code is set to reflect the new status of the space count.

Affected: (SPD), (TSA+1) to (TSA+CC), CC Trap: Push-down stack limit

$(R) \longrightarrow (SPD)_{15-31} + 1 \ldots (R+CC-1) \longrightarrow (SPD)^t_{15-31} + CC$

$(SPD)_{15-31}{}^{+CC} \longrightarrow SPD_{15-31}{}^t$

$(SPD)_{33-47}{}^{-CC} \longrightarrow SPD_{33-47}$

$(SPD)_{49-63}{}^{+CC} \longrightarrow SPD_{49-63}$

Condition code settings:

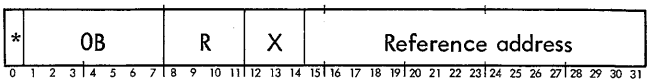
1	2	3	4	Result of PSM	
0	0	0	0	Space count > 0.	Instruction completed
0	1	0	0	Space count = 0.	
0	0	1	0	Word count + CC > 2^{15}-1, TW = 1.	Instruction aborted
1	0	0	0	Space count < CC, TS = 1.	
1	0	0	1	Space count < CC, word count = 0, TS = 1.	
1	0	1	0	Space count < CC, word count + CC > 2^{15}-1 TS = 1, and TW = 1.	
1	1	0	0	Space count = 0, TS = 1.	
1	1	0	1	Space count = 0, word count = 0, TS = 1.	
1	1	1	0	Space count = 0, word count + CC > 2^{15}-1, TS = 1, and TW = 1.	

[t] For real extended mode of addressing this is a 20-bit field (12-31); for real and virtual addressing modes it is a 17-bit field (15-31).

If the instruction operation extends into a memory page protected either by the access protection codes or write locks, the memory protection trap can occur. If the operation extends into a memory region that is physically not present, the nonexistent memory address trap can occur.

If the address of the elements within the stack (pointed to by the top-of-stack address) is in the range 0 through 15, then the registers indicated by the R field of the PSM instruction are stored in the general registers rather than in main memory. In this case the results will be unpredictable if any source registers are also used as destination registers.

PLM PULL MULTIPLE
(Doubleword index alignment)



PULL MULTIPLE loads a sequential set of general registers from the push-down stack defined by the stack pointer doubleword located at the effective doubleword address of PLM. The condition code must contain a count of the number of words to be pulled from the stack. (An initial value of 0000 for the condition code specifies that 16 words are to be pulled from the stack.) The registers are treated as a circular set (with register 0 following register 15), the first register to be loaded from the stack is register R+CC-1, and the contents of the current top-of-stack location becomes the contents of this register. The last register to be loaded is register R.

If there is a sufficient number of words in the stack to load all of the specified registers, PLM operates as follows:

1. Registers R+CC-1 to register R are loaded in descending sequence, beginning with the contents of the location pointed to by the current top-of-stack address $(SPD_{15-31})^t$ and ending with the contents of the location pointed to by the current top-of-stack address minus CC-1.

2. The current top-of-stack address is decremented by the value of CC, to point to the new top-of-stack location.

3. The space count (SPD_{33-47}) is incremented by the value of CC and the word count is decremented by the value of CC.

4. The condition code is set to reflect the new status of the word count.

Affected: (SPD), (R+CC-1) to (R), CC Trap: Push-down stack limit

$((SPD)_{15-31}{}^t \longrightarrow R + CC - 1, \ldots,$

$((SPD)_{15-31} - |CC - 1|) \longrightarrow R^t$

$(SPD)_{15-31} - CC \longrightarrow SPD_{15-31}{}^t$

$(SPD)_{33-47} + CC \longrightarrow SPD_{33-47}$

$(SPD)_{49-63} - CC \longrightarrow SPD_{49-63}$

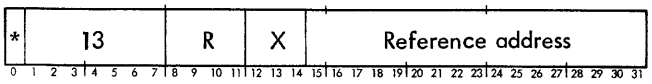
Condition code settings:

1	2	3	4	Result of PLM	
0	0	0	0	Word count > 0	Instruction completed
0	0	0	1	Word count = 0	
0	0	1	0	Word count < CC, TW = 1	Instruction aborted
0	0	1	1	Word count = 0, TW = 1	
0	1	1	0	Space count = 0, word count < CC, TW = 1	
0	1	1	1	Space count = 0, word count = 0, TW = 1	
1	0	0	0	Space count + CC > 2^{15}-1, TS = 1	
1	0	1	0	Space count + CC > 2^{15}-1, word count < CC, TS = 1, and TW = 1	
1	0	1	1	Space count + CC > 2^{15}-1, word count = 0, TS = 1, and TW = 1	

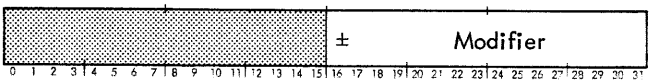
If the instruction operation extends into a memory page protected either by the access protection codes or write locks, the memory protection trap can occur. If the operation extends into a memory region that is physically not present, the nonexistent memory address trap can occur.

If the address of the elements within the stack (pointed to by the top-of-stack address) is in the range 0 through 15, then the words to be loaded are taken from the general registers rather than from main memory. In this case, the results will be unpredictable if any of the source registers are also used as destination registers.

MSP MODIFY STACK POINTER
(Doubleword index alignment)



MODIFY STACK POINTER modifies the stack pointer doubleword, located at the effective doubleword address of MSP by the contents of register R. Register R must have the following format:



Bit positions 16 through 31 of register R are treated as a signed integer, with negative integers in two's complement form (i.e., a fixed-point halfword). The modifier is algebraically added to the top-of-stack address, subtracted from the space count, and added to the word count in the stack pointer doubleword. If, as a result of MSP, either the space count or the word count would be decreased below 0 or increased above 2^{15}-1, the instruction is aborted. Then, the basic processor either traps to location X'42' or sets the condition code to reflect the reason for aborting, depending on the stack limit trap inhibits.

If the modification of the stack pointer doubleword can be successfully performed, MSP operates as follows:

1. The modifier in register R is algebraically added to the current top-of-stack address $(SPD_{15-31})^{t}$, to point to a new top-of-stack location. (If the modifier is negative, it is extended to 17 bits by appending a high-order 1.)
2. The modifier is algebraically subtracted from the current space count (SPD_{33-47}) and the result becomes the new space count.
3. The modifier is algebraically added to the current word count (SPD_{49-63}) and the result becomes the new word count.
4. The condition code is set to reflect the new status of the new space count and new word count.

Affected: (SPD), CC Trap: Push-down stack limit

$(SPD)_{15-31} + (R)_{16-31SE} \longrightarrow SPD_{15-31}$ [t]

$(SPD)_{33-47} - (R)_{16-31} \longrightarrow SPD_{33-47}$

$(SPD)_{49-63} + (R)_{16-31} \longrightarrow SPD_{49-63}$

Condition code settings:

1	2	3	4	Result of MSP	
0	0	0	0	Space count > 0, word count > 0.	Instruction completed
0	0	0	1	Space count > 0, word count = 0.	
0	1	0	0	Space count = 0, word count > 0.	
0	1	0	1	Space count = 0, word count = 0, modifier = 0.	

[t] For real extended mode of addressing this is a 20-bit field (12-31); for real and virtual addressing modes it is a 17-bit field (15-31).

If CC1, or CC3, or both CC1 and CC3 are 1's after execution of MSP, the instruction was aborted but the push-down stack limit trap was inhibited by the trap-on-space inhibit (SPD_{32}), by the trap-on-word inhibit (SPD_{48}), or both. The condition code is set to reflect the reason for aborting as follows:

1	2	3	4	Status of space and word counts
-	-	-	0	Word count > 0.
-	-	-	1	Word count = 0.
-	-	0	-	$0 \leq$ word count + modifier $\leq 2^{15}-1$.
-	-	1	-	Word count + modifier < 0, and TW = 1 or word count + modifier > $2^{15}-1$, and TW = 1.
-	0	-	-	Space count > 0.
-	1	-	-	Space count = 0.
0	-	-	-	$0 \leq$ space count - modifier < $2^{15}-1$.
1	-	-	-	Space count - modifier < 0, and TS = 1 or space count - modifier > $2^{15}-1$, and TS = 1.

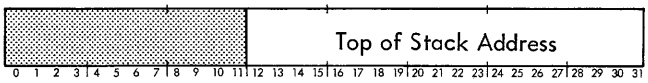
PUSH-DOWN INSTRUCTIONS (PRIVILEGED)

The computer has two privileged push-down instructions: PUSH STATUS (PSS) and PULL STATUS (PLS). These two instructions and a Status Stack Pointer Doubleword facilitate the storing (pushing) or loading (pulling) of a particular environment (contents of 16 general registers and Program Status Words) into or out of a memory stack.

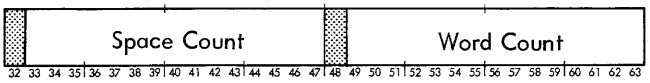
STATUS STACK POINTER DOUBLEWORD

The Status Stack Pointer Doubleword (SSPD) always resides in real memory locations 0 and 1 and is dedicated for PSS and PLS instructions. The format of parameters contained within the Status Stack Pointer Doubleword are as follows:

Real Memory Location 0:



Real Memory Location 1:



TOP OF STACK ADDRESS

The Top of Stack Address (TSA) is always a 20-bit real memory word address and is never mapped. Depending upon programming considerations, the initial TSA is a specific value either as the result of a Mode 0, WRITE DIRECT instruction or as the result of a PSS or PLS instruction, as described below.

During each PSS instruction, the memory stack is accessed 28 times and the TSA is incremented by 1 before each access. The first memory stack location accessed has a relative address equal to the initial TSA plus 1, ..., and the 28th memory stack location accessed has a relative address equal to the initial TSA plus 28. Although 28 memory stack locations are accessed in an ascending sequence, only 20 locations (as selected by the hardware) will contain the basic processor environment. Eight locations (whose contents are designated as "indeterminate", in Figure 12) are reserved and must not be used.

For each PLS instruction, access to the memory stack is contingent upon the Word Count as described subsequently. If access is permitted, the memory stack is accessed 28 times and the TSA is decremented by 1 after each access. The first memory stack location accessed by a PLS instruction has a relative address equal to the initial TSA, the second memory stack location accessed has a relative address equal to the initial TSA minus 1, ..., and the 28th memory stack location accessed has a relative address equal to the initial TSA minus 27. Although 28 memory stack locations are accessed in a descending sequence, the hardware selects and pulls the contents of only 20 locations containing valid information, as shown in Figure 12, and loaded into the general registers and PSWs. The contents of eight locations designated as indeterminate are ignored.

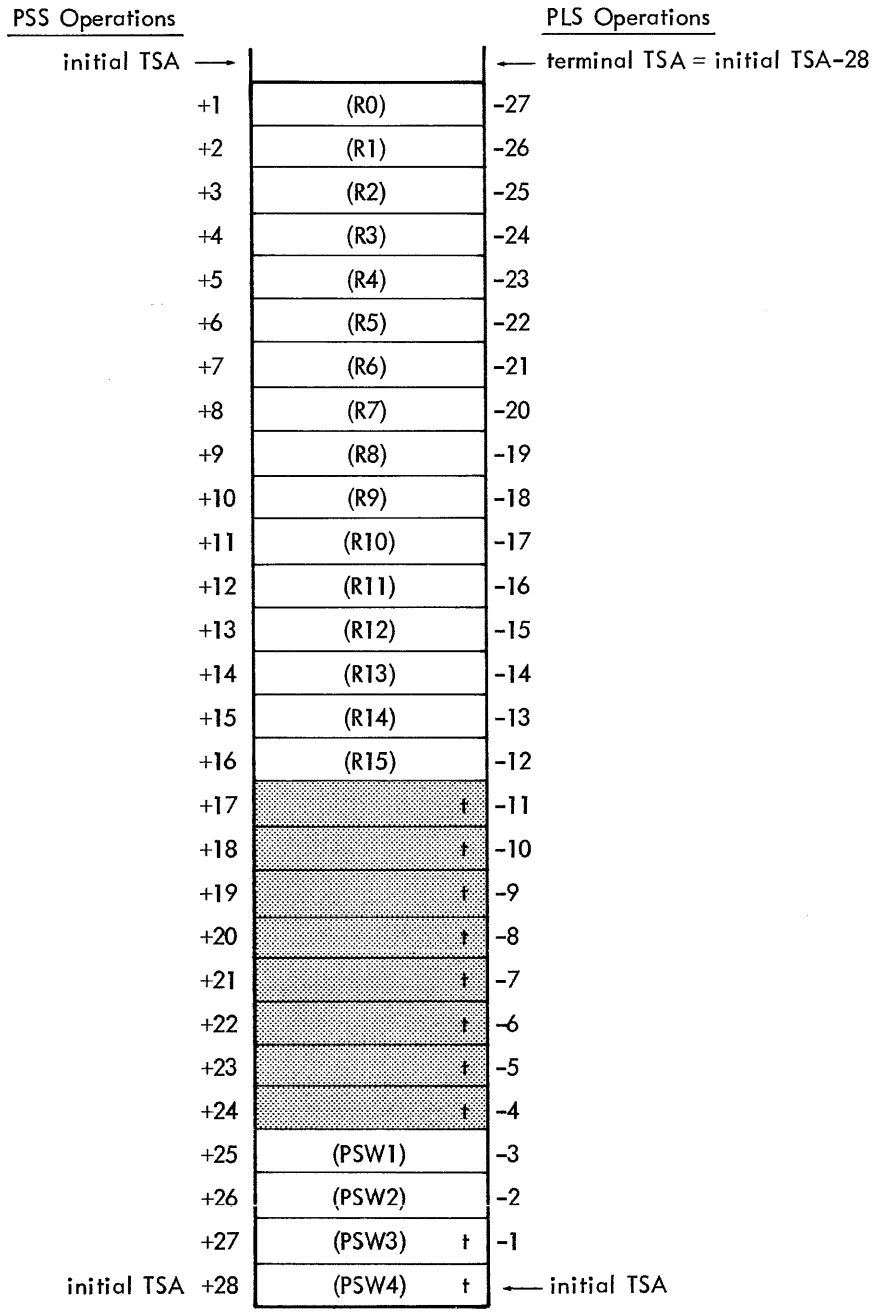
If the terminal (last) TSA for a PSS or PLS instruction is not modified by a Mode 0 WRITE DIRECT instruction, it may be used as the initial TSA for a subsequent PSS or PLS instruction. Each PSS instruction causes the memory stack to be increased by 28 word locations and each PLS instruction causes the memory stack to be decreased by 28 word locations. The information is pushed and pulled on a last-in, first-out basis.

Note: The PLS instruction is contingent upon the Word Count value, as described below.

SPACE COUNT

The Space Count field (bit positions 33-47) of the Status Stack Pointer Doubleword is a 15-bit counter that may contain a value of 0 through 32,767. Depending upon programming considerations, the initial Space Count is a specific value either as the result of executing a Mode 0, WRITE DIRECT instruction or a PLS or PSS instruction.

During a PSS instruction, the Space Count is decremented by 1 for each word pushed into the memory stack. If the Space Count is decremented to a value of zero before all the words have been pushed, the PSS instruction continues (i.e., no trapping occurs). The environment is stored into



[t] As a function of the hardware, the contents of these 8 locations are indeterminate after a PSS instruction and ignored by a PLS instruction. These locations are reserved for future enhancements and must not be used.

Figure 12. Typical 28-Word Portion of Memory Stack for PSS and PLS

appropriate memory stack locations as specified by the TSA; however, subsequent values of the Space Count are indeterminate.

During a PLS instruction, the Space Count is incremented by 1 for each word pulled from the memory stack. If the Space Count is incremented beyond a value of 32,767, bit position 32 is set to 1 (signifying an overflow condition); however, the PLS instruction continues (i.e., no trapping occurs).

Note: Once bit position 32 has been set to a 1, it can be reset to a 0 only by executing a Mode 0, WRITE DIRECT instruction. That is, bit position 32 can not be reset to a 0 by the decrementing process performed during a PSS instruction.

WORD COUNT

The Word Count field (bit positions 49-63) of the Status Stack Pointer Doubleword is a 15-bit counter that may contain a value of 0 through 32,767. Depending upon programming considerations, the initial Word Count is a specific value either as the result of executing a Mode 0, WRITE DIRECT instruction or as the result of executing a PSS or PLS instruction.

During a PSS instruction, the Word Count is incremented by 1 for each word pushed into the memory stack. Thus, the terminal Word Count for a PSS instruction exceeds the initial Word Count by 28. If the Word Count value exceeds 32,767, bit position 48 is set to a 1 (signifying that an overflow condition has occurred); however, the PSS instruction continues the stacking operation (i.e., no trapping occurs).

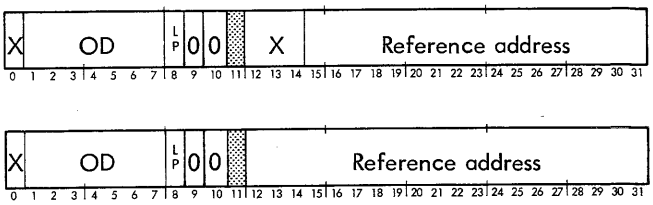
If the initial Word Count for a PLS instruction is equal to or greater than 28, the Word Count is decremented by 1 for each word pulled from the memory stack and the terminal Word Count will be 28 less than the initial Word Count. Note that if bit position 48 was set to a 1 by a PSS instruction previously, it can not be reset to a 0 by the decrementing performed during a PLS instruction.

If the initial Word Count for a PLS instruction is equal to zero, the parameters within the Status Stack Pointer Doubleword are neither effective nor affected by the PLS instruction. However, default PSWs are loaded from real memory locations 2 and 3.

If the initial Word Count for a PLS instruction is less than 28 and not equal to zero, the other parameters of the Status Stack Pointer Doubleword are not effective and none of the parameters are affected by the PLS instruction. Instead the BP traps to location X'4D' (instruction exception trap) and TCC2 is set.

PSS PUSH STATUS

(Doubleword index alignment, privileged)



PUSH STATUS loads new Program Status Words from an effective doubleword location and stores the current environment (current Program Status Words and contents of all 16 general registers) into a memory stack, as defined by the Status Stack Pointer Doubleword. Note that the reference address points to the memory location of the new PSWs.

The PSS instruction is used for three types of operations: as a normal instruction in an ongoing program; as an interrupt instruction; and as a trap instruction. The effective address of a PSS instruction is generated in one of the following ways:

PSS – normal instruction (see first instruction diagram)

When a PSS instruction is encountered in the course of execution of normal programs, the effective address is generated according to the rules for addressing then in effect as described by the currently active PSWs; that is, the basic processor is operating in real, real extended, or virtual addressing mode. The flags in bit positions 9 and 10 have no effect and must be coded as zeros.

PSS – interrupt instruction (see second instruction diagram)

A PSS instruction (in an interrupt location) executed as a result of an interrupt is called an interrupt instruction. In the interrupt execution sequence, the 20-bit reference address is always real, independent of the map invoking bit in the PSWs. There is no indexing possible since the designator field is preempted by the reference address. Indirect addressing is permitted with precisely the same constraints. The indirect address word contains a 20-bit real address with the same properties as the reference address described above. The flags in bit positions 9 and 10 have no effect and must be coded as zeros.

PSS – trap instruction (see second instruction diagram)

A PSS instruction (in a trap location) executed as a result of a trap entry operation is called a trap instruction. In a trap execution sequence, the 20-bit reference address may be either a real address or a virtual address according to the map invoking bit in the PSWs. There is no indexing possible since the index field is used for addressing. If indirect addressing is specified, the effective address is generated according to the rules for addressing then in effect as described by the currently active PSWs. Bit positions 9 and 10 must be coded as zeros.

Depending upon the type of addressing, the reference address of the PSS instruction is converted into an effective virtual doubleword address, as described under "PSS Address Calculations", in Chapter 2. Except for the Register Block Pointer field (bit positions 56-59) and the interrupt group inhibit bits (bit positions 37, 38, and 39), the contents of the effective location are always loaded as the new PSWs. If the LP flag (bit 8 of the PSS instruction) is a 1, the Register Block Pointer of the new PSWs is also loaded. If the LP flag is a 0, the old Register Block Pointer is retained. The interrupt group inhibit bits of the new PSWs are "ORed" with the corresponding bits of the old PSWs.

The current environment (comprised of 20 words) is stored in memory stack locations having the following relative addresses: initial TSA+1 through initial TSA+16, initial TSA+25, and initial TSA+26. Memory stack locations having relative addresses of initial TSA+17 through initial TSA+24, initial TSA+27, and initial TSA+28 are reserved and the contents are indeterminate.

The parameters of the Status Stack Pointer Doubleword (as contained within working registers) are appropriately modified to reflect the progress of the PSS instruction and conditions of the memory stack (i.e., the TSA and Word Count are incremented and the Space Count is decremented for each memory word location accessed, as described under Status Stack Pointer Doubleword).

If the Word Count exceeds 32,767 (maximum count for bits 49-63) or if the Space Count is reduced to zero before the PSS instruction is completed, the stacking operations continue until 28 words have been pushed (i.e., no trapping occurs). When the Word Count exceeds 32,767, bit 48 is set to a 1. Attempting to decrement the Space Count below zero causes the Space Count to become indeterminate.

Affected: (PSWs), CC, Memory Stack, Status Stack Pointer Doubleword.

(PSWs) and CC:

$ED_{0-3} \longrightarrow$ CC;

$ED_{4-7} \longrightarrow$ FR, FS, FZ, FN;

$ED_{8} \longrightarrow$ MS;

$ED_{9} \longrightarrow$ MM;

$ED_{10} \longrightarrow$ DM;

$ED_{11} \longrightarrow$ AM;

$ED_{15-31} \longrightarrow$ IA;

$ED_{32-35} \longrightarrow$ WK;

ED_{37-39} u CI, II, EI $\longrightarrow$ CI, II, EI
(Note: "u" represents inclusive OR.)

$ED_{56-59} \longrightarrow$ RP only if $(I_8) = 1$

$ED_{60} \longrightarrow$ RA

$ED_{61} \longrightarrow$ MA

Memory Stack:

(General Register n) $\longrightarrow$ (initial TSA+(n+1) where n has ascending values from 0 through 15.

PSW1 $\longrightarrow$ (initial TSA+25)

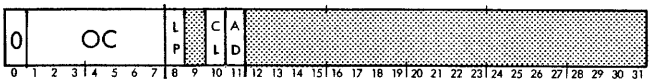
PSW2 $\longrightarrow$ (initial TSA+26)

Status Stack Pointer Doubleword:

TSA+1 $\longrightarrow$ TSA until terminal TSA=initial TSA+28; Word Count + 1 $\longrightarrow$ Word Count until terminal Word Count = initial Word Count + 28, (if Word Count > 32,767, set bit 48 to 1);

Space Count - 1 $\longrightarrow$ Space Count until terminal Space Count = initial Space Count - 28 (if Space Count = 0, Space Count - 1 is indeterminate).

PLS PULL STATUS (nonaddressing, privileged)



PULL STATUS, in conjunction with the Status Stack Pointer Doubleword, may cause one or more of the following functions to be performed:

1. Selectively load a new environment (PSWs and 16 general registers) from the memory stack; or,

2. Selectively load default PSWs from dedicated memory locations; and,

3. Selectively clear and arm or clear and disarm the highest priority level currently in the active state.

If the initial Word Count of Status Stack Pointer Doubleword is equal to or greater than 28, a new environment is loaded from the memory stack. Twenty eight memory stack locations are accessed in a descending sequence, starting at a location having an address equal to the initial TSA (part of the Status Stack Pointer Doubleword). The hardware selects and loads the contents of 20 memory locations into the general registers and as the PSWs (i.e., the contents of locations having relative addresses of initial TSA-2, initial TSA-3, and initial TSA-12 through initial TSA-27). The contents of 10 memory stack locations (having relative addresses equal to initial TSA, initial TSA-1, and initial TSA-4 through initial TSA-11) are ignored.

Portions of the new PSWs are dependent upon the LP flag (bit 8) of the PLS instruction as well as the interrupt group inhibit bits of the old PSWs and the PSWs as pulled from the memory stack. If the LP flag is a 1, a new Register Block Pointer (as pulled from the memory stack) is loaded as part of the new PSWs. If the LP flag is a 0, the old Register Block Pointer is retained as the Register Block Pointer for the new PSWs. The new interrupt group inhibit bits (CI,

II, EI) are generated by "ORing" the old CI, II, EI bits with the contents of bits 37, 38, and 39 of the PSWs as pulled from the memory stack.

The clearing and arming or disarming the highest priority interrupt level currently active is dependent upon the coding of the CL and AD flags (bit positions 10 and 11, respectively) of the PLS instruction. If the CL flag is a 0, the interrupt level is not affected. If the CL flag is a 1 and the AD flag is a 0, the interrupt level is set to the disarmed state. If the CL flag is a 1 and the AD flag is a 1, the interrupt level is set to the armed state. Note that if the interrupt level is to be modified (CL flag is set to a 1), the instruction may be delayed until the interrupt system is available.

Summary description of CL and AD flags and effect on interrupt level and PDF flag follows:

Bit Positions 10 (CL)	11 (AD)	Function
0	0	No effect upon interrupt level or PDF flag.
0	1	Reset PDF flag
1	0	Clear and disarm interrupt level
1	1	Clear and arm interrupt level

If the initial Word Count is zero, default PSWs are loaded from real memory locations 2 and 3 and the other parameters of the Status Stack Pointer Doubleword are not effective and no parameters are affected.

Portions of the new PSWs (interrupt inhibit group bits and the Register Block Pointer) may be selected or generated in the following manner:

If the LP flag (bit 8) of the PSL instruction is a 1, the new Register Block Pointer will be as obtained from the default PSWs. If the LP flag is a 0, the Register Block Pointer of the old PSWs is retained as the Register Block Pointer for the new PSWs.

The CI, II, and EI bits of the old PSWs are "ORed" with the contents of bit positions 37, 38, and 39 of the default PSWs to generate the CI, II, and EI bits of the new PSWs.

Depending upon the coding of the CL and AD flags (bit positions 10 and 11, respectively) of the PLS instruction, the highest priority interrupt level currently in the active state may be modified. If the CL flag is a 0, the interrupt level is not affected. If the CL flag is a 1 and the AD flag is a 0, the interrupt level is cleared and placed into the disarmed state. If the CL flag is a 1 and the AD flag is a 1, the interrupt level is cleared and placed into the armed state. Note that if the interrupt level is to be modified (i.e., the CL flag is a 1), the instruction may be delayed until the interrupt system is available.

A summary description of the action on the interrupt level as a function of the CL and AD flag is as follows:

Bit Positions 10 (CL)	11 (AD)	Function
0	0	No effect upon interrupt level or PDF flag
0	1	Reset PDF flag
1	0	Clear and disarm interrupt level
1	1	Clear and arm interrupt level

If the initial Word Count within the Status Stack Pointer Doubleword is less than 28 and not equal to 0, the basic processor traps to location X'4D' (instruction exception trap) without loading any new status or affecting the parameters of the Status Stack Pointer Doubleword and the TCC2 bit is set to 1.

Affected: If word count ≥ 28, (PSWs), CC, Status Stack Pointer Doubleword Interrupt System if $(I)_{10}=1$.

Traps: Instruction exception, if word count is less than 28 and not 0; nonexistent instruction if bit 0=1.

If word count = 0, (PSWs), CC, and Interrupt System, if $I_{(10)}=1$.

(PSWs) and CC

$ED_{0-3} \longrightarrow CC;$

$ED_{5-7} \longrightarrow FS, FZ, FN;$

$ED_{8} \longrightarrow MS;$

$ED_{9} \longrightarrow MM;$

$ED_{10} \longrightarrow DM;$

$ED_{11} \longrightarrow AM;$

$ED_{15-31} \longrightarrow IA;$

$ED_{32-35} \longrightarrow WK$

$ED_{37-39} \cup CI, II, EI \longrightarrow CI, II, EI$
(Note: "u" represents inclusive OR.)

$ED_{56-59} \longrightarrow RP$ only if $(I)_{8} = 1$

$ED_{60} \longrightarrow RA$

$ED_{61} \longrightarrow MA$

Note: If the word count ≥ 28, the effective doubleword (ED) is pulled from memory stack locations (relative addresses initial TSA-24 and initial TSA+1). If the word count=0, the ED is pulled from real memory locations 2 and 3.

Status Stack Pointer Doubleword: (Only if initial Word Count ≥ 28)

TSA-1 ⟶ TSA until terminal TSA = initial TSA-28; Word Count - 1 ⟶ Word Count until terminal Word Count = initial Word Count - 28 (if initial Word Count > 32,767, bit 48 not affected); and,

Space Count + 1 ⟶ Space Count until terminal Space Count = initial Space Count + 28 (if Space Count > 32,767, then set bit 32 to 1).

Interrupt System:

If $(I)_{10} = 1$ and $(I)_{11} = 1$, clear and arm interrupt level.

If $(I)_{10} = 1$ and $(I)_{11} = 0$, clear and disarm interrupt level.

EXECUTE/BRANCH INSTRUCTIONS

The following instructions can cause the basic processor to execute instructions in an order other than that of sequentially ascending instruction addresses:

Instruction Name	Mnemonic
Execute	EXU
Branch on Conditions Set	BCS
Branch on Conditions Reset	BCR
Branch on Incrementing Register	BIR
Branch on Decrementing Register	BDR
Branch and Link	BAL

The EXECUTE instruction can be used to insert another instruction into the program sequence, and the branch instructions can be used to alter the program sequence, either unconditionally or conditionally. If a branch is unconditional (or conditional and the branch condition is satisfied), the instruction pointed to by the effective address of the branch instruction is normally the next instruction to be executed. If a branch is conditional and the condition for the branch is not satisfied, the next instruction is normally taken from the next location, in ascending sequence, after the branch instruction.

NONALLOWED OPERATION TRAP DURING EXECUTION OF BRANCH INSTRUCTION

The next instruction after a branch instruction may reside in two possible places: the location following the branch instruction or a location designated by the branch instruction. Either of these two locations may be in a protected memory region or in a region that is physically nonexistent. The execution of the branch does not cause a trap unless the instruction that is actually to follow the branch instruction is in a protected or nonexistent memory region. Traps do not occur because of any anticipation on the part of the hardware.

A nonallowed operation trap condition during execution of a branch instruction will occur for the following reasons:

1. The branch instruction is indirectly addressed and the branch conditions are satisfied, but the address of the location containing the direct address is either nonexistent or unavailable for read access to the program in the slave mode.

2. The branch instruction is unconditional (or the branch is conditional and the condition for the branch is satisfied), but the effective address of the branch instruction is either nonexistent or unavailable for instruction or read access to the program (in slave or master-protected mode).

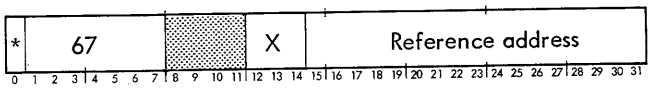
If either of the above situations occurs, the basic processor aborts execution of the branch instruction and executes a nonallowed operation trap.

Prior to the time that an instruction is accessed from memory for execution, bit positions 15-31 of the program status words contain the virtual address of the instruction, referred to as the instruction address. At this time, the basic processor traps to location X'40' if the actual address of the instruction is nonexistent or instruction-access protected. If the instruction address is existent and is not instruction-access protected, the instruction is accessed and the instruction address portion of the program status words is incremented by 1, so that it now contains the virtual address of the next instruction in sequence (referred to as the updated instruction address).

If a trap condition occurs during the execution sequence of any instruction, the basic processor decrements the updated instruction address by 1 and then traps to the location assigned to the trap condition. If neither a trap condition nor a satisfied branch condition occurs during the execution of an instruction, the next instruction is accessed from the location pointed to by the updated instruction address. If a satisfied branch condition occurs during the execution of a branch instruction (and no trap condition occurs), the next instruction is accessed from the location pointed to by the effective address of the branch instruction.

In the real extended addressing mode, a 20-bit address may be used as a branch address via indexing or indirect addressing. If such a branch address, (A), is beyond the first 128K of real memory, the instruction at (A) will be executed, but the next instruction address will be (A+1) in the original 128K block unless (A) contains a branch instruction. Note that with this exception all instructions executed in the real extended addressing mode must lie in the first 128K of real memory.

EXU EXECUTE
(word index alignment)



EXECUTE causes the basic processor to access the instruction in the location pointed to by the effective address of EXU and execute the subject instruction. The execution of the subject instruction, including the processing of trap and interrupt conditions, is performed exactly as if the subject instruction were initially accessed instead of the EXU instruction. If the subject instruction is another EXU, the basic processor executes the subject instruction pointed to by the effective address of the second EXU as described above. Such "chains" of EXECUTE instructions may be of any length, and are processed (without affecting the updated instruction address) until an instruction other than EXU is encountered. After the final subject instruction is executed, instruction execution proceeds with the next instruction in sequence after the initial EXU (unless the subject instruction is an LPSD or XPSD instruction, or is a branch instruction and the branch condition is satisfied).

If an interrupt activation occurs between the beginning of an EXU instruction (or chain of EXU instructions) and the last interruptible point in the subject instruction, the BP processes the interrupt-servicing routine for the active interrupt level and then returns program control to the EXU instruction (or the initial instruction of a chain of EXU instructions), which is started anew. Note that a program is interruptible after every instruction access, including accesses made with the EXU instruction, and the interruptibility of the subject instruction is the same as the normal interruptibility for that instruction.

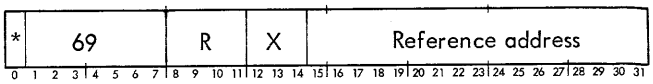
If a trap condition occurs between the beginning of an EXU instruction (or chain of EXU instructions) and the completion of the subject instruction, the basic processor traps to the appropriate trap location. The instruction address stored by the XPSD instruction in the trap location is the address of the EXU instruction (or the initial instruction of a chain of EXU instructions).

Affected: Determined by subject instruction

Traps: Determined by subject instruction

Condition code settings: Determined by subject instruction.

BCS BRANCH ON CONDITIONS SET
(Word index alignment)



BRANCH ON CONDITIONS SET forms the logical product (AND) of the R field of the instruction word and the current condition code. If the logical product is nonzero, the branch condition is satisfied and instruction execution proceeds with the instruction pointed to by the effective address of the BCS instruction. However, if the logical product is zero, the branch condition is unsatisfied and instruction execution then proceeds with the next instruction in normal sequence.

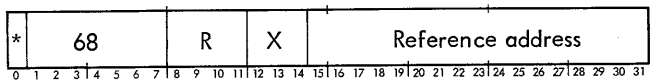
Affected: (IA) if CC $\cap$ R $\neq 0$

If CC $\cap$ $(I)_{8-11} \neq 0$, $EVA_{15-31} \longrightarrow IA$

If CC $\cap$ $(I)_{8-11} = 0$, IA not affected

If the R field of BCS is 0, the next instruction to be executed after BCS is always the next instruction in ascending sequence, thus effectively producing a "no operation" instruction.

BCR BRANCH ON CONDITIONS RESET
(Word index alignment)



BRANCH ON CONDITIONS RESET forms the logical product (AND) of the R field of the instruction word and the current condition code. If the logical product is zero, the branch condition is satisfied and instruction execution then proceeds with the instruction pointed to by the effective address of the BCR instruction. However, if the logical product is nonzero, the branch condition is unsatisfied and instruction execution then proceeds with the next instruction in normal sequence.

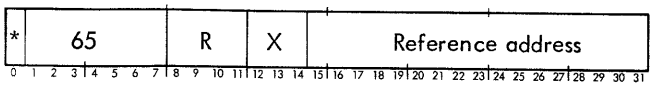
Affected: (IA) if CC $\cap$ R = 0

If CC $\cap$ $(I)_{8-11} = 0$, $EVA_{15-13} \longrightarrow IA$

If CC $\cap$ $(I)_{8-11} \neq 0$, IA not affected

If the R field of BCR is 0, the next instruction to be executed after BCR is always the instruction located at the effective address of BCR, thus effectively producing a "branch unconditionally" instruction.

BIR BRANCH ON INCREMENTING REGISTER
(Word index alignment)



BRANCH ON INCREMENTING REGISTER computes the effective virtual address and then increments the contents of general register R by 1. If the result is a negative value, the branch condition is satisfied and instruction execution then proceeds with the instruction pointed to by the effective address of the BIR instruction. However, if the result is zero or a positive value, the branch condition is not satisfied and instruction execution proceeds with the next instruction in normal sequence.

Affected: (R), (IA)

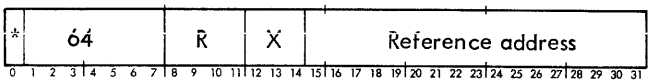
$(R) + 1 \longrightarrow R$

If $(R)_0 = 1$, $EVA_{15-31} \longrightarrow IA$

If $(R)_0 = 0$, IA not affected

If the branch condition is satisfied and if the effective address of BIR is either unavailable to the program (slave or master-protected mode) for instruction access or is nonexistent, the basic processor aborts execution of the BIR instruction and traps to location X'40'. In this case, the instruction address stored by the XPSD instruction in location X'40' is the virtual address of the aborted BIR instruction. If the basic processor traps because of instruction access protection, register R will contain the value that existed just before the BIR execution (i.e., updated instruction address). If a memory parity error occurs due to the accessing of the instruction to which the program is branching, the basic processor aborts execution of the BIR and traps to location X'4C' with register R unchanged.

BDR BRANCH ON DECREMENTING REGISTER
(Word index alignment)



BRANCH ON DECREMENTING REGISTER computes the effective virtual address and then decrements the contents of general register R by 1. If the result is a positive value, the branch condition is satisfied and instruction execution then proceeds with the instruction pointed to by the effective address of the BDR instruction. However, if the result is zero or a negative value, the branch condition is unsatisfied and instruction execution proceeds with the next instruction in normal sequence.

Affected: (R), (IA)

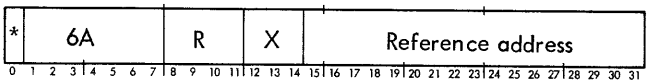
$(R) - 1 \longrightarrow R$

If $(R)_0 = 0$ and $(R)_{1-31} \neq 0$, $EVA_{15-31} \longrightarrow IA$

If $(R)_0 = 1$ and $(R) = 0$, IA not affected

If the effective address of BDR is unavailable to the program (slave or master-protected mode) for instruction access and the branch condition is satisfied, or if the effective address of BDR is nonexistent, the basic processor aborts execution of the BDR instruction and traps to location X'40'. In this case, the instruction address stored by the XPSD instruction in location X'40' is the virtual address of the aborted BDR instruction. If the basic processor traps because of instruction access protection, register R will contain the value that existed just before the BDR instruction. If a memory parity error occurs due to the accessing of the instruction to which the program is branching, the basic processor aborts execution of the BDR and traps to location X'4C' with register R unchanged.

BAL BRANCH AND LINK
(Word index alignment)



BRANCH AND LINK determines the effective virtual address, loads the updated instruction address (the virtual address of the next instruction in normal sequence after the BAL instruction) into bit positions 15-31 of general register R, clears bit positions 0-14 of register R to 0's and then replaces the updated instruction address with the effective virtual address. Instruction execution proceeds with the instruction pointed to by the effective address of the BAL instruction.

The BAL instruction in real extended addressing will store the full address of the next instruction in the specified R register. Positions 0-9 of the specified register will be set equal to zero.

Affected: (R), (IA)

$IA \longrightarrow R_{15-31}$; $0 \longrightarrow R_{0-14}$; $EVA_{15-31} \longrightarrow IA$

If the effective address of BAL is unavailable to the program (slave or master-protected mode) for instruction access and the branch condition is satisfied, or if the effective address of BAL is nonexistent, the basic processor aborts execution of the BAL instruction and traps to location X'40' (nonallowed operation trap). In this case, the instruction address stored by the XPSD instruction in location X'40' is the virtual address of the aborted BAL instruction. If the basic processor traps because of instruction access protection, register R will contain the updated instruction address. If a memory parity error occurs due to the accessing of the instruction to which the program is branching, the basic processor aborts execution of the BAL and traps to location X'4C' with register R changed to the updated instruction address.

CALL INSTRUCTIONS

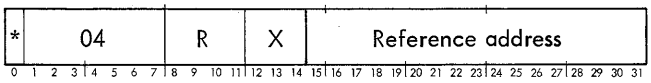
Each of the four CALL instructions causes the basic processor to trap to a specific location for the next instruction in sequence. The four CALL instructions, their mnemonics, and the locations to which the basic processor traps are:

Instruction Name	Mnemonic	Trap Location
CALL 1	CAL1	X'48'
CALL 2	CAL2	X'49'
CALL 3	CAL3	X'4A'
CALL 4	CAL4	X'4B'

Each of these four trap locations must contain an EXCHANGE PROGRAM STATUS WORDS (XPSD) instruction. Execution of XPSD in the trap location for a CALL instruction is described under "Control Instructions, XPSD Exchange Program Status Words". If the XPSD instruction is coded with bit position 9 set to 1, the next instruction (executed after the XPSD) is taken from one of 16 possible locations, as designated by the value in the R field of the CALL instruction. Each of the 16 locations may contain an instruction that causes the basic processor to branch to a specific routine; thus, the four CALL instructions can be used to enter any of as many as 64 unique routines.

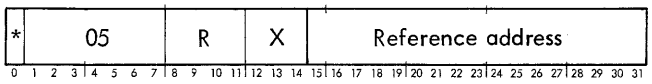
The effective address of either a direct or indirect CALL instruction is not used for a memory reference and, therefore, cannot cause a trap.

CAL1 CALL 1
(Word index alignment)



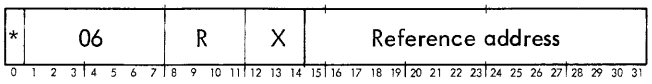
CALL 1 causes the basic processor to trap to location X'48'.

CAL2 CALL 2
(Word index alignment)



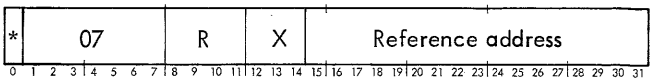
CALL 2 causes the basic processor to trap to location X'49'.

CAL3 CALL 3
(Word index alignment)



CALL 3 causes the basic processor to trap to location X'4A'.

CAL4 CALL 4
(Word index alignment)



CALL 4 causes the basic processor to trap to location X'4B'.

CONTROL INSTRUCTIONS

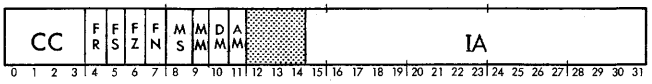
The following privileged instructions are used to control the basic operating conditions of the basic processor:

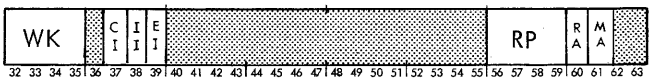
Instruction Name	Mnemonic
Load Program Status Words	LPSD
Exchange Program Status Words	XPSD
Load Register Pointer	LRP
Move to Memory Control	MMC
Load Real Address	LRA
Load Memory Status	LMS
Wait	WAIT
Read Direct	RD
Write Direct	WD

If execution of any control instruction is attempted while the basic processor is in the slave mode (i.e., while bit 8 of the current program status words is a 1), the basic processor unconditionally traps to location X'40' prior to executing the instruction.

PROGRAM STATUS WORDS

Program status words have the following structure when stored in memory:



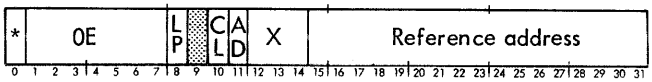


Bit Position	Designation	Function
0-3	CC	Condition code
4	FR	Floating round
5	FS	Floating significance mask
6	FZ	Floating zero mask
7	FN	Floating normalize mask

Bit Position	Designation	Function
8	MS	Master/slave mode control
9	MM	Memory map mode control
10	DM	Decimal arithmetic trap mask
11	AM	Fixed-point arithmetic overflow trap mask
15-31	IA	Instruction address
32-35	WK	Write key
37	CI	Counter interrupt group inhibit
38	II	I/O interrupt group inhibit
39	EI	External interrupt inhibit
56-59	RP	Register pointer
60	RA	Register altered
61	MA	Mode altered

The detailed functions of the various portions of the program status words are described in Chapter 2, "Program Status Words".

LPSD LOAD PROGRAM STATUS WORDS
(Doubleword index alignment, privileged)



LOAD PROGRAM STATUS WORDS replaces bits 0 through 39, 60 and 61 of the current program status words with bits 0 through 39, 60 and 61 of the effective doubleword.

Control bits used in the LPSD instruction are:

Bit Position	Designation	Control Function
8	LP	Load pointer control
10	CL	Clearing of interrupt level
11	AD	Armed/disarmed state

The following conditional operations are performed:

1. If bit position 8 (LP) of LPSD contains a 1, bits 56 through 59 (register pointer) of the current program status words are replaced by bits 56 through 59 of the effective doubleword; if bit 8 of LPSD is a 0, the current register pointer value remains unchanged.

2. If bit position 10 (CL) of LPSD contains a 1, the highest priority interrupt level currently in the active state is cleared (i.e., reset to either the armed state or the disarmed state); the interrupt level is armed if bit 11 (AD) of LPSD is a 1, or is disarmed if bit 11 of LPSD is a 0. If bit 10 of LPSD is a 0, no interrupt level is affected in any way, regardless of whether bit 11 of LPSD is 1 or 0. If bit 10 of the LPSD is a 0 and bit 11 of LPSD is 1, the PDF flag is cleared. (Interrupt levels are described in detail in Chapter 2, "Interrupt System".)

Bit position 10 (CL)	11 (AD)	Function
1	0	Clear and disarm interrupt level.
1	1	Clear and arm interrupt level.
0	1	Clear PDF flag.
0	0	No control action.

3. The PDF flag is normally reset by the last instruction of a trap routine, which is an LPSD instruction having bit 10 equal to 0 and bit 11 equal to 1.

These portions of the effective doubleword that correspond to undefined fields in the program status words are ignored.

Affected: (PSWs), interrupt system if $(I)_{10} = 1$

$ED_{0-3} \longrightarrow$ CC; $ED_{5-7} \longrightarrow$ FS, FX, FN;

$ED_{8} \longrightarrow$ MS; $ED_{9} \longrightarrow$ MM;

$ED_{10} \longrightarrow$ DM; $ED_{11} \longrightarrow$ AM;

$ED_{15-31} \longrightarrow$ IA; $ED_{32-35} \longrightarrow$ WK;

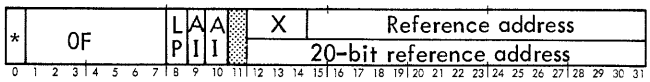
$ED_{37-39} \longrightarrow$ CI, II, EI; if $(I)_{8} = 1$, $ED_{56-59} \longrightarrow$ RP

$ED_{60} \longrightarrow$ RA; $ED_{61} \longrightarrow$ MA

If $(I)_{10} = 1$ and $(I)_{11} = 1$, clear and arm interrupt

If $(I)_{10} = 1$ and $(I)_{11} = 0$, clear and disarm interrupt

XPSD EXCHANGE PROGRAM STATUS WORDS
(Doubleword index alignment, privileged)



EXCHANGE PROGRAM STATUS WORDS stores the currently active PSWs in the doubleword location addressed by the effective address of the XPSD instruction. The following doubleword is then accessed from memory and loaded into the active PSWs registers.

The XPSD instruction is used for three distinct types of operations: as a normal instruction in an ongoing program; as an interrupt instruction; and as a trap instruction.

Control bits used in the XPSD instructions are:

Bit Position	Designation	Control Function	Where used
8	LP	Load pointer control	All XPSDs
9	AI	Address Increment	Trap XPSD
10	AT	Addressing type	All XPSDs

The effective address of an XPSD instruction is generated in one of the following ways:

XPSD (normal instruction)

When an XPSD instruction is encountered in the course of execution of normal programs, the AT (bit 10) of the instruction determines the type of addressing to be used.

If AT = 0, the reference address is 20 bits (12-31). Indexing is not allowed. Indirect addressing is allowed with the same constraints as the reference address. Addressing is always real, independent of the current PSWs.

If AT = 1, the reference address is 17 bits (15-31). Address calculations are according to standard addressing rules as determined by the current PSWs. Indexing and indirect addressing are allowed.

XPSD (interrupt instruction)

An XPSD instruction (in an interrupt location) executed as a result of an interrupt is called an interrupt instruction. The type of addressing to be used is determined by the basic processor mode and the AT (bit 10) of the instruction.

In the extended addressing mode (MA = 1 and MM = 0), the AT bit is used to determine the type of addressing to be used. If AT = 0, the reference address is 20 bits (12-31). Indexing is not allowed. Indirect addressing is allowed with the same constraints as the reference address. Addressing is always real, independent of the current PSWs. If AT = 1, the reference address is 17 bits (15-31). Address calculations are according to standard addressing rules as determined by the current PSWs. Indexing and indirect addressing are allowed.

When the addressing mode is not extended addressing, the reference address is 20 bits (12-31). If AT = 0, indexing is not allowed. Indirect addressing is allowed with the same constraints as the reference address. Addressing is always real, independent of the current PSWs. If AT = 1, the 20-bit reference address is subject to PSWs bit 9, as is the contents of the indirect address if indirect is specified.

XPSD (trap instruction)

An XPSD instruction (in a trap location) executed as a result of a trap entry operation is called a trap instruction. Addressing is the same as for the interrupt XPSD (see above).

The following additional operations are performed on the new program status words if, and only if, the XPSD is being executed as the result of a nonallowed operation (trap to location X'40') or a CALL instruction (trap to location X'48', X'49', X'4A', or X'4B'):

1. Nonallowed operations — the following additional functions are performed when XPSD is being executed as a result of a trap to location X'40':

   a. Nonexistent instruction — if the reason for the trap condition is an attempt to execute a nonexistent instruction, bit position 0 of the new program status words (CC1) is set to 1. Then, if bit 9 (AI) of XPSD is a 1, bit positions 15-31 of the new program status words (next instruction address) are incremented by 8.

   b. Nonexistent memory address — if the reason for the trap condition is an attempt to access or write into a nonexistent memory region, bit position 1 of the new program status words (CC2) is set to 1. Then, if bit 9 of XPSD is a 1, the instruction address portion of the new program status words is incremented by 4.

   c. Privileged instruction violation — if the reason for the trap condition is an attempt to execute a privileged instruction while the basic processor is in the slave mode, bit position 2 of the new program status words (CC3) is set to 1. Then, if bit position 0 of XPSD is 1, the instruction address portion of the new program status words is incremented by 2.

   d. Memory protection violation — if the reason for the trap condition is an attempt to read from or write into a memory region to which the program does not have proper access, bit position 3 of the new program status words (CC4) is set to 1. Then, if bit 9 of XPSD is a 1, the instruction address portion of the new program status words is incremented by 1.

   There are certain circumstances under which two of the above nonallowed operations can occur simultaneously. The following operation codes (including their counterparts) are considered to be both nonexistent and privileged: X'0C' and X'0D'. If either of these operation codes is used as an instruction while the basic processor is in the slave or master-protected mode, CC1 and CC3 are both set to 1's; if bit 9 of XPSD is a 1, the instruction address portion of the new program status words is incremented by 10. If an attempt is made to access or write into a memory region that is both nonexistent and prohibited to the program by means of the

memory control feature, CC2 and CC4 are both set to 1's; if bit 9 of XPSD is a 1, the instruction address of the new program status words is incremented by 5.

2. CALL instructions — the following additional functions are performed when XPSD is being executed as a result of a trap to location X'48', X'49', X'4A', or X'4B'.

   a. The R field of the CALL instruction causing the trap is logically inclusively ORed into bit positions 0-3 (CC) of the new PSWs.

   b. If bit position 9 of XPSD contains a 1, the R field of the CALL instruction causing the trap is added to the instruction address portion of the new PSWs.

3. Watchdog timer, parity error, or instruction exception trap — the following additional functions are performed when XPSD is being executed as a result of a trap to location X'46', X'4C', or X'4D', respectively.

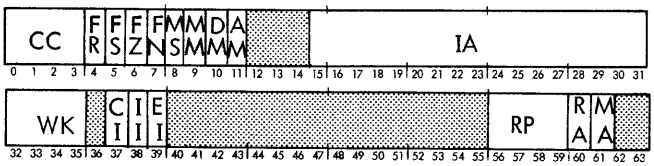
   a. The contents of TCC 1-4 are logically inclusively ORed into bit positions 0-3 (CC) of the new PSWs.

   b. If bit position 9 of XPSD contains a 1, the contents of TCC 1-4 are added to the instruction address portion of the new PSWs.

If bit position 9 of XPSD contains a 0, the instruction address portion of the new PSWs always remains at the value established by the second effective doubleword. Bit position 9 of XPSD is effective only if the instruction is being executed as the result of a nonallowed operation, CALL instruction watchdog timer, parity error, or instruction exception trap. Bit position 9 of XPSD must be coded with a 0 in all other cases; otherwise, the results of the XPSD instruction are undefined.

The current program status words are stored in the doubleword location pointed to by the effective address of XPSD in the following form:

Program Status Words



The current program status words (as illustrated above) are replaced by new program status words as described below.

1. The effective address of XPSD is incremented by 2 so that it points to the next doubleword location. The contents of the next doubleword location are referred to as the second effective doubleword, or ED2.

2. Bits 0-35, 60, and 61 of the current program status words are unconditionally replaced by bits 0-35, 60, and 61 of the second effective doubleword. The affected portions of the program status words are:

Bit Position	Designation	Function
0-3	CC	Condition code
4-7	FR, FS, FZ, FN	Floating control
8	MS	Master/slave mode control
9	MM	Mapping mode control
10	DM	Decimal arithmetic trap mask
11	AM	Fixed-point arithmetic trap mask
15-31	IA	Instruction address (real or virtual)
32-35	WK	Write key
60	RA	Register altered
61	MA	Mode altered

3. A logical inclusive OR is performed between bits 37 through 39 of the current program status words and bits 37 through 39 of the second effective doubleword.

Bit Position	Designation	Function
37	CI	Counter interrupt inhibit
38	II	I/O interrupt inhibit
39	EI	External interrupt inhibit

   If any (or all) of bits 37, 38, or 39 of the second effective doubleword are 0's, the corresponding bits in the current program status words remain unchanged; if any (or all) of bits 37, 38, or 39 of the second effective doubleword are 1's, the corresponding bits in the current program status words are set to 1's. See "Interrupt System", Chapter 2, for a detailed discussion of the interrupt inhibits.

4. If bit position 8 (LP) of XPSD contains a 1, bits 58 and 59 (register pointer) of the current program status words are replaced by bits 58 and 59 of the second effective doubleword; if bit 8 of XPSD is a 0, the current register pointer value remains unchanged.

Affected: (EDL), (PSWs)

If $(I)_{10} = 1$, trap or interrupt instructions only, effective address is subject to current active addressing mode.

If $(I)_{10} = 0$, trap or interrupt instructions only, effective address is independent of current active addressing mode.

PSD ⟶ EDL

$ED2_{0-3}$ ⟶ CC; $ED2_{4-7}$ ⟶ FR, FS, FZ, FN

$ED2_8$ ⟶ MS; $ED2_9$ ⟶ MM

$ED2_{10}$ ⟶ DM; $ED2_{11}$ ⟶ AM; ED_{15-31} ⟶ IA

$ED2_{32-35}$ ⟶ WK

$ED2_{37-39}$ ∪ CI, II, EI ⟶ CI, II, EI

If $(I)_8 = 1$, $ED2_{56-59}$ ⟶ RP

If $(I)_8 = 0$, RP not affected

$ED2_{60}$ ⟶ RA

$ED2_{61}$ ⟶ MA

If nonexistent instruction, 1 ⟶ CC1 then, if $(I)_9 = 1$, IA + 8 ⟶ IA

If nonexistent memory address, 1 ⟶ CC2 then, if $(I)_9 = 1$, IA + 4 ⟶ IA

If privileged instruction violation, 1 ⟶ CC3 then, if $(I)_9 = 1$, IA + 2 ⟶ IA

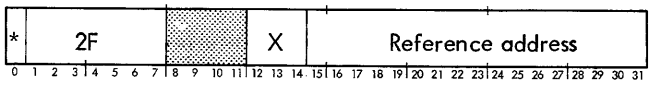
If memory protection violation, 1 ⟶ CC4 then, if $(I)_9 = 1$, IA + 1 ⟶ IA

If CALL instruction, CC ∪ $CALL_{8-11}$ ⟶ CC then, if $(I)_9 = 1$, IA + $CALL_{8-11}$ ⟶ IA

If $(I)_9 = 0$, IA not affected

If watchdog timer, parity error, or instruction exception trap, $ED2_{0-3}$ ∪ TCC1-4 ⟶ CC1-4 then, if $(I)_9 = 1$, IA + TCC1-4 ⟶ IA

LRP LOAD REGISTER POINTER
(Word index alignment, privileged)



LOAD REGISTER POINTER loads bits 24-27 of the effective word into the register pointer (RP) portion of the current program status words. Bit positions 0 through 23 and 28 through 31 of the effective word are ignored, and no other portion of the program status words is affected. If the LOAD REGISTER POINTER instruction attempts to load the register pointer with a value that points to a nonexistent block of general registers, the basic processor traps to location X'4D'.

Affected: RP Trap: Instruction exception

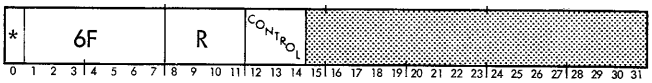
EW_{24-27} ⟶ RP

MOVE TO MEMORY CONTROL INSTRUCTIONS

The following instructions may be used to selectively move a string of control words from a control image area to specified memory control registers:

Instruction Name	Mnemonics
Move to Memory Control	MMC
Load Map (8-bit format)	LMAP
Load Map (11-bit format)	LMAPRE
Load Protection Code	LPC
Load Locks (2-bit format)	LLOCKS
Load Locks (4-bit format)	LLOCKSE

MMC MOVE TO MEMORY CONTROL
(Word index alignment, privileged, continue after interrupt)



The MMC instruction may be used to perform any move to memory control operation. Depending upon the type and format of the control image, the move to memory control operation may be performed either by an MMC instruction with a specific value in the control field (bit position 12-14) or by a special purpose instruction (i.e., LMAP, LMAPRE, LPC, LLOCKS, or LLOCKSE), as shown below:

Control Field of MMC instruction: Bit positions			Type and format of control image to be loaded	Alternate Instruction Mnemonic
12	13	14		
0	0	1	Memory write protection locks (2-bit format)	LLOCKS
0	1	1	Memory write protection locks (4-bit format)	LLOCKSE
0	1	0	Access protection (always 2-bit format)	LPC
1	0	0	Memory map (8-bit format)	LMAP
1	0	1	Memory map (11-bit format)	LMAPRE

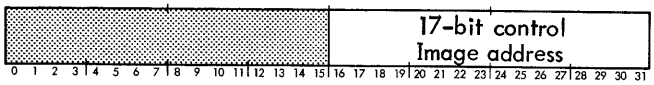
Attempting to execute an MMC instruction with any code other than the five shown above causes the instruction to trap to location X'4D' (instruction exception trap).

Normally, bit positions 15-31 may be ignored insofar as the operation of the MMC instruction is concerned. The results of the instruction are the same whether MMC is indirectly or directly addressed. However, if MMC is indirectly addressed and the indirect reference address is nonexistent, the nonallowed operation trap (location X'40') is activated.

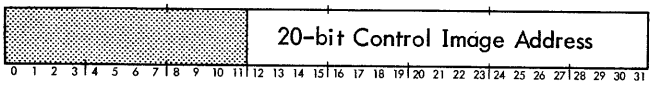
The R field, which must be coded with an even value, designates an even-odd pair of general registers (R and Ru1) that contain additional control information required by the MMC instruction. If the R field is coded with an odd value a trap to location X'4D' (instruction exception trap) occurs.

Depending upon the type of addressing, the contents of register R may be as follows:

If MA = 0, contents of register R are:



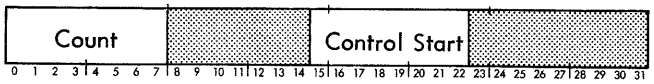
If MA = 1 and MM = 0, the contents of register R are:



In either case, the Control Image Address is the virtual address of a control word within the control image area to be loaded into a block of memory control registers, as specified by the contents of register Ru1.

Depending upon the type of control image being loaded, the contents of register Ru1 may be in one of the following three formats:

For loading memory map image (either 8-bit or 11-bit format), contents of register Ru1 are:



For loading 4-bit write lock images, contents of register Ru1 are:

For loading access protection or 2-bit write lock images, contents of register Ru1 are:

The Count field (bit positions 0-7) specifies the number of words to be loaded from the control image area. If the initial word count is zero, a word count of 256 is implied.

The Control Start field (bit positions 15-20, 21, or 22) points to the beginning of the memory region controlled by the registers to be loaded. The significance of this field is different for the 5 modes of MMC operations and is described within each mode below.

Affected: (R), (Ru1), memory control storage

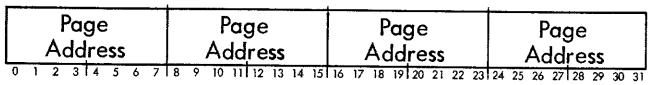
Traps: Instruction exception, nonallowed operation.

LOADING THE MEMORY MAP

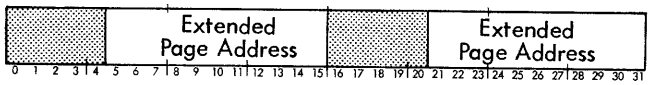
CONTROL IMAGE

Each word of the memory map control image contains either four 8-bit page addresses or two 11-bit extended page addresses, as illustrated below:

Typical memory map control image word (8-bit format):

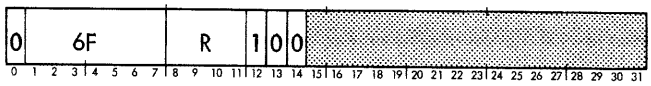


Typical memory map control image word (11-bit format):

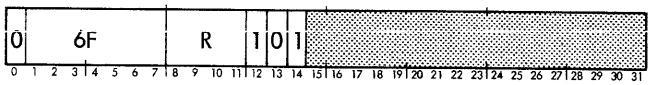


Depending upon the memory map control image format, the instruction format is one of the following:

LMAP LOAD MAP (8-bit format)

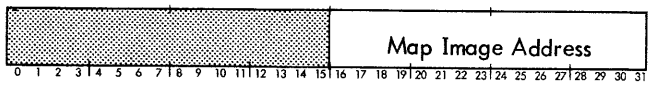


LMAPRE LOAD MAP REAL EXTENDED (11-bit format)

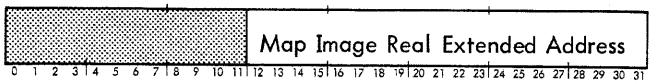


Depending upon the type of addressing, the format of register R contents is one of the following:

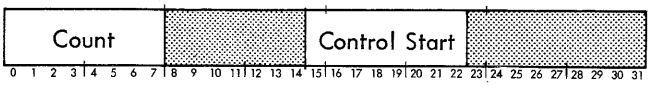
If MA = 0;



If MA = 1 and MM = 0;



For either memory map format and either type of addressing, the contents of register Ru1 are:



MEMORY MAP LOADING PROCESS

The initial map image address (in register R) is the virtual address of the first word of the memory map control image.

The initial count, as contained in register Ru1 specifies the word length of the control image to be loaded. A word count of 64 (for 8-bit format) or 128 (for 11-bit format) is sufficient to load an entire block of 256 memory map control registers. The memory map control registers are treated as a circular set, with the first register following the last; thus, a word count greater than 64 (8-bit format) or 128 (11-bit format) causes the first registers to be overwritten.

The initial value of the control start field of register Ru1 points to the first page (512 words) of virtual addresses that are to be controlled by the memory map control image being loaded. The memory map control image is loaded into the memory map control registers one word at a time. As the contents of each word are loaded into either two or four memory map control registers, the map image address is incremented by 1, the word count is decremented by 1, and the value in the control start field is incremented either by four (if the memory map control image is in the 8-bit format) or by two (if the memory map control image is in the 11-bit format). The loading process continues until the word count is reduced to zero.

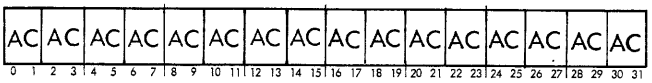
When the load process is completed, the map image address of register R contains a value equal to the sum of the initial map image address plus the initial word count, the word count of register Ru1 has a value of zero, and the control start field of register Ru1 contains a value equal to the sum of the initial contents plus four or two times the initial word count.

LOADING THE ACCESS PROTECTION CONTROLS

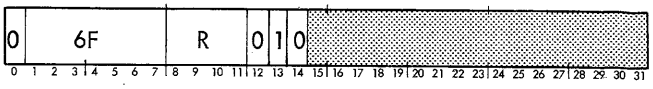
CONTROL IMAGE

Each access protection control image word contains sixteen 2-bit fields; or, the access protection codes for 16 consecutive pages of virtual memory. Thus, the access protection control image for 128K word (256 page) virtual memory is contained within 16 contiguous memory locations, designated as the access protection control image area.

The format of a typical access protection control image word is:

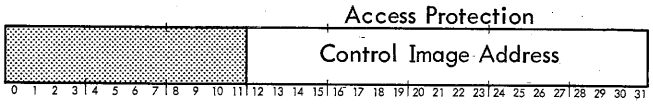


The instruction format for loading the access protection code is:

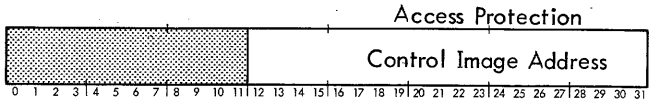


Depending upon the type of addressing, the format of register R contents is one of the following:

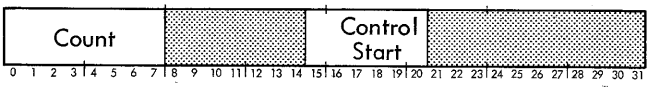
If MA = 0;



If MA = 1 and MM = 0;



For either type of addressing, the contents of register Ru1 are:



ACCESS PROTECTION LOADING PROCESS

The initial access protection control image address in register R is the virtual address of the first word of the access protection control image.

The initial count in register Ru1 specifies the word length of the control image to be loaded. A word count of 16 is sufficient to load the entire block of 256 access protection control registers. The access protection control registers are treated as a circular set, with the first register following the last; thus, a word count greater than 16 causes the first registers loaded to be overwritten.

The initial value of the control start field of register Ru1 points to the first page (512 words) of virtual addresses that are to be controlled by the access protection control image being loaded. The access protection control image is loaded into the access control registers one word at a time, thus loading the control registers for 16 consecutive pages with the contents of each image word. As each image word is loaded, the access protection control image address is incremented by 1, the word count is decremented by 1, and the value in the control start field is incremented by 4. The loading process continues until the word count is reduced to 0.

When the loading process is completed, the parameters contained within registers R and Ru1 have the following values:

Access protection control image address = initial access protection control image address plus the initial word count.

Count = 0.

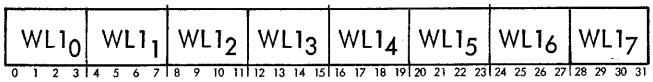
Control Start = initial contents plus 4 times the initial word count.

MEMORY WRITE PROTECTION LOCKS

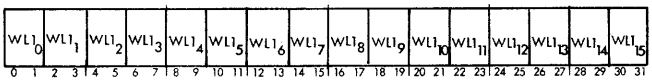
CONTROL IMAGE

Each write lock control image word may contain either eight 4-bit write lock images or sixteen 2-bit write lock images, as illustrated below:

Typical write locks image word (4-bit format);



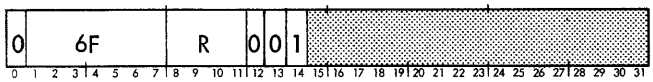
Typical write locks image word (2-bit format);



The number of words required to define the memory write locks control image is dependent upon the format of the write lock images and the number of write lock registers to be loaded by a single MMC instruction. (For example, if the write lock images are of the 4-bit format and the memory system is maximum size (1,024,000 words or 2048 pages) with 2048 write lock control registers, the control image may be defined by 256 words (i.e., 256 words times 8 write lock images per word is equal to 2048 write lock images or one write lock image per each write lock control register). If the write lock images are of the 2-bit format and the memory size is the same, as described above, the control image may be defined by 128 words.

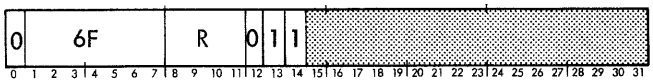
The instruction format for loading 2-bit write lock images is:

LLOCKS LOAD LOCKS (2-bit format)

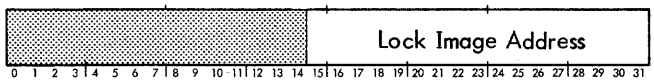


The instruction format for loading 4-bit write lock images is:

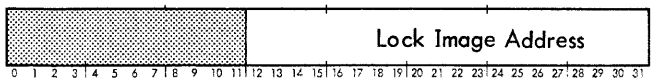
LLOCKSE LOAD LOCKS (4-bit format)



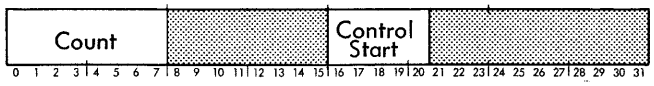
If MA = 0, the contents of register R are:



If MA = 1 and MM = 0, the contents of register R are:



When loading 2-bit write lock images, the contents of register Rul are:



When loading 4-bit write lock images, the contents of register Rul are:

LOADING PROCESS

Depending upon the addressing mode of the basic processor, the contents of register R are interpreted as either a 17-bit or a 20-bit virtual address of an image word within the memory write locks control image area (source of write lock images). The initial lock image address points to the first image word. After the contents of the image word (either 8 or 16 write lock images) are loaded into an equivalent number of write lock registers, the lock image address is incremented by one. Thus, successive image words are accessed in an ascending sequence.

Depending upon the instruction format, the hardware appends either one or two low order zeros, as necessary, to convert the 9-bit or 10-bit control start field into an 11-bit real page address. In addition to being the real page address of 512 consecutive memory word locations, the value of the 11-bit control start field is also the address of the associated write lock control register. The value of the control start field at the time the image word is accessed is the address of the first of either 8 or 16 write lock control registers that will be loaded by the write lock images contained within one image word. When all of the write lock images of a given word have been loaded into either 8 or 16 write lock control registers, the value of the 9-bit or 10-bit control start field is incremented by 4. (Note that this is equivalent to incrementing the value of the effective 11-bit field by a value of either 8 or 16, the number of control registers loaded.)

The count field of register Rul specifies the number of image words, and indirectly the number of write lock images to be loaded. Depending upon the instruction format, each image word is interpreted as containing either eight 4-bit write lock images or sixteen 2-bit write lock images. In the case of 2-bit write lock images, the hardware appends two high order zeros to each image as it is loaded into the 4-bit control register. Thus, the number of write lock control registers loaded is always either 8 or 16 times the initial value of the count field. If the initial value of the count field is zero, it is interpreted to be 256 words. During the loading operation, the count field is decremented by one after the contents of each image word are loaded into the appropriate number of control registers. The loading operation continues until the word count is reduced to zero. At that time, the value of the lock image address is equal to its

initial value plus the initial value word count and the value of the 9- or 10-bit control start field is equal to its initial value plus 4 times the initial word count.

The memory write lock registers are treated as a circular set, with the register for memory addresses X'0'-X'1FF' (first page) immediately following the register for memory addresses X'FFE00'-X'FFFFF' (last page). Overwriting the first registers occurs when 2-bit write lock images are being processed and the word count is greater than 128.

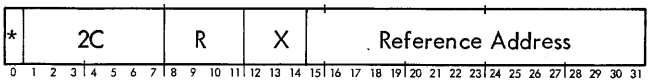
INTERRUPTION OF MMC

The execution of MMC can be interrupted after each word of the control image has been moved into the specified control register. Immediately prior to the time that the instruction in the interrupt or trap location is executed, the instruction address portion of the program status words contains the virtual address of the MMC instruction, register R contains the virtual address of the next word of the control image to be loaded, and register Ru1 contains a count of the number of control image words remaining to be moved and a value pointing to the next memory control register to be loaded. After interrupt, the MMC instruction may be resumed from the point it was interrupted.

MEMORY ACCESS TRAPS BY MMC INSTRUCTION

A trap during execution of the MMC instruction can occur if the pages containing the control images are nonexistent or are protected in the master-protected mode. The registers R and Ru1 may be altered for the above case. If a parity error should occur during access of a control image word, the MMC instruction will trap with the Register Altered indicator set indicating that a change has been made to the memory control registers. The registers R and Ru1 will be restored to their initial values, prior to the point at which the trap occurred.

LRA LOAD REAL ADDRESS
(Word index alignment, privileged)



LOAD REAL ADDRESS converts the address portion of the effective word into a real byte, halfword, word, or doubleword address (as specified by CC1 and CC2 at the beginning of the LRA instruction) and loads that real address and status information (as listed below) into register R. Upon completion of the LRA instruction, additional information pertaining to the LRA instruction or to the real address is provided via the condition code.

Prior to executing an LRA instruction, CC1 and CC2 must be set to an appropriate value (as shown below).

CC1	CC2	Type of real address to be generated
0	0	Byte (22 bits)
0	1	Halfword (21 bits)
1	0	Word (20 bits)
1	1	Doubleword (19 bits)

The effective virtual address for the LRA instruction itself may be generated in a normal manner (i.e., indirect addressing, indexing, and/or mapping, as applicable, may be specified and performed) with all standard trapping conditions in effect.

The address loaded into the R register is dependent upon the value of the address portion of the effective word. If the address portion of the effective word is equal to or greater than 16, it is converted (mapped) into a 19, 20, 21, or 22-bit real address, as specified by CC1 and CC2.

Note: Converting an effective virtual address into a real address by mapping is performed independently of the state of the map bit in the current PSWs.

If the address portion of the effective word is less than 16, it is not mapped into a real address. Instead, a 19, 20, 21, or 22-bit effective virtual address is generated, as specified by CC1 and CC2.

In either case a 19, 20, 21, or 22-bit real or effective virtual address is loaded into a corresponding number of low order bit positions of the R register (i.e., the least significant bit of the address is always loaded into bit position 31 of register R). Except for bit positions reflecting status information, all high order bit positions within register R are set to zero. Contents of the various bit positions of register R after an LRA instruction are as follows:

Bits	Contents
0-9	Reserved; always set to 0.
10-31	Real or effective virtual address. For 21-, 20-, and 19-bit addresses, as specified by initial value of CC1 and CC2, bit positions 10, 11, and 12 will be set to zeros, as required.

Affected: (R),CC

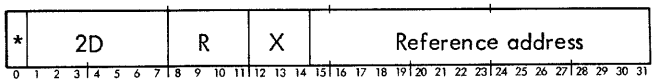
Condition code settings:

1	2	3	4	Results in R register
0	0	-	-	No abnormal condition.
1	1	-	-	Address in R is real but for a nonexistent memory location.

1 2 3 4	Results in R register
1 1 0 0	Address in R is an effective virtual address (address of a general register).
	Note: Condition code setting 11-- and 1100 may be distinguished in the software by examining the address (bits 10-31).
- - 0 0 - - 0 1 - - 1 0 - - 1 1	Access protect code for the page containing the memory location specified by the generated address.

Note: This instruction requires two memory references to the same location for its execution. To preclude other processors from accessing the effective location during this time, the memory unit containing the effective location is reserved (not accessible to other processors) until the LRA instruction is completed.

LMS LOAD MEMORY STATUS
(Word index alignment, privileged)



LOAD MEMORY STATUS is used to determine memory unit status and/or to perform diagnostic action on a memory unit. The effective address is used to determine the memory unit. The condition code setting immediately before execution determines the diagnostic action to be performed. The effective address always references memory even if it is less than 16. The condition code can be set to the desired value before execution of LMS with the LCF or LCFI instructions. Register R is loaded with the result of the action. The condition code is set at the conclusion of execution to reflect the status of the word loaded (if any).

Affected: (R),CC

Trap: See "Trap System", Chapter 2.

Initial condition code settings:

1 2 3 4	LMS Action
0 0 0 0	Read and set — causes the same action as the LOAD AND SET (LAS) instruction, except for condition code settings. Normal traps are allowed including write protect.
0 0 0 1	Read and inhibit parity — loads the effective word into R. If a memory parity error is detected, the memory does not take a "snapshot" or generate a Memory Fault Interrupt (MFI).
	It does, however, generate the Memory Parity Error signal. The basic processor inhibits the trap that would ordinarily occur for the memory parity error.
0 0 1 0	Clear memory — stores zero in the memory location specified by the address.
0 0 1 1	Reserved.
0 1 0 0	Reserved.
0 1 0 1	Reserved.
0 1 1 0	Read write lock — loads a pair of 4-bit write locks into byte 3 of R (bits 24-31) and 0 in all other bit positions of R. The write lock stored in bits 24-27 is stored in the memory system's Write Lock memory at the location corresponding to bits 17-21 of the effective address, bit 22=0. The write lock stored in bits 28-31 corresponds to bits 17-21 of the effective address, bit 22=1.
0 1 1 1	Write write lock — stores byte 3 of the data word sent to memory as a pair of write locks in the memory system's Write Lock memory at a location corresponding to bits 17-21 of the effective address, bit 22=0 (for data bits 24-27) and bits 17-21 of the effective address, bit 22=1 (for data bits 28-31).
1 0 0 0	Read status word 0† — loads status word 0 into R (see Table 9).
1 0 0 1	Reserved.
1 0 1 0	Read status word 1† — loads status word 1 into R (see Table 10).
1 0 1 1	Reserved.
1 1 0 0	Read status word 0 and clear.
1 1 0 1	Reserved.
1 1 1 0	Write double error — stores an arbitrary word into a specified memory location, with two differences compared to a normal Write Word instruction: (1) Byte 3 in memory is forced to zero; (2) the arbitrary word is stored in memory with an intentional wrong parity; on a subsequent read of that word, the memory issues the parity error signal.
1 1 1 1	Reserved.

Condition code settings after execution.

†Primarily of diagnostic concern.

Table 9. Status Word 0

Field	Bits	Comments
	0	Reserved
	1	Power status
	2-7	Memory unit error code
	8-9	Memory type
Ports	10	Port 1 enabled
	11	Port 2 enabled
	12	Port 3 enabled
	13	Port 4 enabled
	14	Port 5 enabled
	15	Port 6 enabled
	16	Port 1 serviced
	17	Port 2 serviced
	18	Port 3 serviced
	19	Port 4 serviced
	20	Port 5 serviced
	21	Port 6 serviced
Memory fault types	22	0
	23	Uncorrectable memory unit error
	24	Memory module selection error
	25	Address parity error
	26	Data in parity error
	27	Write lock parity error
	28	Port selection error
	29	Undefined operation
	30	Control error
	31	Multiple error

For "read and inhibit parity" operations, the status of the word loaded (if any) is stored in the condition code bits at the conclusion of execution as follows:

CC1: Memory Parity Error (from memory)

CC2: Data Bus Check (from CPU)

CC3: Parity Bit (from memory)

CC4: 0

Table 10. Status Word 1

Field	Bits	Comments
	0	Interleave switch ON
	1-3	Memory unit size: 000 8K 001 16K 010 24K 011 32K 100 40K 101 48K 110 56K 111 64K
	4-6	Memory unit number (binary code)
Starting Address	7	Starting address bit 12
	8	Starting address bit 13
	9	Starting address bit 14
	10	Starting address bit 15
	11	Starting address bit 16
	12	Starting address bit 17
	13	Starting address bit 18
	14	Reserved
	15-31	Address received, bits 15-31

WAIT WAIT
(Word index alignment, privileged)

WAIT causes the basic processor to cease all operations until an interrupt activation occurs, or until the operator puts the basic processor in the IDLE mode and then back to RUN (see Chapter 5). The instruction address portion of the PSWs is updated before the basic processor begins waiting; therefore, while it is waiting, the INSTRUCTION ADDRESS indicators contain the virtual address of the next location in ascending sequence after WAIT and the contents in the next location are displayed in the DISPLAY indicators on the processor control console. If any input/output operations are being performed when WAIT is executed, the operations proceed to their normal termination.

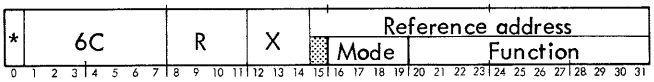
When an interrupt activation occurs while the basic processor is waiting, it processes the interrupt-servicing routine. Normally, the interrupt-servicing routine begins with an XPSD instruction in the interrupt location, and ends with an LPSD instruction at the end of the routine. After the LPSD instruction is executed, the next instruction to be executed in the interrupted program is the next instruction in sequence after the WAIT instruction. If the interrupt is to a

single-instruction interrupt location, the instruction in the interrupt location is executed and then instruction execution proceeds with the next instruction in sequence after the WAIT instruction. When the basic processor execution mode is changed from RUN mode to IDLE mode and back to RUN while the basic processor is waiting, instruction execution proceeds with the next instruction in sequence after the WAIT instruction.

Affected: PC

If WAIT is indirectly addressed and the indirect reference address is nonexistent, the nonallowed operation trap to location X'40' will not occur. The effective virtual address of the WAIT instruction, however, is not used as a memory reference (thus does not affect the normal operation of the instruction).

RD READ DIRECT
(Word index alignment, privileged)



The basic processor is capable of directly communicating with other elements of the system, as well as performing internal control operations, by means of the READ DIRECT/WRITE DIRECT (RD/WD) lines. The RD/WD lines consist of 16 address lines, 32 data lines, two condition code lines, and various control lines that are connected to various basic processor circuits and to special system equipment.

READ DIRECT causes bits 16 through 31 of the effective virtual address to be presented to other elements of the system on the RD/WD address lines. Bits 16-31 of the effective virtual address identify a specific element of the system that is expected to return information (two condition code bits plus a maximum of 32 data bits) to the basic processor. The significance and number of data bits returned depend on the selected element. If the R field of RD is nonzero, up to 32 bits of the returned data are loaded into general register R; however, if the R field of RD is 0, the returned data is ignored and general register 0 is not changed. Bits CC3 and CC4 of the condition code are set by the addressed element, regardless of the value of the R field.

Bits 16-19 of the effective virtual address of RD determine the mode of the RD instruction, as follows:

Bit Position

16 17 18 19	Mode
0 0 0 0	Internal basic processor control.
0 0 0 1	Interrupt control.
0 0 1 0	Xerox testers.
0 0 1 1 ⋮ 1 1 1 1	Unassigned.
1 1 1 1	Special systems control (for customer use with specially designed equipment).

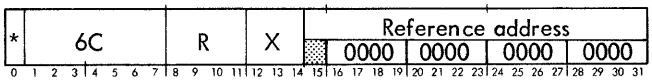
If bits 16-19 select mode 2 through mode F, CC1 and CC2 are set to zero and CC3 and CC4 are set according to the state of the two condition code lines from the external device.

READ DIRECT, INTERNAL BASIC PROCESSOR CONTROL (MODE 0)

In this mode, the basic processor is able to read the sense switches, the basic processor address, and the interrupt inhibit bits of the PSWs as follows:

READ SENSE SWITCHES

The following configuration of RD can be used to read the four SENSE switches in the System Control Processor:

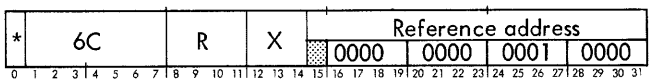


If a particular SENSE switch is set, the corresponding bit of the condition code is set to 1; if a SENSE switch is zero, the corresponding bit of the condition code is set to 0 (see "Read Sense Switches" in Chapter 5).

In this case, only the condition code is affected.

READ BASIC PROCESSOR

The following RD configuration is used to read the basic processor's address:



If the R field is nonzero, the cluster number in which the basic processor resides is obtained from the associated processor interface and loaded into register R bits 21-23. All other bits in the register are cleared to zero.

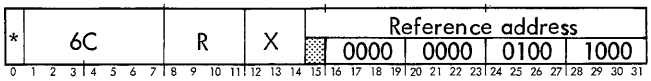
Affected: (R)

Cluster Address $\longrightarrow R_{21-23}$

$0 \longrightarrow R_{0-20}$ and R_{24-31}

READ INTERRUPT INHIBITS

The following configuration of RD can be used to read the contents of the interrupt inhibit field:



If the R field of RD is nonzero, the contents of the interrupt inhibit field (bits 37, 38, 39) of the program status words are transferred to the least significant 3 bits of the specified R register (bits 29, 30, 31). The remainder of the R register (bits 0-28) is cleared to zeros.

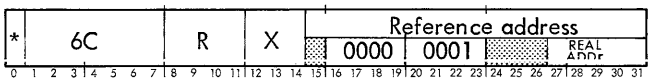
Affected: (R)

$(PSWs)_{37-39} \longrightarrow R_{29-31}$

$0 \longrightarrow R_{0-28}$

Note that a copy of the interrupt inhibits is retained in the Interrupt Status Register in the Processor Interface associated with each basic processor.

LOAD FROM LOW MAIN MEMORY



The instruction allows reading the contents of real memory locations 0-31 (locations 0-15 shadowed by the general purpose registers). This allows access to the Status Stack Pointer Doubleword in locations 0-1 and the default Program Status Words (Interrupt Stack is empty) in locations 2-4.

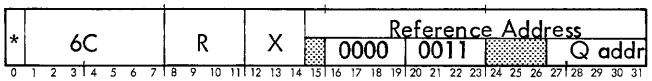
If the R field is nonzero, the contents of the main memory location identified by bits 27-31 are loaded into R.

Affected: (R)

EW $\longrightarrow$ R

READ INTERNAL CONTROL REGISTERS

The following configuration of RD is used to read the contents of internal control (or Q) registers:



If the R field of the RD instruction is nonzero, the contents of the internal control register, as specified by the "Q Address" field of the instruction (bit positions 27-31), are loaded into register R. Although the Q address field permits any of 32 addresses to be specified, only the following may be used:

Q Address	Contents
X'1D'	(Bits 0-13) - Reserved (Bits 14-31) - "Branch from" Program Counter
X'1E'	(Bits 0-7) - Reserved (Bits 8-31) - Load Device Address

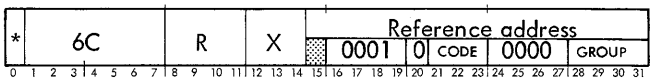
All other Q addresses from X'00' - X'1F' are reserved.

Affected: (R)

EW $\longrightarrow$ R

READ DIRECT, INTERRUPT CONTROL (MODE 1)

The following configuration of RD is used to control the sensing of the various states of the individual interrupt levels within the basic processor interrupt system:



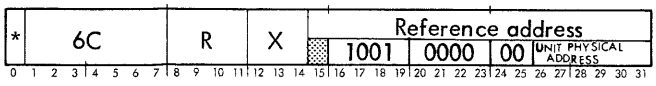
Bits 28 through 31 of the effective address specify the identification number of the group of interrupt levels to be controlled by the READ DIRECT instruction.

The R field of the RD instruction specifies a general register that will contain the bits sensed from the individual interrupt levels within a specified group. For external interrupt groups, bit position 16 of register R contains the appropriate indicator bit for the highest priority (lowest number) interrupt level within the group and bit position 31 of register R contains the indicator bit for the lowest priority interrupt level within the group. For assignments in Group X'0', see Table 11. Each interrupt level in the designated group is sensed according to the function code specified by bits 21 through 23 of the effective address of RD. The codes and their associated functions are as follows:

Code	Function
001	Read Armed or Waiting State. Set to 1 the bits in the selected register which correspond to interrupt levels in this group that are in either the armed or the waiting state. Reset all other bits to zero.
010	Read Waiting or Active State. Set to 1 the bits in the selected register which correspond to each interrupt level in this group that is in either the waiting or the active state. Reset all other bits to zero.
100	Read Enabled. Set to 1 the bits in the selected register which correspond to each interrupt level in this group which is enabled. Reset all other bits to zero.

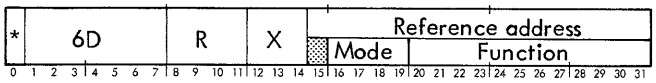
READ DIRECT (MODE 9)

READ CONFIGURATION CONTROL PANEL



The mode 9 instruction reads the state of the Configuration Control Panel for the addressed cluster or unit. Physical addresses are assigned at the time of system configuration. The returned status to Register R is shown in Tables 11 and 12.

WD WRITE DIRECT
(Word index alignment, privileged)



WRITE DIRECT causes bits 16-31 of the effective virtual address to be presented to other elements of the system on the RD/WD address lines (see READ DIRECT). Bits 16-31 of the effective virtual address identify a specific element of the system that is to receive control information from the basic processor. If the R field of WD is nonzero, the 32-bit contents of register R are transmitted to the specified element on the RD/WD data lines. If the R field of WD is 0, 32 0's are transmitted to the specified element (instead of the contents of register 0). The specified element may return information to set the condition code.

Bits 16-19 of the effective virtual address determine the mode of the WD instruction, as follows:

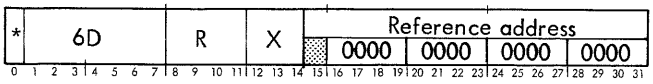
16 17 18 19	Mode
0 0 0 0	Internal basic processor control.
0 0 0 1	Interrupt control.
0 0 1 0	Xerox testers.
0 0 1 1 ⋮ 1 1 1 0	Unassigned.
1 1 1 1	Special systems control (for customer use with specially designed equipment).

If bits 16-19 select mode 2 through mode F, CC1 and CC2 are set to zero and CC3 and CC4 are set according to the state of the two condition code lines from the external device.

WRITE DIRECT, INTERNAL BASIC PROCESSOR CONTROL (MODE 0)

LOAD SENSE SWITCHES

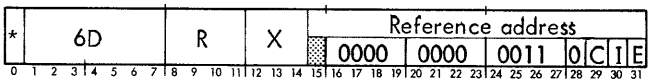
The following configuration of WD can be used to load the sense switches in the System Control Processor:



If the R field is nonzero, bits 0 through 3 of Register R will be loaded into sense switches 1 through 4 in the System Control Processor. If the R field is zero, sense switches will be reset to zeros. (See the section "System Control Panel" in Chapter 5.)

SET INTERRUPT INHIBITS

The following configuration of WD can be used to set the interrupt inhibits (bit positions 37-39 of the PSWs):

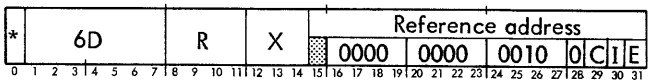


A logical inclusive OR is performed between bits 29-31 of the effective virtual address and bits 37-39 of the PSWs. If any (or all) of bits 29-31 of the effective virtual address are 1's, the corresponding inhibit bits in the PSWs are set to 1's; the current state of an inhibit bit is not affected if a corresponding bit position of the effective virtual address contains a 0.

Note that a copy of the Interrupt Inhibits is retained in the Interrupt Status Register in the Processor Interface associated with each basic processor.

RESET INTERRUPT INHIBITS

The following configuration of WD can be used to reset the interrupt inhibits:



If any (or all) of bits 29-31 of the effective virtual address are 1's, the corresponding inhibit bits in the PSWs are reset to 0's; the current state of an inhibit bit is not affected if a corresponding bit position of the effective virtual address contains a 0.

Note that a copy of the Interrupt Inhibits is retained in the Interrupt Status Register in the Processor Interface associated with each basic processor.

Table 11. Read Direct Mode 9 Status Word

RD Status Word Bit No.	Processor Cluster 1	Memory Unit 1
00	System Select	System Select
01	Clock Select	Clock Select
02	Processor Cluster Address 2^2	Unit No. 2^2
03	Processor Cluster Address 2^1	Unit No. 2^1
04	Processor Cluster Address 2^0	Unit No. 2^0
05	BP Enable	Port Enable 1
06	MIOP Enable	Port Enable 2
07	DIO Enable	Port Enable 3
08	Not Assigned	Port Enable 4
09	ALTSEL	Port Enable 5
10	FSELA	Port Enable 6
11	FSELBO	Not Assigned
12	FSELBI	Not Assigned
13	Real Time Clock 1-S0	Interleave Enable
14	Real Time Clock 1-S1	Starting Address S12
15	Real Time Clock 2-S0	Starting Address S13
16	Real Time Clock 2-S1	Starting Address S14
17	Real Time Clock 3-S0	Starting Address S15
18	Real Time Clock 3-S1	Starting Address S16
19	Subjective Time Clock -S0	Starting Address S17
20	Subjective Time Clock -S1	Starting Address S18
21	Not Assigned	Not Assigned
22	Not Assigned	Not Assigned
23	Not Assigned	Not Assigned
24	Not Assigned	Not Assigned
25	Not Assigned	Not Assigned
26	Not Assigned	Not Assigned
27	[t]Chassis Type-2^4	[t]Chassis Type-2^4
28	Chassis Type-2^3	Chassis Type-2^3
29	Chassis Type-2^2	Chassis Type-2^2
30	Chassis Type-2^1	Chassis Type-2^1
31	Chassis Type-2^0	Chassis Type-2^0

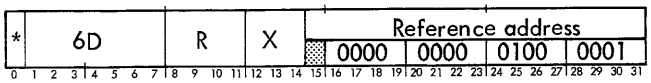
[t]See Chassis Type Table.

Table 12. Chassis Type Assignments

Chassis Type	2^4	2^3	2^2	2^1	2^0	Configuration Information
Processor Clusters	1	1	0	0	0	Reserved
	1	1	0	0	1	Reserved
	1	1	0	1	0	Processor Cluster Type 1
	1	1	0	1	1	Reserved
	1	1	1	0	0	Reserved
	1	1	1	0	1	Reserved
	1	1	1	1	0	Reserved
	1	1	1	1	1	Reserved
Controller Clusters	1	0	0	0	0	Reserved
	1	0	0	0	1	Reserved
	1	0	0	1	0	Reserved
	1	0	0	1	1	Reserved
	1	0	1	0	0	Reserved
	1	0	1	0	1	Reserved
	1	0	1	1	0	Reserved
	1	0	1	1	1	Reserved
Memory Units	0	1	0	0	0	Memory Unit Type 1
	0	1	0	0	1	Reserved
	0	1	0	1	0	Reserved
	0	1	0	1	1	Reserved
	0	1	1	0	0	Reserved
	0	1	1	0	1	Reserved
	0	1	1	1	0	Reserved
	0	1	1	1	1	Reserved
Reserved	0	0	0	0	0	Not available
	0	0	0	0	1	Reserved
	0	0	0	1	0	Reserved
	0	0	0	1	1	Reserved
	0	0	1	0	0	Reserved
	0	0	1	0	1	Reserved
	0	0	1	1	0	Reserved
	0	0	1	1	1	Reserved

SET ALARM INDICATOR

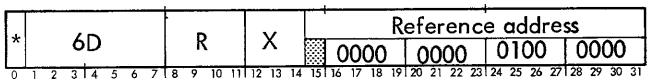
The following configuration of WD is used to set the ALARM indicator on the maintenance section of the processor control panel:



If the processor is in the RUN mode and the AUDIO switch on the maintenance section of the processor control panel is in the ON position, a 1000-Hz signal is transmitted to the basic processor speaker. The signal may be interrupted by changing from RUN mode to IDLE mode, by moving the AUDIO switch to the OFF position, or by resetting the ALARM indicator.

RESET ALARM INDICATOR

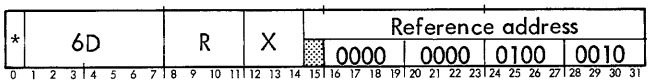
The following configuration of WD is used to reset the ALARM indicator:



The ALARM indicator is also reset by either the RESET BP or the RESET SYSTEM Command entered from the operator's control console.

TOGGLE PROGRAM-CONTROLLED-FREQUENCY FLIP-FLOP

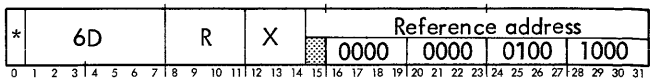
The following configuration of WD is used to set and reset the basic processor program-controlled-frequency (PCF) flip-flop:



The output of the PCF flip-flop is transmitted to the basic processor speaker through the AUDIO switch on the maintenance section of the System Control Panel. If the PCF flip-flop is reset when the above configuration of WD is executed, the WD instruction sets the PCF flip-flop; if the PCF flip-flop was previously set, the WD instruction resets it. A program can thus generate a desired frequency by setting and resetting the PCF flip-flop at the appropriate rate. Execution of the above configuration of WD also resets the ALARM indicator.

LOAD INTERRUPT INHIBITS

The following configuration of WD can be used to transfer the contents of the specified R register (R_{29-31}) to the Interrupt Inhibit field ($PSWs_{37-39}$).

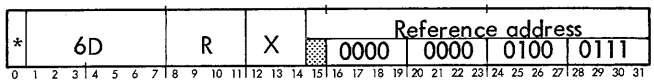


Affected: ($PSWs_{37-39}$)

(R_{29-31}) ⟶ $PSWs_{37-39}$

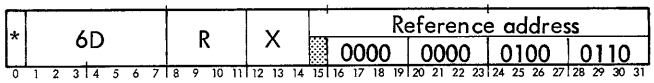
TURN ON MODE ALTERED FLAG

The following configuration of WD is used to set the Mode Altered Flag (PSWs 61) to 1:

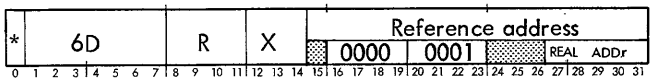


TURN OFF MODE ALTERED FLAG

The following configuration of WD is used to reset the Mode Altered Flag (PSWs 61) to 0:



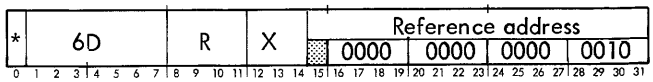
STORE IN LOW MAIN MEMORY



This instruction writes into main memory locations 0-31 (locations 0-15 shadowed by the general purpose registers and reserved locations). This allows storing or changing the Status Stack Pointer Doubleword in locations 0-1 and the default Program Status Words (Status Stack is empty) in locations 2 through 4.

If the R field is nonzero, the contents of R are stored in the main memory location identified by bits 27-31.

TRAP TO LOCATION X'47'

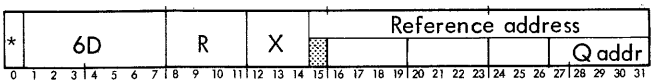


This instruction causes the basic processors to trap to location X'47'.

A line in the Processor Bus is raised by the initiating basic processor (or the associated PI). This line, when true, causes the basic processors to trap to X'47' (including the one that executes the instruction).

WRITE INTO INTERNAL CONTROL REGISTER

The following configuration of WD is used to write into the internal control (or Q) registers:



If the R field is nonzero, the contents of register R are loaded in the control register, as specified by the "Q Address" field (bit positions 27-31) of the WD instruction. Except for the four Q addresses listed below, all other addresses are reserved:

Q Address	Significance
X'1D'	(Bits 00-13) - Reserved. (Bits 14-31) - Write into the "Branch From" program counter.
X'1E'	(Bits 00 through 07) - Reserved. (Bits 08 through 31) - Write into the "Load Device Address" register.

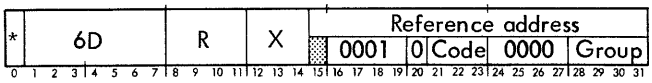
If the R field is zero, the specified register is loaded with all zeros.

Affected: (EL)

(R) ⟶ (EL)

WRITE DIRECT, INTERRUPT CONTROL (MODE 1)

The following configuration of WD is used to set and reset the various states of the individual interrupt levels within the basic processor interrupt system:



Bits 28-31 of the effective address specify the identification number (see Table 11) of the group of interrupt levels to be controlled by the WD instruction.

The R field of the WD instruction specifies a general register that contains the selection bits for the individual interrupt levels within the specified group. For external interrupt groups, bit 16 of register R contains the selection bit for the highest-priority (lowest-numbered) interrupt level within the group, and bit 31 of register R contains the selection bit for the lowest-priority (highest-numbered) interrupt level within the group. For assignments in Group X'0', see Table 11.

Except for Power on/Power off interrupt levels, which can not be disabled, disarmed, or inhibited, each level in the designated group is operated on according to the function code specified by bits 21-23 of the effective address of WD. The codes and their associated functions are as follows:

Code	Function
000	Set active all selected levels currently in the armed or waiting states.
001†	Disarm all levels selected by a 1; all levels selected by a 0 are not affected.
010†	Arm and enable all levels selected by a 1; all levels selected by a 0 are not affected.
011†	Arm and disable all levels selected by a 1; all levels selected by a 0 are not affected.
100	Enable all levels selected by a 1; all levels selected by a 0 are not affected.
101	Disable all levels selected by a 1; all levels selected by a 0 are not affected.
110	Enable all levels selected by a 1 and disable all levels selected by a 0.
111	Trigger all levels selected by a 1. All such levels that are currently armed advance to waiting state.

†These codes clear the current interrupts, i.e., remove from the active or waiting state all levels selected by a 1 (see Figure 12).

INPUT/OUTPUT INSTRUCTIONS

The I/O instruction set is comprised of eight instructions, as listed below.

Instruction Name	Mnemonic
Start Input/Output	SIO
Test Input/Output	TIO
Test Device	TDV
Halt Input/Output	HIO
Reset Input/Output	RIO
Poll Processor	POLP
Poll and Reset Processor	POLR
Acknowledge Input/Output Interrupt	AIO

OVERALL CHARACTERISTICS

All I/O instructions are privileged and can be performed only when the basic processor (BP) is in either the master or master-protected mode. If the BP attempts to execute an I/O instruction when it is in the slave mode (bit 8 of the current PSW is a 1), the instruction is aborted at the time the operation code is decoded and the BP traps to location X'40'. Programs operating in the slave mode must request I/O services from the System Monitor.

At the end of every I/O instruction, the condition code bits represent a summary description of the results of the I/O operation and conditions within the addressed I/O subsystem. Specific condition code settings and meanings (unique for each I/O instruction) are contained in the detailed description for each I/O instruction.

All I/O instructions, except RIO, may request detailed I/O status information. The type and amount of I/O status information that may be requested is determined by the operation code and the R field of the I/O instruction. The R field also designates which general register(s) is to be loaded with the requested information. (Refer to I/O Status Information for further details.)

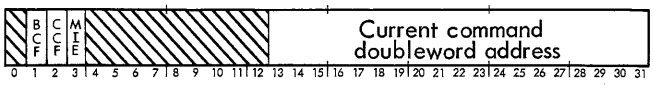
I/O instructions are similar to other word-addressing instructions in that bits 15-31 may be modified by indirect addressing and/or indexing. However, the final value of these bits is not used as an effective virtual address for memory reference. Instead, depending upon the I/O instruction, these bits are used as an extension to the operation code field, as an I/O address to select a particular I/O subsystem, or they may be reserved. Further details of I/O instructions are illustrated in Figure 13 and described in Table 13.

I/O STATUS INFORMATION

SIO, TIO, TDV, AND HIO INSTRUCTIONS

If the R field is coded with a 0, no status information is requested nor loaded. If the R field is odd, one word of status information is requested to be loaded into register R as specified by the R field. If the R field is even (not zero), two words of status information are requested to be loaded into registers R and Ru1.

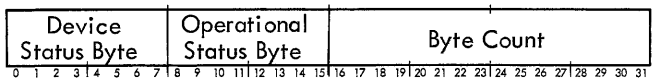
The following I/O status information may be loaded into register R only when the R field is coded with an even (nonzero) value.



The significance of each bit within register R is described in Table 14.

The following I/O status information may be loaded into register R if the R field is odd, or into register Ru1 if the R field is even and not zero.

The format of information within the specified general register (R or Ru1) is shown below.



Device Status Byte. These eight bits (0-7) when loaded into the specified general register provide status information pertaining to the addressed device and device controller or IOP. The significance of each bit when requested by an SIO, TIO, and HIO instruction is described in Table 15. The significance of these bits when requested by a TDV instruction is different and is described in the applicable peripheral device reference manual.

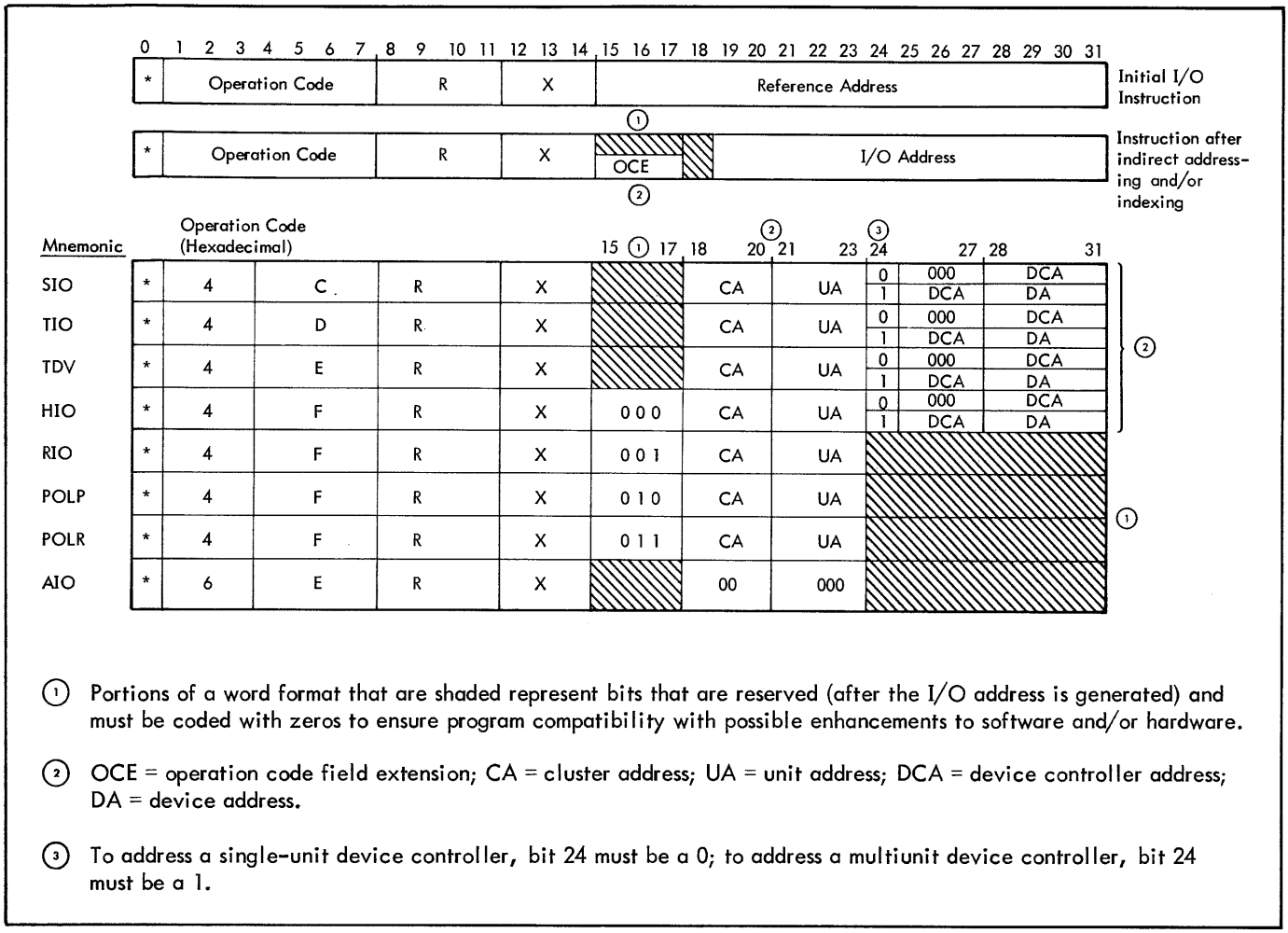
Operational Status Byte. Bits 8-15 of the specified general register (R or Ru1) indicate either the presence (1) or absence (0) of various errors which may have occurred during an I/O operation. The significance of the individual bits within the operational status byte are described in Table 16.

Table 17 is the summary description of the Device Status Byte and the Operational Status Byte.

Byte Count. Bits 16-31 of register Ru1 indicate the number of bytes that have to be transmitted to or from memory in the operation called for by the current I/O command doubleword.

RIO INSTRUCTION

No status information is returned to the general registers for an RIO instruction (the R field is ignored). Only condition code bits (CC1 - CC3) are set to reflect the I/O conditions.



① Portions of a word format that are shaded represent bits that are reserved (after the I/O address is generated) and must be coded with zeros to ensure program compatibility with possible enhancements to software and/or hardware.

② OCE = operation code field extension; CA = cluster address; UA = unit address; DCA = device controller address; DA = device address.

③ To address a single-unit device controller, bit 24 must be a 0; to address a multiunit device controller, bit 24 must be a 1.

Figure 13. Formats of I/O Instructions

Table 13. Description of I/O Instructions

Bit Position	Applicable Instructions (Mnemonics)	Function and/or Description
0	All I/O instructions	If this bit is a 1, bits 15-31 of the initial I/O instruction are modified by indirect addressing.
1-7	SIO, TIO, TDV, and AIO	For these four instructions, the operation code uniquely defines the I/O operation that is to be performed.
	HIO, RIO, POLP, and POLR	Within bit positions 1-7, these four instructions all have the same operation code (X'4F'). The instructions are differentiated by using bits 15, 16, and 17 as an extension of the operation code field.
8-11	SIO, TIO, TDV, and HIO	The value of the R field specifies how much status information is requested from the addressed I/O subsystem (IOP, device controller, and device) and into which general register(s) the status information is to be loaded. If the value of the R field is even and not 0, two words of status information are requested to be loaded into registers R and Ru1. If the value of the R field is odd, one word of status information is requested to be loaded into register R.
	RIO	Although the R field is not used by the RIO instruction, the R field may be coded with any value as required by the program.

Table 13. Description of I/O Instructions (cont.)

Bit Position	Applicable Instructions (Mnemonics)	Function and/or Description
8-11 (cont.)	POLP and POLR	This field specifies which general register (including register 0) is to receive processor (MIOP, RMP, BP, MI, PI, or System Control Processor) fault information.
	AIO	If the R field is 0, no status information is requested. If the R field is not 0, the designated general register is to be loaded with the requested status information.
12-14	All I/O instructions	The X field may be used to specify indexing.
15-17	SIO, TIO, TDV, and AIO	After the I/O address is generated, these bits are reserved and must be coded with zeros.
	HIO, RIO, POLP, and POLR	These bits are an extension to the operation code field (bits 1-7) and permit each of these instructions to be uniquely defined. Note that these bits are subject to modifications due to indirect addressing or indexing. The final configuration of these bits must be as shown below: HIO = 000 RIO = 001 POLP = 010 POLR = 011
18-31	All I/O instructions (except AIO)	The I/O address (after any indirect addressing and/or indexing) is contained within these bits. Depending upon the I/O instruction, the required I/O address may be comprised of (1) a cluster address; (2) a cluster address and a unit address; (3) a cluster address, a unit address, and a device controller address; or (4) a cluster address, a unit address, a device controller address, and a device address. Subfields of the final I/O address field are described below.
18 : 23	All I/O instructions (except AIO)	These bits constitute the cluster address (CA) and the unit address (UA) field of an I/O instruction. Cluster and unit addresses may be assigned in the following manner: 1. The assignment of addresses is mutually exclusive, that is, no two units may have the same address. 2. Bits 18-20 represent a cluster address. 3. Bits 21-23 represent a unique unit within that cluster. Since all processor clusters contain as a minimum a Processor Interface (PI) unit and a memory interface (MI) unit, the address (110) 21-23 and (111) 21-23 have been preassigned to these units.
	AIO	After the I/O address is generated, these bits are reserved and must be coded with zeros.
24	SIO, TIO, TDV, and HIO	If the I/O instruction is addressed to a single-unit device controller, this bit must be coded as a 0. If the I/O instruction is addressed to a multiunit device controller, this bit must be coded as a 1. Note that bit 24 is not considered as part of the device controller address.

Table 13. Description of I/O Instructions (cont.)

Bit Position	Applicable Instructions (Mnemonics)	Function and/or Description
24 (cont.)	RIO, POLP, POLR, and AIO	After the I/O address is generated, this bit is reserved and must be coded with a zero.
25 : 31	SIO, TIO, TDV, and HIO	If the I/O instruction is addressed to a single-unit device controller (bit 24 is a 0), bits 25-31 represent one of 16 possible device controller addresses (X'00' - X'0F'). There is no need to specify a device address. If the I/O instruction is addressed to a multiunit (e.g., magnetic tape) device controller (bit 24 is a 1), bits 25-27 represent one of eight possible device controller addresses (X'0' - X'7') and bits 28-31 represent one of 16 possible device addresses (X'0' - X'F'). Device controller addresses assigned to controllers within the same I/O channel (e.g., MIOP), must be mutually exclusive. Note that bit 24, which must be a 0 when addressing a single-unit device controller and a 1 when addressing a multiunit device controller, is not considered a part of the decive controller address. Thus, for example, if the device controller address X'0' is assigned to a multiunit device controller within an MIOP, no other device controller (single or multiunit) within that MIOP may have an address of X'0'.
	RIO, POLP, POLR, and AIO	After the I/O address is generated, these bits are reserved and must be coded with zeros.

Table 14. I/O Status Information (Register R)

Bit Position	Significance
0	Reserved[t]
1[tt]	Bus Check Fault (BCF). This bit is set to 1 if a discrepancy exists between the parity error status in the memory unit and the IOP when an IOP is performing a main memory read cycle. If the error occurs while accessing data then the device halt is controlled by the Halt-on-Transmission-Error flag (bit position 36 of an I/O command doubleword). If the error occurs while fetching a command, the operation is terminated immediately with an "unusual end".
2[tt]	Control Check Fault (CCF). This bit is set to 1 when a parity error occurs during a subchannel read operation within the MIOP. The operation terminates immediately with an "unusual end".

Table 14. I/O Status Information (Register R) (cont.)

Bit Position	Significance
3[tt]	Memory Interface Error (MIE). IOP Halt condition is the same as a Bus Check Fault.
4-12	Reserved[t]
13-31	Current Command Doubleword Address. The 19 high-order bits of the main memory address from which the command doubleword for the I/O operation currently being processed by the addressed I/O subsystem is fetched.

[t] To ensure program compatibility with possible software and/or hardware enhancements, it is recommended that reserved bits be treated as indeterminate and not used (i.e., masked).

[tt] The IOP unconditionally sets the Processor Fault Indicator (PFI) whenever a Bus Check Fault, Control Check Fault, Control Memory Fault, or Memory Interface Error occurs. The IOP fault status register is set with status information as listed under the POLP or POLR instructions.

Table 15. Device Status Byte (Register R or Ru1) (SIO, TIO, and HIO only)

Bit Position	Significance
0	Interrupt Pending. This bit is set to a 1 if the addressed device has requested an interrupt that has not been acknowledged by the BP with an AIO instruction. If this bit is a 1, the current SIO instruction is not accepted. Condition code bits are set to reflect this action and any requested status information is loaded into the designated general register(s). SIO instructions will not be accepted until the interrupt pending condition is cleared. Normally, before a device can request an interrupt, the following conditions must prevail: 1. Appropriate flag(s) (IZC, ICE, and/or IUE; bit positions 33, 35, and 37, respectively) within the I/O command doubleword must be set to 1. 2. The flagged event (byte count reduced to zero for the IZC flag, "channel end" condition for the ICE flag, or "unusual end" condition for the IUE flag) must occur. 3. IOP may signal device controller to raise interrupt without examining interrupt flags, if: a. A connection address error is detected. b. Any error is detected when IOP is accessing an IOCD. For case a, no interrupt status will be set in response to an AIO. For case b, an IUE signal is sent back in response to an AIO. An I/O interrupt may also be requested by certain devices via M modifier bits within the basic order for that device (see Operational Command Doublewords). A BP will respond to an interrupt request from a particular I/O subsystem if (1) the I/O interrupt level (X'5C') is armed, enabled, and not inhibited; and (2) that there is no higher priority interrupt level in the active or waiting state.
1,2	Device Condition. If bits 1 and 2 are 00 (device "ready"), all device conditions required for proper operation are satisfied. If bits 1 and 2 are 01 (device "not operational"), the addressed device has developed some condition that will not allow it to proceed; in either case, operator intervention is usually required. If bits 1 and 2 are 10 (device "unavailable"), the device has more than one channel of communication available and it is engaged in an operation controlled by a controller other than the one specified by the I/O address. If bits 1 and 2 are 11 (device "busy "), the device has accepted a previous SIO instruction and is already engaged in an I/O operation.
3	Device Mode. If this bit is 1, the device is in the "automatic" mode; if this bit is 0, the device is in the "manual" mode and requires operator intervention. This bit can be used in conjunction with bits 1 and 2 to determine the type of action required. For example, assume that a card reader is able to operate, but no cards are in the hopper. The card reader would be in state 000 (device "ready", but manual intervention required), where the state is indicated by bits 1, 2, and 3 of the I/O status response. If the operator subsequently loads the card hopper and presses the card reader START switch, the reader would advance to state 001 (device "ready" and in automatic operation). If the card reader is in state 000 when an SIO instruction is executed, the SIO would be accepted by the reader and the reader would advance to state 110 (device "busy", but operator intervention required). Should the operator then place cards in the hopper and press the START switch, the card reader state would advance to 111 (device "busy" and in "automatic" mode), and the input operation would proceed. Should the card reader subsequently become empty (or the operator press the STOP switch) and command chaining is being used to read a number of cards, the card reader would return to state 110. If the card reader is in state 001 when an SIO instruction is executed, the reader advances to state 111, and the input operation continues as normal. Should the hopper subsequently become empty (or should the operator press the card reader STOP switch) and command chaining is being used to read a number of cards, the reader would go to state 110 until the operator corrected the situation. For RMP, this bit is always set to one.

Table 15. Device Status Byte (Register R or Ru1) (SIO, TIO, and HIO only) (cont.)

Bit Position	Significance
4	Unusual End. If this bit is a 1, the previous I/O operation terminated in an "unusual end". Unusual end conditions occur for various reasons that are unique to each device (refer to applicable peripheral reference manual for further details).
5,6	Device Controller or IOP Condition. The function of these two bits is dependent upon the type of IOP (MIOP or RMP) addressed by the I/O instruction. MIOP Operations: If bits 5 and 6 are 00 (device controller "ready"), all device controller conditions required for its proper operation are satisfied. If bits 5 and 6 are 01 (device controller "not operational"), some condition has developed that does not allow it to operate properly. Operator intervention is usually required. If bits 5 and 6 are 10 (device controller "unavailable"), the device controller is currently engaged in an operation controlled by an IOP other than the one addressed by the I/O instruction. If bits 5 and 6 are 11 (device controller "busy"), the device controller has accepted a previous SIO instruction and is currently engaged in performing an operation for the addressed IOP. RMP Operations: If bits 5 and 6 are 00 (IOP "ready"), all RMP conditions required for its proper operation are satisfied. If bits 5 and 6 are 11 (IOP "busy"), the IOP has accepted a previous SIO instruction and is currently engaged in performing that I/O operation. If bits 5 and 6 are 01, the IOP is not operational. If bits 5 and 6 are 10, the IOP is in an undefined state.
7	Reserved. To ensure program compatibility with possible software and/or hardware enhancements, it is recommended that this bit be treated as indeterminate and not used (i.e., masked).

Table 16. Operational Status Byte (Register Ru1)

Bit Position	Significance
8	Incorrect Length. This bit is set to 1 if an incorrect length condition occurred within the responding subchannel. An incorrect length condition is caused by a "channel end" (or end of record) condition occurring before the device controller has a "count done" signal from the IOP (indicating that the byte count has been reduced to zero), or is caused by the device controller receiving a count done signal before channel end (or end of record): e.g., count done before 80 columns have been read from a card. When set to a 1, the incorrect length bit, by itself, always signifies that an incorrect length condition has occurred. If the SIL flag (bit 38 of the I/O command doubleword) is coded with a 0, the detected incorrect length condition is to be interpreted as an error condition. If the SIL flag is coded with a 1, the detected incorrect length condition is to be interpreted as a nonerror condition. If an incorrect length condition is to result in a device halt, the SIL flag must be coded with a 0 and the HTE flag (bit 36 of the I/O command doubleword) must be coded with a 1.
9	Transmission Data Error. This bit is set to 1 if the device controller or IOP detected a parity error or data overrun in the transmittal information. A device halt occurs as a result of a transmission data error only if the HTE flag of the I/O command doubleword is coded with a 1.
10	Transmission Memory Error. This bit is set to 1 if a memory parity error was detected during a data input/output operation. A device halt occurs as a result of a transmission memory error only if the HTE flag of the I/O command doubleword is coded with a 1.
11	Memory Address Error. This bit is set to 1 if a nonexistent memory address is detected during a chaining operation or a data input/output operation. This bit is cleared during a successful SIO or HIO.
12	IOP Memory Error. This bit is set to 1 if the IOP detects a memory parity error while fetching a command. The bit is cleared during a successful SIO or HIO.
13	IOP Control Error. This bit is set to 1 if the IOP detects two successive Transfer in Channel commands. The bit is cleared during a successful SIO or HIO.

Table 16. Operational Status Byte (Register Ru1) (cont.)

Bit Position	Significance
14	IOP Halt. This bit is set to 1 if an error condition is detected which causes the IOP to issue a halt order to the addressed I/O device. Error conditions which may cause an IOP halt (independent of the HTE flag within the I/O command doubleword) are: 1. Bus check fault that occurs while fetching a command 2. Control check fault 3. Memory address error 4. IOP memory error 5. IOP control error

Table 16. Operational Status Byte (Register Ru1) (cont.)

Bit Position	Significance
14 (cont.)	Error conditions which may cause an IOP halt only if the HTE flag is coded with a 1 are: 1. Bus check fault that occurs while fetching data 2. Transmission memory error 3. Transmission data error 4. Incorrect length condition occurring while the SIL flag is coded with a 0. An IOP halt condition causes the current operation to terminate immediately as an "unusual end".
15	This bit is set to a 1 if a Write Lock Violation (WLV) occurs.

Table 17. Status Response Bits for I/O Instructions

Position and State in Register Ru1

Device Status Byte								Operational Status Byte								Significance for SIO, HIO, and TIO	Significance for TDV
0	1	2	3	4	5[t]	6[t]	7	8	9	10	11	12	13	14	15		
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	interrupt pending	↑
-	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	device ready	
-	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	device not operational	
-	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	device unavailable	
-	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	device busy	
-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	device manual	unique to the device and the device controller
-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	device automatic	
-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	device unusual end	
-	-	-	-	-	0	0	-	-	-	-	-	-	-	-	-	device controller ready	
-	-	-	-	-	0	1	-	-	-	-	-	-	-	-	-	device controller not operational	
-	-	-	-	-	1	0	-	-	-	-	-	-	-	-	-	device controller unavailable	
-	-	-	-	-	1	1	-	-	-	-	-	-	-	-	-	device controller busy	
-	-	-	-	-	-	-	▒	-	-	-	-	-	-	-	-	reserved	↓
-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	incorrect length	↑
-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	transmission data error	
-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	transmission memory error	
-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	memory address error	same as for SIO, HIO, and TIO
-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	IOP memory error	
-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	IOP control error	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	IOP halt	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	write lock violation	↓

[t]The significance of bits 5 and 6 when response is from an RMP is as follows:

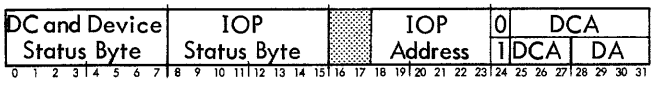
Bit 5	Bit 6	RMP Function
0	0	RMP ready
0	1	RMP not operational
1	0	reserved
1	1	RMP busy

POLP and POLR INSTRUCTIONS

The R field of these two instructions always specifies a general register (including register 0) that may receive up to 16 bits of fault status information from an addressed BP, RMP or MIOP. Each bit indicates the presence (1) or absence (0) of a specific fault condition within the polled processor (as listed in Table C-1). Note that the information represented by a particular bit is also dependent upon the type of processor polled (e.g., bit 18 may indicate a memory parity error in the BP or a control check fault within an MIOP).

AIO INSTRUCTION

For this instruction, if the R field has a value of 0, no status information is requested nor loaded. If the R field has a value of X'1' through X'F', the specified register may receive one word of I/O information pertaining to an I/O interrupt.



Device and Device Controller Status Byte. Bits 0-7 of the status word obtained by an AIO instruction from a responding I/O subsystem are unique to the device and device controller. These bits are described in the applicable peripheral device reference manual.

IOP Status Byte. Bits 8-15 indicate the presence (1) or absence (0) of various operation errors and interrupts that may have occurred during an I/O operation. The functions of individual bits within the IOP Status Byte are described in Table 18.

Table 19 is a summary description of the Device/Device Controller Status Byte and the IOP Status Byte.

Bits 16-18. These bits of the AIO response are reserved. To ensure program compatibility with any enhancements (software and/or hardware), it is recommended that these bits be treated as indeterminate and not used (i.e., masked).

Table 18. IOP Status Byte

Bit Position	Significance
8	Incorrect Length. This bit is set to 1 if an incorrect length condition occurred within the responding subchannel. An incorrect length condition is caused by a "channel end" (or end of record) condition occurring before the device controller has a "count done" signal from the IOP (indicating that

Table 18. IOP Status Byte (cont.)

Bit Position	Significance
8 (cont.)	the byte count has been reduced to zero), or is caused by the device controller receiving a count done signal before channel end (or end of record): e.g., count done before 80 columns have been read from a card. When set to a 1, the incorrect length bit, by itself, always signifies that an "incorrect length" condition has occurred. If the SIL flag (bit 38 of the I/O command doubleword) is coded with a 0, the detected incorrect length condition is to be interpreted as an error condition. If the SIL flag is coded with a 1, the detected incorrect length condition is to be interpreted as a nonerror condition. If an incorrect length condition is to result in a device halt, the SIL flag must be coded with a 0 and the HTE flag (bit 36 of the I/O command doubleword) must be coded with a 1.
9	Transmission Data Error. This bit is set to 1 if, since the last accepted SIO instruction addressed to this subchannel, the device controller or IOP detected a parity error or data overrun in the transmitted information. A device halt occurs as a result of a transmission data error only if the HTE flag of the I/O command doubleword is coded with a 1.
10	Zero Byte Count Interrupt. This bit is set to 1 if the interrupt on zero byte count flag is 1 and zero byte count is detected.
11	Channel End Interrupt. This bit is set to 1 if the interrupt at channel end flag is 1 and "channel end" is reported by the device to the IOP.
12	Unusual End Interrupt. This bit is set to 1 if the interrupt at unusual end flag is 1 and unusual end is reported by the device to the IOP, or if the IOP halt is signaled to the device controller by the IOP.
13	Write Lock Violation. This bit is set to 1 if the memory signaled a Write Lock Violation in the course of transmitting information from the device to the memory. If the HTE flag and the IUE flag are set, the operation will terminate with an "unusual end".
14	Reserved.
15	Reserved.

Table 19. Status Response Bits for AIO Instruction

Position and State in Register R																
Device Status Byte								Operational Status Byte								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Significance
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	unique to the device and the device controller
-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	incorrect length
-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	transmission data error
-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	zero byte count interrupt
-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	channel end interrupt
-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	unusual end interrupt
-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	write lock violation
-	-	-	-	-	-	-	-	-	-	-	-	-	-	▒	-	reserved
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	▒	reserved

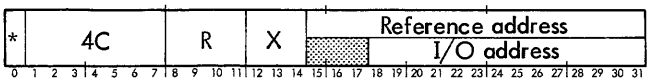
I/O Address. Depending upon the type of device controller responding to the AIO instruction, the I/O address may be comprised either of a processor address and a single-unit device controller address or a processor address, a multiunit device controller address, and a device address. The subfields of the I/O address are described in Table 20.

Table 20. I/O Address (AIO Response)

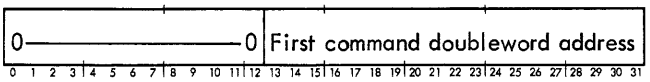
Bit Position	Significance
18-20	This field contains the cluster address.
21-23	This field contains the unit address.
24-27	This field contains all ones.
28-31	This field contains the device address.

SIO START INPUT/OUTPUT
(Word index alignment, privileged)

Instruction Register



General Register 0



START INPUT/OUTPUT performs the following:

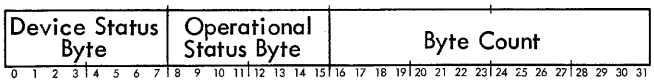
1. Attempts to initiate an input or output operation — whether an I/O operation is started or not is dependent upon conditions within the addressed I/O subsystem (see meanings of condition code settings).
2. Specifies which IOP, channel, device controller, and input/output device is to be selected (bits 18-31 of the effective virtual address of the instruction word).
3. Specifies the address of the first command doubleword for the subsequent I/O operation (bits 13-31 of general register 0).
4. Specifies how much additional status information is to be returned from the I/O system (R field, bits 8-11 of instruction word).
5. Specifies which general registers are to be loaded with the requested status information (R field, bits 8-11, of instruction word).
6. Set MIOP in test mode by using device controller address X'3F' or X'7F'. Note that device controller addresses X'3F' and X'7F' are prohibited for normal operation.

General register 0 is temporarily dedicated during SIO instruction execution and must contain the doubleword memory address of the first command doubleword specifying the operation to be started. The required address information must be in general register 0 when the SIO is executed.

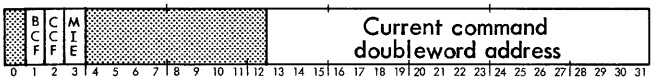
Status information for an SIO instruction is always returned via condition code bits. Additional information may be requested and returned via the general registers as specified by the R field of the SIO instruction. However, the return of the additional information is dependent upon conditions encountered within the addressed I/O subsystem (see meanings of condition code settings).

If the R field is coded with a 0, no additional status information is requested.

If the R field is coded with an odd value, one word of status information is requested to be loaded into register R. The format of this information is as follows:



If the R field is coded with an even (nonzero) value, two words of status information are requested. The format of information within register Ru1 is as shown above. The format of information within register R is as follows:



These responses provide the program with information necessary to determine the current status of the addressed I/O subsystem. The byte count field indicates the number of bytes that are to be transmitted to or from memory in the operation called for by the current command doubleword. The other fields are described in Tables 14-17.

Affected: (R), (Ru1), CC

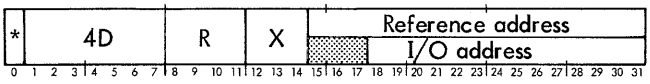
The meaning of the condition code bits during an SIO instruction is:

1	2	3	4	Meaning
0	0	0	0	I/O address recognized, SIO accepted, and status information in general registers is correct.
0	0	1	0	For RMP, I/O address recognized and SIO accepted; however, status information in general registers may be incorrect. For MIOP, not possible.
0	1	0	0	I/O address recognized, SIO not accepted because device controller or device is busy, and status information in general registers is correct.
0	1	1	0	For RMP, I/O address recognized, SIO not accepted because device controller or device is busy, and status information in general registers may be incorrect. For MIOP, not possible.
1	0	1	0	Processor Interface detected parity error on returned status and/or condition code. The result of the SIO is indeterminate.
1	1	0	0	I/O address not recognized, SIO not accepted, and status information returned to general registers is incorrect.
1	1	1	0	No I/O address recognized and SIO aborted because an error detected when the IOP attempted to read and transfer the SIO parameters (device/device controller address, R field information, and first command doubleword address) from the BP to the IOP via main memory. Status information returned to general registers is incorrect.

If CC4 = 1, the MIOP is in test mode and the meaning of the condition code during an SIO is:

1	2	3	4	Meaning
1	0	0	1	Set test mode is successful.
1	0	1	1	Set test mode is successful, but a Bus Check Fault was detected.

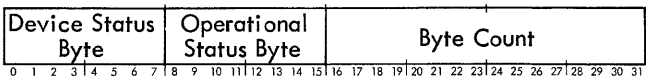
TIO TEST INPUT/OUTPUT
(Word index alignment, privileged)



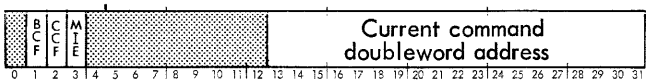
TEST INPUT/OUTPUT is used to make an inquiry on the status of data transmission. The operation of the selected IOP, device controller, and device is not affected, and no operations are initiated or terminated by this instruction. The responses to TIO provide the program with the information necessary to determine the current status of the device, device controller, and IOP, the number of bytes remaining to be transmitted into or from main memory in the operation, and the present point at which the IOP is operating in the command list.

If the R field of the TIO instruction is 0, no general registers are affected, but the condition code is set.

If the R field of TIO is an odd value, the condition code is set and the I/O status and byte count are loaded into register R as follows:



If the R field of the TIO instruction is an even value and not 0, the condition code is set, register Ru1 is loaded as shown above, and register R is loaded as follows:



Refer to Tables 14-17 for functions of individual bits within status words.

Affected: (R), (Ru1), CC

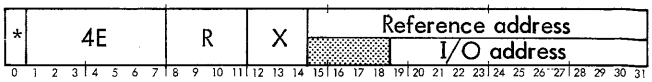
If CC4 = 0, the MIOP is in a normal mode of operation and the meaning of the condition code during a TIO is:

1	2	3	4	Meaning
0	0	0	0	I/O address recognized, acceptable SIO is currently possible, and status information in general registers is correct.
0	0	1	0	For RMP, I/O address recognized, acceptable SIO is currently possible; however, status information in the general registers may be incorrect. For MIOP, not possible.
0	1	0	0	I/O address recognized but acceptable SIO is not currently possible because device controller or device is busy. Status information in general registers is correct.
0	1	1	0	For RMP, I/O address recognized but acceptable SIO is not currently possible because device controller or device is busy; status information in general registers may be incorrect. For MIOP, not possible.
1	0	1	0	Processor Interface detected parity error on returned status and/or condition code. The result of the TIO is indeterminate.
1	1	0	0	I/O address not recognized, TIO not accepted, and status information returned to general registers is incorrect.
1	1	1	0	No I/O address recognized and TIO aborted because an error detected when the IOP attempted to read and transfer the TIO parameters (device/device controller address and R field information) from the BP to the IOP via main memory. Status information returned to general registers is incorrect.

If CC4 = 1, the MIOP is in the test mode and the meaning of the condition code during a TIO is:

1	2	3	4	Meaning
0	0	0	1	Unit is performing an Order Out operation.
0	1	0	1	Unit is performing an Order In operation.
1	0	0	1	Unit is performing a Data Out operation.
1	0	1	1	Parity error detected by Processor Interface on returned status and/or condition code. The result of the TIO is indeterminate.
1	1	0	1	Unit is performing a Data In operation.
1	1	1	1	BCF detected while unit performing a Data In operation.

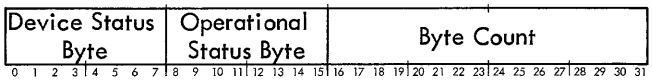
TDV TEST DEVICE
(Word index alignment, privileged)



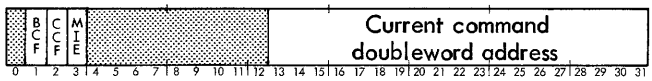
TEST DEVICE is used to provide information about a device other than that obtainable by means of the TIO instruction. The operation of the selected IOP, device controller, and device is not affected, and no operations are initiated or terminated. The responses to TDV provide the program with information giving details on the condition of the selected device, the number of bytes remaining to be transmitted in the current operation, and the present point at which the IOP is operating in the command list.

If the R field of the TDV instruction is 0, the condition code is set, but no general registers are affected.

If the R field of TDV is an odd value, the condition code is set and the device status and byte count are loaded into register R as follows:



If the value of the R field of TDV is an even value and not 0, the condition code is set, register Ru1 is loaded as shown above, and register R is loaded as follows:



Refer to the applicable peripheral reference manual for description of Device Status Byte. Refer to Tables 16 and 17 for functions of other bits within status words.

Affected: (R), (Ru1), CC

If CC4 = 0, the MIOP is in a normal mode of operation and the meaning of the condition code during a TDV is:

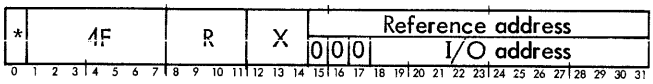
1	2	3	4	Meaning
0	0	0	0	I/O address recognized, no device-dependent condition present, and status information in general registers is correct.
0	0	1	0	For RMP, I/O address recognized and no device-dependent condition present; however, status information in general registers may be incorrect. For MIOP, not possible.
0	1	0	0	I/O address recognized and device-dependent condition is present or device controller is in test mode.
0	1	1	0	For RMP, I/O address recognized, device-dependent condition is present, or device controller is in test mode; but status information in the general registers may be incorrect. For MIOP, not possible.

1	2	3	4	Meaning
1	0	1	0	Processor Interface detected parity error on returned status and/or condition code. The result of the TDV is indeterminate.
1	1	0	0	I/O address not recognized, TDV not accepted, and status information returned to the general registers is incorrect.
1	1	1	0	No I/O address recognized and TDV aborted because an error detected when the IOP attempted to read and transfer the TDV parameters (device/device controller address and R field information) from the BP to the IOP via main memory. No status information returned to general registers.

If CC4 = 1, the MIOP is in the test mode and the meaning of the condition code during a TDV is:

1	2	3	4	Meaning
0	0	0	1	Unit is performing an Order Out operation.
0	1	0	1	Unit is performing an Order In operation.
1	0	0	1	Unit is performing a Data Out operation.
1	0	1	1	Parity error detected by Processor Interface on returned status and/or condition code. The result of the TDV is indeterminate.
1	1	0	1	Unit is performing a Data In operation.
1	1	1	1	BCF detected while unit performing a Data In operation.

HIO HALT INPUT/OUTPUT
(Word index alignment,[t] privileged)

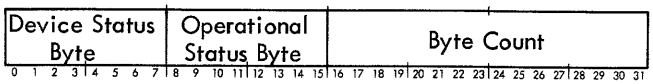


HALT INPUT/OUTPUT causes the addressed device to immediately halt its current operation (perhaps improperly, in the case of magnetic tape units, when the device is forced to stop at other than an interrecord gap). If the device is in an interrupt-pending condition, the condition is cleared.

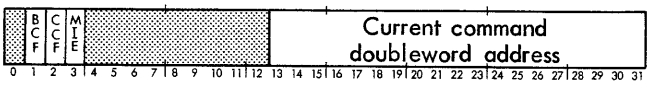
[t]When indexing operation code 4F instructions (HIO, RIO, POLP, POLR), the programmer must make certain that the summation of the contents of the index register and the I/O address (bits 18-31 of the instruction word) does not affect bits 15-17. When indirect addressing is used, the contents of the indirect address location (bits 15, 16, and 17) must specify the desired operation code extension.

If the R field of the HIO instruction is 0, the condition code is set, but no general registers are affected.

If the R field is an odd value, the condition code is set and the following information is loaded into register R.



If the R field of HIO is an even value and not 0, the condition code is set, register Ru1 is loaded as shown above, and register R contains the following information.



This information shows the status of the addressed I/O subsystem at the time of the halt. The byte count field shows the number of bytes remaining to be transmitted to or from memory. Other fields are described in Table 14-17.

The HIO instruction must have zeros in bit positions 15, 16, and 17 to differentiate it from the RIO, POLP, and POLR instructions, which also have X'4F' as an operation code (bits 1-7).

Affected: (R), (Ru1), CC

If CC4 = 0, the MIOP is in a normal mode of operation and the meaning of the condition code during an HIO instruction is:

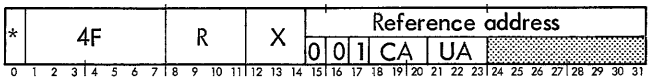
1	2	3	4	Meaning
0	0	0	0	I/O address recognized, HIO accepted, device controller not busy at time of HIO, and status information in general registers is correct.
0	0	1	0	For RMP, I/O address recognized, HIO accepted, and device controller not busy at time of HIO; but status information in general registers may be correct. For MIOP, not possible.
0	1	0	0	I/O address recognized, HIO accepted, and device controller busy at the time of the HIO, and status information is correct.
0	1	1	0	For RMP, I/O address recognized, HIO accepted, and device controller busy at the time of the HIO; but the status information in the general registers may be incorrect. For MIOP, not possible.
1	0	0	0	Not possible.
1	0	1	0	Processor Interface detected parity error on returned status and/or condition code. The result of the HIO is indeterminate.

1	2	3	4	Meaning
1	1	0	0	I/O address not recognized, HIO not accepted, and no status information returned to general registers.
1	1	1	0	No I/O address recognized and HIO aborted because an error detected when the IOP attempted to read and transfer the HIO parameters (device/device controller address and R field information) from the BP to the IOP. No status information returned to general registers.

If CC4 = 1, the MIOP is in the test mode and the meaning of the condition code during an HIO is:

1	2	3	4	Meaning
0	0	0	1	Unit is performing an Order Out operation.
0	1	0	1	Unit is performing an Order In operation.
1	0	0	1	Unit is performing a Data Out operation.
1	0	1	1	Processor Interface detected parity error on returned status and/or condition code. The result of the HIO is indeterminate.
1	1	0	1	Unit is performing a Data In operation.
1	1	1	1	BCF detected while unit performing a Data In operation.

RIO RESET INPUT/OUTPUT
(Word index alignment, [t] privileged)



RESET INPUT/OUTPUT causes the selected IOP to generate an I/O reset signal to all devices attached to it. In addition to the operation code X'4F', bits 15, 16, and 17 must be coded as 001, respectively.

An RIO instruction resets the selected unit in the same manner as Z[c]RIO on the operator's control console. However, unlike the control command, the RIO instruction resets only the addressed unit and may be controlled by the executing program. Since the BP may be addressed as an IOP, it will accept an RIO instruction that causes the BP to reset itself in the same manner as Z[c]RBP. (Note that this procedure is not normal practice.)

Cluster addresses (CA), bit positions 18-20, may have values of X'0'-X'7'. Cluster addresses X'0'-X'6' may be assigned to any cluster containing processors (i.e., BP, MIOP, and/or RMP). In a monoprocessor system, cluster address X'0' is assigned to the cluster containing the basic processor (BP). Cluster address X'7' is assigned only to the cluster containing a system processor. If CA equals X'7', the UA field is reserved. Unit addresses (UA), bit positions 21-23, may have values of X'0'-X'7'. Unit addresses are required only if the cluster address is X'0'-X'6', (i.e., cluster contains either a BP, MIOP, and/or RMP). Unit addresses X'0'-X'5' may be assigned to processors within the cluster. Unit address X'5' in cluster X'0' is reserved for the BP. Unit address X'6' is assigned always to the MI and unit address X'7' is assigned always to the PI for all clusters.

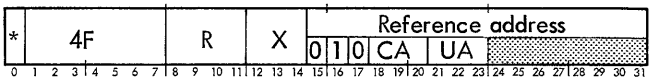
Status information is returned only in the condition code bits. The R field is not used.

Affected: CC1, CC2, CC3

Condition code settings are as shown below:

1	2	3	4	Meaning
0	0	0	-	I/O address recognized.
1	1	0	-	I/O address not recognized.

POLP POLL PROCESSOR
(Word index alignment, [t] privileged)



POLL PROCESSOR causes the addressed unit to return unit fault status in bits 16-31 of register R[tt]. This status information is unit dependent (see Appendix C, Table C-1).

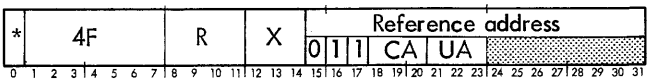
In addition to the operation code of X'4F', bits 15, 16, and 17 must be coded as 010, respectively.

Affected: (R), CC1, CC2, CC3

Condition Code settings are as shown below:

1	2	3	4	Result of POLP
0	0	0	-	Processor fault interrupt not pending.
0	1	0	-	Processor fault interrupt pending.
1	1	0	-	Unit address not recognized.

POLR POLL AND RESET PROCESSOR
(Word index alignment, [t] privileged)



POLL AND RESET PROCESSOR causes the selected unit to return unit fault status in bits 16 to 31 of register R[tt] and resets the unit's fault status register. This status information is unit dependent (see Appendix C, Table C-1).

---

[t] See footnote to HIO instruction.

[tt] This fault status is duplicated in bits 0 to 15 of register R.

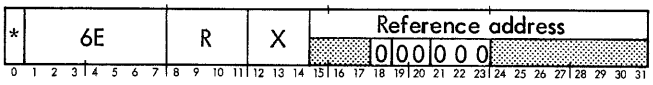
The POLR instruction also resets and clears this unit's Processor Fault Interrupt signal and the error status register. In addition to the operation code of X'4F', bits 15, 16, and 17 must be coded as 011, respectively.

Affected: (R), CC1, CC2, CC3

Condition code settings for the POLR instruction are:

1	2	3	4	Result of POLR
0	0	0	-	Processor fault interrupt not pending.
0	1	0	-	Processor fault interrupt pending.
1	1	0	-	Unit address not recognized.

AIO ACKNOWLEDGE INPUT/OUTPUT INTERRUPT
(Word index alignment, privileged)



ACKNOWLEDGE INPUT/OUTPUT INTERRUPT is used to acknowledge an input/output interrupt and to identify the I/O subsystem (processor, device controller, device) that is causing the interrupt and why. If more than one I/O subsystem has an interrupt pending, only the subsystem with the highest priority will respond to the AIO. Bits 18-23 of the effective virtual address of the AIO instruction (normally used to specify the cluster and unit addresses of the I/O address field) must be coded 000000 to specify the standard I/O system interrupt acknowledgment (other codings of these bits are reserved for use with special I/O systems). The remainder of the I/O selection code field (bit positions 24-31) are not used in the standard I/O interrupt acknowledgment (the address of the interrupt source is a part of the response from the standard I/O system to the AIO instruction).

Standard I/O interrupts are program controlled via the control flags (IZC, ICE, IUE, HTE, and SIL) within the I/O command doublewords (IOCDs) that comprise the command list for the I/O operation. If a particular flag is coded as a 1 and if the corresponding condition occurs within the I/O operation, then an I/O interrupt is requested (e.g., if the IZC flag is set to 1 and if the byte count for the I/O operation has been decremented to zero, then an I/O interrupt is requested by that I/O subsystem to indicate the end of that I/O operation; if the IZC flag is coded as a 0, no I/O interrupt is requested as a result of the byte count being decremented to zero).

If two or more flags are coded to cause an interrupt for two or more conditions, an interrupt is requested whenever any of the "flagged" conditions is detected.

For some conditions (transmission errors, incorrect length), two or more flags must be properly coded (see Chapter 4 for further details on IOCDs).

Some error conditions (e.g., parity error on reading command doubleword) will unconditionally cause an I/O interrupt.

The various conditions which may result in an I/O interrupt, the coding of the corresponding control flags within the IOCD, and the bit position within the status word (returned to register R) that indicates the presence (1) or absence (0) of that interrupt condition are listed below:

Condition	Control Flags Coding	Status Bit Set
Zero byte count	IZC = 1	10
Channel end	ICE = 1	11
Transmission memory error	IUE = 1, HTE = 1	12
Write lock violation	IUE = 1, HTE = 1	12
Incorrect length	IUE = 1, HTE = 1 and SIL = 0	8, 12
Memory address error, IOP memory error, IOP control error, or device connection address parity error	(no flag needed)	12
Transmission data error	IUE = 1, HTE = 1	9, 12
Unusual end	IUE = 1	12
IOP halt	IUE = 1	12, 14

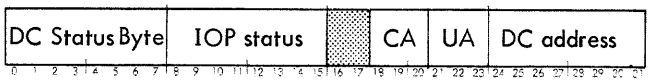
Interrupts may also be requested by certain I/O devices when they execute specific orders (e.g., when a magnetic tape unit executes a Rewind and Interrupt order). Refer to the applicable peripheral reference manual for further details.

When a device interrupt condition occurs, the IOP forwards the request to the interrupt system I/O interrupt level. If this interrupt level is armed, enabled, and not inhibited, the BP eventually acknowledges the interrupt request and executes the XPSD instruction in main memory location X'5C', which normally leads to the execution of an AIO instruction.

For the purpose of acknowledging standard I/O interrupts, the IOPs, device controllers, and devices are connected in a preestablished priority sequence that is customer-assigned and is independent of the physical locations of the portions of the I/O system in a particular installation.

If the R field of the AIO instruction is 0, the condition code is set but the general register is not affected.

If the R field of AIO is not 0, the condition code is set and register R is loaded with the following information.



The functions of bits within the DC status byte (which are unique to the device and device controller) are described in applicable peripheral reference manuals. The functions of other bits in the AIO response word are described in Tables 18, 19, and 20.

The AIO instruction resets the interrupt request signal for the I/O subsystem responding to the AIO (i.e., I/O subsystem identified by bits 19-31 of register R).

Affected: (R), CC

If CC4 = 0, the MIOP is operating in a normal mode of operation and the condition code settings for AIO are shown below:

1	2	3	4	Result of AIO
0	0	0	0	Normal interrupt recognized and reset. Status information in general register is correct.
0	0	1	0	For RMP, normal interrupt recognized and reset; status information in the general register may be incorrect. For MIOP, not possible. Parity error on returned status and/or condition code. The result of the AIO is indeterminate.
1	0	1	0	Processor interface detected.
0	1	0	0	Unusual condition interrupt recognized and reset. Status information in general register is correct.
0	1	1	0	For RMP, unusual condition interrupt recognized and reset; status information in the general register may be incorrect. For MIOP, not possible.
1	0	0	0	Interrupt recognized and reset. Status information not returned.
1	1	0	0	No I/O device requesting an interrupt and no status information returned to the general register.
1	1	1	0	Not possible.

If CC4 = 1, the MIOP is in the test mode and the meaning of the condition code during an AIO is:

1	2	3	4	Meaning
0	0	0	1	Unit is performing an Order Out operation.
0	1	0	1	Unit is performing an Order In operation.
1	0	0	1	Unit is performing a Data Out operation.
1	0	1	1	Parity error detected by Processor Interface.
1	1	0	1	Unit is performing a Data In operation.
1	1	1	1	BCF detected while unit is performing a Data In operation.

4. INPUT/OUTPUT OPERATIONS

To accommodate the variety and number of I/O devices which may be required for scientific and commercial applications, a Xerox 560 computer system may include the following: External Direct Input/Output (DIO) interface, Multiplexor Input/Output Processors (MIOPs), and Rotating Memory Processors (RMPs).

EXTERNAL DIO INTERFACE

An external DIO interface permits standard and specially designed I/O devices to perform I/O operations (normally in a real-time environment) that are controlled directly by the basic processor (BP). Appropriate control signals and up to one word (32 bits) of data may be exchanged between the BP and an addressed I/O device for each READ DIRECT or WRITE DIRECT instruction executed by the BP.

During a WRITE DIRECT instruction (Mode 2 through F), the BP holds the control and data lines stable until an acknowledgment signal is received from the addressed I/O device. During a READ DIRECT instruction (Mode 2 through F), the BP holds the control lines stable until the addressed I/O device furnishes the data accompanied with an acknowledgment signal. Any delay encountered in receiving the acknowledgment signal, for either READ DIRECT or WRITE DIRECT instructions, does not have an adverse effect upon I/O operations being performed by the MIOP or RMP systems.

Refer to Xerox publication 90 09 73 (Interface Design Manual) for further details pertaining to the external DIO interface. Also, refer to appropriate peripheral reference manuals for details on control and data signals.

MULTIPLEXOR INPUT/OUTPUT PROCESSOR (MIOP)

An MIOP permits standard and commercially available I/O devices (e.g., card readers, card punches, magnetic tape units, etc.) to be controlled primarily by individual I/O subchannels within the MIOP and associated device controllers. Depending upon the number of I/O subchannels assigned (maximum of 16, as described under "Device Controllers"), an equivalent number of I/O operations may be performed simultaneously.

DEVICE CONTROLLERS

All I/O devices associated with an MIOP are connected via an appropriate device controller. Depending upon the number and type of I/O devices to be connected, one or more of the following types of device controllers may be connected to an MIOP:

1. Single-unit device controller (internal or external).

2. Multi-unit device controller (internal or external).

3. Unit-record controller (internal or external).

Generally, an internal device controller is physically connected via the internal I/O interface.

An external device controller is located remotely to the MIOP and may require one or more separate chassis to accommodate it.

A single-unit device controller (internal or external) is specifically designed to control only one I/O device, usually a unit-record device such as a card reader, a card punch, or a line printer. Characteristics of a single-unit device controller are dependent upon the device controlled. (Refer to an appropriate peripheral reference manual for further information.)

A multi-unit device controller (internal or external) is specially designed to control more than one I/O device, where all the I/O devices are of the same type (e.g., magnetic tape units or RADs). However, only one I/O device at a time may be actively involved in a data transfer operation. Characteristics of a multi-unit device controller are dependent upon the I/O devices controlled. For example, a multi-unit device controller for magnetic tape units may control up to eight units. (Refer to an appropriate peripheral reference manual for further information.)

Unit-record controllers (internal or external) are designed to control up to eight unit record type of I/O devices (e.g., card readers, card punches, line printers). All I/O devices attached to a unit-record controller need not be of the same type. All I/O devices attached to a unit-record controller may perform separate I/O operations, including data transfers, simultaneously.

The number of device controllers, as well as the number of I/O devices, that may be connected to an MIOP is dependent upon the following considerations:

1. The maximum number of I/O subchannels within an MIOP is 16.

2. Each single-unit device controller (internal or external) requires one I/O subchannel.

3. Each multi-unit device controller (internal or external) requires one of the first eight subchannels within the MIOP.

4. Each unit-record controller (internal or external) requires one I/O subchannel per each unit record device attached, up to a maximum of eight.

5. The maximum number of internal device controllers within an MIOP is eight (where a unit-record device controller is equivalent to one, regardless of the number of assigned subchannels).

6. Any I/O subchannel not assigned to an internal device controller may be assigned to an external device controller. Thus, if an MIOP has no internal device controller, all 16 I/O subchannels may be assigned to external device controllers.

ROTATING MEMORY PROCESSOR (RMP)

Each RMP is a special purpose, single-channel IOP designed to enhance high-speed data transfers between main memory and any one of up to eight disk units. Functionally, an RMP is comparable to an MIOP, except: (1) at any given time, only one disk unit may be selected for a data transfer operation, (2) data transfer rate of disk units are generally higher than data transfer rates of I/O devices attached to an MIOP, and (3) the device controller function is performed by the RMP, hence disk units are connected directly to the RMP rather than via a device controller. (Note: Although only one disk unit may be actively transferring data at any given time, the other units may be active in performing control functions, e.g., seeking).

INPUT/OUTPUT PROCESSOR (IOP) FUNDAMENTALS

This section contains general information, programming concepts, and definition of terms pertaining to I/O operations performed by Input/Output Processors (i.e., MIOP and RMP systems). The large variety of I/O devices which may be used with these IOPs precludes a detailed or exhaustive description of features which are unique to each device. Likewise, a general reference "Refer to an appropriate Xerox peripheral reference manual" is made rather than citing specific manuals.

Within this manual, the following terminology is used to differentiate the hierarchy of control during an I/O operation: The BP executes instructions, the IOPs execute commands, and the device controller/device execute orders.

COMMAND LIST

Each I/O operation performed by an IOP must be defined by a command list. The characteristics and requirements of a command list are as follows:

1. It is normally created by a BP-executed program prior to the time that the defined I/O operation is initiated. It must reside in main memory when the I/O operation is initiated and subsequently executed.

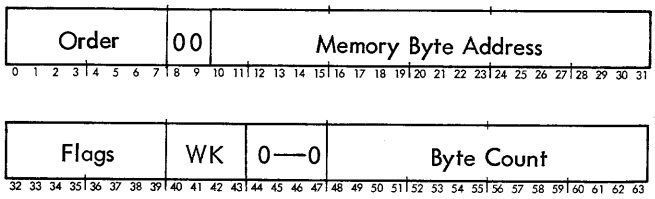
2. Depending upon various programming considerations, the command list may be contained within one or more areas of memory and each area may be comprised of one or more I/O command doublewords (IOCDs).

3. Command list continuity between IOCDs relating to the same logical record or to the same logical file may be specified (see "Data Chain Flag" and "Command Chain Flag" under "Operational IOCDs"). Command list continuity between portions of a command list located in different areas of main memory may be accomplished by including a control IOCD within the command list (see "Transfer in Channel" under "Control IOCDs").

4. Each IOCD is comprised of two words in contiguous memory word locations. The first word must be stored in an even memory word location and the second word must be stored in the next consecutive (odd) memory word location. Each IOCD is either an operational IOCD or a control IOCD and contains coded parameters to define either a complete I/O operation or an integral portion of an I/O operation. (See "Operational IOCD" and "Control IOCD" for further details.)

OPERATIONAL IOCD

An operational IOCD may contain up to five fields of parameters, as required, to define either an entire I/O operation or an integral portion of an I/O operation. The general format and description of parameters contained within an operational IOCD are as follows:



ORDER

This 8-bit field (bit positions 0-7), if required, may be coded to specify either an input or an output order that is executed by the device controller/device. General coding formats and functions of typical I/O orders are listed below:

Bit Position 0 1 2 3 4 5 6 7	Order	Function
M M M M M M 0 1	Write	Output operation
M M M M M M 1 0	Read	Input operation
M M M M M M 1 1	Control	Output control information
M M M M 0 1 0 0	Sense	Input control information
M M M M 1 1 0 0	Read Backward	Input data, in reverse sequence

Orders that are executed by a specific type of device are listed and described in the appropriate Xerox peripheral equipment reference manual.

When an operational IOCD is fetched by the IOP, the content of the order field, if required, is loaded into an order register within the device controller/device. If two or more IOCDs are required to define a logical record (as described under "Data Chain Flag"), the order obtained from the first IOCD prevails for all subsequent IOCDs within that logical record and any orders contained within the subsequent IOCDs are ignored.

MEMORY BYTE ADDRESS

This 22-bit field (bit positions 10-31), if required, is coded with the initial memory byte address for the I/O operation that will be performed when the current IOCD is executed. When the IOCD is fetched by the IOP, the content of the memory byte address field is loaded into a memory byte address register within the appropriate I/O subchannel of the IOP. Thereafter, the content of the memory byte address register is incremented (or decremented during Read Backward operations) by one for each byte of data or information transmitted, even though access to main memory may be inhibited (as described under "Skip Flag") or the data is rejected by a memory unit (as described under "Write Key").

Depending upon the characteristics of the I/O device, the content of bit positions 10-31 may either be ignored (e.g., "Rewind" order for magnetic tape units) or specify memory byte locations that contain supplemental control information (e.g., starting address for a disk seek operation). Refer to an appropriate Xerox peripheral equipment reference manual for further details.

FLAGS

Each operational IOCD contains eight control flags (bit positions 32-39). As described below, each control flag is coded to specify a particular control function that may be performed by the IOP either during or at the end of the current IOCD.

Data Chain Flag (Bit Position 32). Coding of the data chain flag is dependent upon the number of IOCDs required to define the data transfers for a logical record. If two or more IOCDs are required (e.g., to perform a "gather-write" or a "scatter-read" operation), the data chain flag of each operational IOCD, except the last IOCD, must be coded as a 1. The data chain flag of the last IOCD or the only IOCD (if the record is defined by a single IOCD) is coded as 0. If data chaining is specified and no error conditions are encountered, the IOP will automatically fetch the next operational IOCD when the byte count (described later) of the current IOCD is reduced to zero. (Note: The IOP may also fetch and execute a control IOCD containing a Transfer in Channel command, as described later, before fetching the next operational IOCD.) As a result of fetching the next operational IOCD, all parameters, except the I/O order, are updated and the device controller/device continue to operate as if the I/O operation were defined by a single IOCD (i.e., the data chain operation is transparent to the device controller/device). If data chaining is not specified, the IOP will generate a "count done" signal when the byte count of the current IOCD is reduced to zero. The "count done" signal indicates that the IOP has completed all data transfers for the current logical record. However, as described under "Interrupt on Channel End Flag", the I/O order is not completed until the device signals a "channel end".

Interrupt at Zero Byte Count Flag (Bit Position 33). If an I/O interrupt is to be requested when the byte count of the current IOCD is reduced to zero, the Interrupt at Zero Byte Count (IZC) flag must be coded as a 1. If the I/O interrupt level within the interrupt system (location X'5C') is armed, enabled, and not inhibited, the request will be processed by the BP in accordance with the priority that prevails within the interrupt system, the IOPs, and the I/O subchannels within an MIOP. The occurrence of an I/O interrupt because of a zero byte count condition is reported as status information (bit position 10 of register R) when the BP executes an AIO instruction (normally part of the I/O interrupt handling routine). The I/O interrupt request may be processed without interfering with the I/O operation. (Note: An I/O interrupt may be requested at "channel end" or on "unusual end" condition, as described later.)

Command Chain Flag (Bit Position 34). Command chaining permits an I/O device to execute a multiple number of orders relating to the same I/O operation in a consecutive manner (e.g., when reading a multi-record file, the I/O device may automatically receive a new Read order upon completing the current Read order without the BP executing another SIO instruction). Command chaining, if required, is specified by coding the command chain flag as a 1 in the IOCD of each record, except the last.

If command chaining is specified, the IOP will fetch the next operational IOCD when the device signals a "channel end" unless terminated by an "unusual end" condition. As a result, new parameters are stored in the appropriate registers within the I/O subchannel and a new I/O order is received by the device controller/device.

Thus, an IOP will automatically access main memory and fetch the next operational IOCD if either data chaining or command chaining is specified. If data chaining and command chaining are both specified in the same command doubleword, a data chaining operation will be performed if the byte count is reduced to zero before the device signals a "channel end" and a command chaining operation will be performed if a "channel end" occurs before the byte count is reduced to zero. If neither data chaining or command chaining is specified, the I/O operation is completed when the device signals a "channel end". Note that command chaining is inhibited by "unusual end".

Interrupt at Channel End (Bit Position 35). An I/O interrupt may be requested when the device signals a "channel end" (signifying that the current order has been either completed or terminated) by coding the Interrupt at Channel End (ICE) flag as a 1. If the I/O interrupt level within the interrupt system (location X'5C') is armed, enabled, and not inhibited, the request will be processed by the BP in accordance with the priority that prevails within the interrupt system, the IOPs, and the I/O subchannels of the MIOP. The occurrence of an I/O interrupt because of a "channel end" is reported as status information (bit position 11 of register R) when the BP executes an AIO instruction (normally part of the I/O interrupt-handling routine). The I/O interrupt request may be processed without affecting the I/O operation. (Note: Specific conditions under which a "channel end" signal may be generated are dependent upon the characteristics of the device. Refer to an appropriate Xerox peripheral reference manual for further details.)

Halt on Transmission Error Flag (Bit Position 36). The following errors (or "unusual end" condition) may be detected by the MIOP when an IOCD is being executed:

1. Bus check fault (BCF) while fetching data.
2. Transmission Data Error (TDE); may also be detected by device controller.
3. Transmission Memory Error (TME).
4. Write Lock Violation (WLV), during input operations only.
5. Incorrect length, conditional; see "Suppress Incorrect Length Flag".
6. Memory Interface Error (MIERR) while fetching data.

If the HTE flag is coded as a 0, the above errors are recorded when detected and reported as status information when the BP executed an SIO, TIO, or HIO instruction, but the I/O operation is not halted.

If the HTE flag is coded as a 1, and any error (as listed above) is detected, the I/O operation is terminated immediately. The error is also reported as status information when the BP executes an SIO, HIO, or TIO instruction.

The HTE flag must be coded identically in every IOCD associated with the same logical record. Thus, if data chaining is specified, the HTE flag in the new IOCD must be the same as the HTE flag in the previous IOCD. This restriction applies to data chaining only, and not to command chaining.

In addition to the "unusual end" conditions listed above, which may terminate the I/O operation only if the HTE flag is coded as a 1, any of the following "unusual end" conditions will unconditionally terminate the I/O operation:

1. Memory Address Error (MAE).
2. IOP Control Error (IOPCE).
3. Control Check Error (CCF).
4. IOP Memory Error (IOPME).
5. Bus Check Fault (BCF) while fetching an IOCD.
6. Memory interface Error (MIE) while fetching an IOCD.

Interrupt on Unusual End Flag (Bit Position 37). If an I/O Interrupt is to be requested when an "unusual end" condition is detected while either fetching or executing an IOCD, the Interrupt on Unusual End (IUE) flag must be coded as a 1. If the I/O interrupt level within the interrupt system (location X'5C') is armed, enabled, and not inhibited, the request will be processed by the BP in accordance with the priority that prevalis within the interrupt system, the IOPs, and the I/O subchannels within an MIOP. The occurrence of an I/O interrupt because of an "unusual end" condition is reported as status information (bit position 12 of register R) when the BP executes an AIO instruction (normally part of an I/O interrupt-handling routine). The I/O interrupt request may be processed without affecting the progress of the I/O operation.

If the IUE flag is coded as a 0, an "unusual end" condition may be detected but no interrupt will be requested.

Suppress Incorrect Length Flag (Bit Position 38). An incorrect length condition may occur when the specified byte count is not equal to a fixed or prescribed byte count for a record (e.g., attempting to read more than 80 columns of data from a punched card). Specific conditions under which an incorrect length signal is generated are dependent upon the device. Refer to an appropriate Xerox peripheral equipment reference manual for further details.

If the Suppress Incorrect Length (SIL) flag is coded as a 0 when an incorrect length condition is detected, it is reported as an incorrect length and, depending upon the device, may be reported as an "unusual end". If the HTE flag is also coded as a 1, the I/O operation is terminated and reported as an "unusual end".

If the SIL flag is coded as a 1 when an incorrect length condition is detected, it is reported as an incorrect length but suppressed as an "unusual end". Hence, the I/O operation is not terminated.

The presence or absence of an incorrect length condition is reported as status information when the BP executes an SIO, HIO, AIO, or TIO instruction.

Skip Flag (Bit Position 39). If the Skip (S) flag is coded as a 0, it has no effect upon the I/O operation.

If the S flag is coded as a 1, the IOP is inhibited from accessing main memory and consequently no data is transferred between the main memory and the data buffers of the I/O subchannel. All other operations or functions within the

I/O subchannel (i.e., data transfers between the device and data buffers, updating the memory byte address and byte count, and functions as specified by the control flags) are performed in a normal manner.

For input operations, the Skip flag (in conjunction with data chaining) provides the capability to selectively read portions of a record.

For output operations, the IOP will generate and transmit zeros (X'00') until the byte count is reduced to zero. Thus, for example, if the IOCD contains a Punch Binary order, a byte count of 120, and the S flag is coded as a 1, a blank card may be punched without accessing main memory for data.

WRITE KEY

This four-bit field (bit positions 40-43), if required, may be coded with an appropriate write key. During input operations and providing the Skip control flag is coded as a 0, the IOP will access main memory and furnish a memory unit with up to four bytes of data or information accompanied with a four-bit write key. If the write key matches the preassigned write lock for the memory word location accessed, or if either the key or lock has a value of 0000, the memory unit accepts and stores the information. If the write key does not match the write lock, and neither the key nor the lock has a value of 0000, the memory unit rejects the information, does not disturb the previous content, and transmits a Write Lock Violation (WLV) signal to the IOP. The write key/write lock relationship is compared every time a memory word location is accessed for storing data or information. (Note: The write key/write lock relationship may change during an input operation when the byte address is incremented (or decremented) across a memory page boundary.)

As long as the write key matches the write lock for each memory word location accessed, or the value of either the lock or the key is 0000, the input operation is performed as specified by the other parameters within this IOCD; or the input operation is terminated by an "unusual end" condition which can not be inhibited (i.e., memory address error, control check fault, or IOP memory error).

If the HTE control flag is coded as a 1 when a WLV signal is received, the I/O operation is terminated immediately. If either the ICE or IUE control flag is coded as a 1, an I/O interrupt is requested.

If the HTE control flag is coded as a 0 when a WLV signal is received, the I/O operation continues in a normal manner, even though the data or information may be rejected by a memory unit.

When the IOP receives a WLV signal, the WLV bit within the status information register is set to 1 and remains set until a new I/O operation is initiated within this I/O subchannel by an SIO instruction. Thus, after the first WLV signal has been recorded, subsequent WLV signals have no further effect upon the WLV bit. The status of the WLV bit is reported when the BP executes an SIO, TIO, TDV, HIO, or AIO instruction.

The contents of the write key field is not required and may be ignored when the write key/write lock memory protection feature is not operative (i.e., during any output operation or during any input operation, if the Skip control flag of the current IOCD is coded as a 1).

BYTE COUNT

This 16-bit field (bit positions 48-63), if required, may be coded to specify the total number of data or information bytes that are to be transmitted by the current IOCD. The minimum number of bytes is 1 and the maximum is 65,356 bytes (16,384 words). When the IOCD is fetched, the content of the byte count field is loaded into a byte count register within the appropriate I/O subchannel. Thereafter, the content of the byte count register is decremented by one for each byte transmitted and then tested for a zero byte count condition. (Note: As a consequence of decrementing before testing for a zero byte count condition, an initial byte count value of 0 is interpreted as 65,356 bytes.) Unless the I/O operation is terminated (e.g., as the result of detecting an "unusual end"), data is transmitted until the byte count is reduced to zero. At any time, the progress of the I/O operation may be ascertained by evaluating the current byte count which is furnished as status information when the BP executes an SIO, TIO, HIO, or TDV instruction. (That is, current byte count is equal to the number of bytes remaining to be transmitted and initial byte count minus current byte count is equal to the number of bytes transmitted.) When the byte count is reduced to zero, the MIOP may perform the following functions:

1. Transmit a "count done" signal to the device controller/device if data chaining is not specified.

2. Request an I/O interrupt, if the IZC flag is coded as a 1.

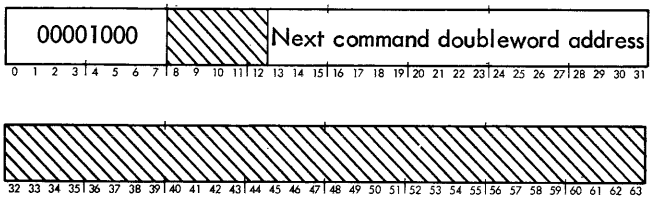
3. Fetch the next IOCD, if the data chain flag is coded as a 1.

Depending upon the characteristics of the I/O device, certain I/O orders (e.g., Rewind for magnetic tape units) may not require a byte count field. In such case, the byte count field is ignored. Refer to an appropriate Xerox peripheral equipment reference manual for further details.

CONTROL IOCD

A control IOCD may contain either a Transfer in Channel or a Stop command.

Transfer in Channel. A control IOCD containing a Transfer in Channel command has the following format:



The Transfer in Channel command is executed within the IOP and has no direct effect on any of the I/O elements external to the addressed IOP. The primary purpose of this command is to permit branching within the command list (i.e., fetching the next operational IOCD from a pair of memory word locations other than the next two consecutive word locations).

When the IOP executes the Transfer in Channel command, it loads the command address register of the appropriate I/O subchannel with the contents of bit positions 13-31 (the "next command doubleword address" field), fetches and loads the new operational IOCD into appropriate registers within the I/O subchannel and order register within the device controller/device (unless data chaining is specified), and then executes the new IOCD. (Bit positions 8-12 and 32-61 are ignored and should be coded as zeros.)

If data chaining or command chaining is specified in the IOCD preceding the IOCD containing a Transfer in Channel command, the chaining flags are not significant to nor altered by the Transfer in Channel command.

When used in conjunction with command chaining, Transfer in Channel command facilitates the control of devices such as unbuffered card punches or unbuffered line printers. For example, assume that it is desired to present the same card image twelve times to an unbuffered card punch. The punch counts the number of times that a record is presented to it and automatically generates a "chain modifier" signal when twelve rows have been punched. The command address register within the I/O subchannel is incremented by two by the "chain modifier" signal and the next consecutive IOCD within the command list is skipped over (not fetched or executed). A command list for punching two cards might be as shown in the following example:

Locations	Description of Command
.	.
.	.
.	.
A, A+1	Punch row for card 1, command chain.
A+2, A+3	Transfer in Channel to location A.
A+4, A+5	Punch row for card 2, command chain.
A+6, A+7	Transfer in Channel to location A+4.
A+8, A+9	Stop
.	.
.	.
.	.

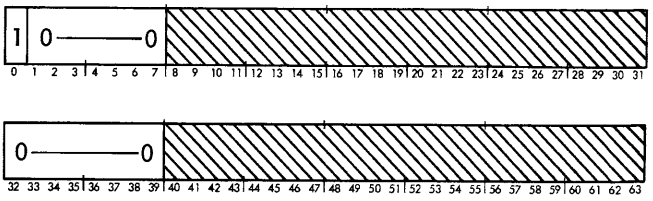
The Transfer in Channel command can be used also in conjunction with data chaining. As one example, consider a situation often encountered in data acquisition applications, where data is transmitted in extremely long, contiguous streams. In this case, the data can be stored alternately in two or more buffer storage areas so that computer processing can be carried out on the data in one buffer while additional data is being input into the other buffer. The command list for such an application might be shown in the following example:

Locations	Description of Command
.	.
.	.
.	.
B, B+1	Read data, store in buffer 1, data chain.
B+2, B+3	Store into buffer 2, data chain.
B+4, B+5	Transfer in Channel to location B.
.	.
.	.
.	.

If the IOP encounters two successive Transfer in Channel commands, an IOP control error (IOPCE) occurs and the I/O operation is terminated immediately. An IOPCE is reported as status information (bit 13 of register Rul) when the BP executes an SIO, HIO, TIO, or TDV instruction.

STOP

A control IOCD with a Stop command has the following format:



The Stop command causes certain devices to stop, generate a "channel end" signal, and also request an I/O interrupt if bit 0 in the IOCD is coded as a 1. If the I/O interrupt

level within the interrupt system (location X'5C') is armed, enabled, and not inhibited, the request will be processed by the BP in accordance with the priority that prevails within the interrupt system, the IOPs, and the I/O subchannels within an MIOP. The occurrence of an I/O interrupt because of a Stop command is reported as status information (bit position 7 of register R) when the BP executes an AIO instruction (normally part of an I/O handling routine).

Bit positions 1-7 must be coded as zeros. Bit positions 8-31 and 40-63 are ignored; but it is recommended that they also be coded as zeros. Bit positions 32-39 are device dependent and must be coded as specified in the appropriate peripheral reference manual.

The Stop command is primarily used to terminate a command chain for an unbuffered device, as illustrated in the first example given for the Transfer in Channel command. Note that not all devices recognize the Stop order.

I/O OPERATION PHASES

This section describes the general sequence of events (or phases) of any I/O operation performed by an IOP, the function performed by the BP, IOP, and device controller/device during each phase, and a description of each type of I/O operation including the applicability of parameters that may be contained within a typical operational IOCD. For explanation purposes, each I/O operation has five major phases: preparation, initiation, fetching, executing, and termination phase. Each phase is further described below.

PREPARATION PHASE

Before an I/O operation may be performed by an IOP, an appropriate command list must reside in main memory.

INITIATION PHASE

Assuming that an appropriate command list resides in main memory, an I/O operation is initiated only if the BP executes an SIO instruction that is accepted by the addressed IOP, device controller, and device. The acceptance or rejection of an SIO instruction is contingent upon conditions within the addressed IOP, device controller, and device and is indicated by the condition codes at the completion of the SIO instruction. In either case, the BP is able to perform other instructions or tasks immediately after executing an SIO instruction. (Refer to "SIO" instruction, Chapter 3, for further details.)

A successful SIO instruction causes the addressed device to go from the "ready" condition to the "busy" condition.

FETCHING PHASE

Although the services of the BP are not required during this phase, the BP may at any time execute either a TIO, TDV, or POL instruction without interfering with the I/O operation. However, excessive TIOs and TDVs may cause a data overrun condition. The BP may also execute either an HIO or RIO instruction and stop the I/O operation. (An HIO may leave the device in an unpredictable state; an RIO resets all controllers and devices on the addressed IOP.) As a result of accepting an SIO instruction, a command address register within the I/O subchannel (assigned to control the addressed device controller/device) is loaded with the first command doubleword address, the content of General Register 0 when the SIO instruction is accepted. At the appropriate time, as determined by the priority, the device controller/device will request that the IOP access main memory and fetch the first word of the IOCD from an even memory word location and increment the command address register by one. The disposition of the first word is dependent upon the contents of the first word.

If the order field contains an I/O order for a device controller/device, the content of the order field is either loaded into an order register within the appropriate device controller/device or ignored (if the IOCD is being fetched for a data chained operation). If the order is a Read Backward order, a control flag is also set within the IOP which allows the memory byte address to be decremented rather than incremented during the data transfer.

For all orders (excluding the Transfer in Channel command, described below), the contents of bit positions 10-31 of the first word is loaded into a memory byte address register of an appropriate I/O subchannel. Depending upon the I/O order, as described under "Execution Phase", the content of the memory byte address register may be used or ignored. If used, it specifies which memory word location is to be accessed and also the number of bytes of data (or control information) to be transferred into or out of that location.

If the order field contains a Transfer in Channel command, it is recognized and executed immediately by the IOP. The content of bit positions 13-31 (designated as the "next command doubleword address" field) is loaded directly into the command address register. The Transfer in Channel command is recognized and executed by the IOP, it is fetched and executed as the result of fetching one word (rather than two), and it is transparent to the device controller/device (that is, it is executed without affecting the continuity of an order that is data chained or an I/O operation that is command chained). Note: Although bit positions 0-3 and 8-12 are currently ignored, it is recommended that they be coded as zeros.

Immediately after executing a Transfer in Channel command, the IOP will automatically fetch the first word of the next IOCD as specified by the contents of the "next command doubleword address" field. If the order field of the next IOCD also contains a Transfer in Channel command, the I/O operation is terminated immediately and the IOP enters a Halt state because an IOP control error (IOPCE) occurred (attempting to execute two successive Transfer in Channel commands).

Otherwise, the first word of the next IOCD is fetched and loaded as described above, and the second word is fetched and loaded as described below.

Since the Transfer in Channel command permits IOCDs to be fetched from nonconsecutive locations, IOCDs containing Transfer in Channel commands may be included within a command list either to achieve command list continuity from one segment of a command list to another segment or to construct reiterative loops.

For all IOCDs, except a control IOCD containing a Transfer in Channel command, the IOP will automatically access main memory at the appropriate time, as determined by the priority that prevails for accessing main memory, and fetch the second word of the IOCD from the next consecutive ascending (odd) memory word location of the command list and increment the command address register by one. Thus, in all cases, after a fetching operation is completed, the content of the command address register will be an even (or doubleword) address.

The contents of the second word are stored in appropriate registers within the I/O subchannel. Depending upon the I/O order, as described under "Execution Phase", the contents of the various fields are either used or ignored.

In addition to the IOP Control Error (IOPCE), the following types of "unusual end" conditions may be detected during the fetching phase of an I/O operation: Memory Address Error (MAE), Control Check Fault (CCF), IOP Memory Error (IOPME), Bus Check Fault (BCF), and Memory Interface Error (MIE). The detection of any of these errors causes the I/O operation to be terminated and if the IUE flag is set to a 1, an "unusual end" interrupt is requested.

EXECUTION PHASE

Although the services of the BP are not required during this phase, the BP may at any time execute either a TIO, TDV, or POL instruction without interfering with the I/O operation. However, excessive testing may cause a data overrun condition. The BP may also execute either an HIO or RIO instruction and stop the I/O operation. After the second word of an IOCD is fetched and providing no "unusual end" condition was detected, the IOCD is executed as prescribed by the parameters contained therein. As a function of the order and the status of the Skip flag, if applicable, an IOCD may be executed in one of five ways, as described below:

1. Certain Control orders (e.g., Stop) may be executed by the device while the IOP monitors the operation in accordance with the applicable control flags. Since no memory accesses and data (or information) transfers occur, the contents of the memory byte address register, write key register, and byte count register may be ignored. Other Control orders (e.g., Rewind for a magnetic tape unit) are listed and described in applicable Xerox peripheral equipment reference manuals.

   Depending upon the control function performed, certain Control orders may be a part of an I/O operation which may be continued after the Control order is executed. For example, an I/O operation involving a magnetic tape unit may contain a Rewind order to reposition the tape prior to reading (or writing) one or more records.

   Note: Within the context of the above explanation, the Control order is defined to be one that does not transfer any information; thus, data chaining is precluded within the IOCD containing the Control order; however, command chaining may be specified. Control orders that involve information transfers when executed are described below (see paragraphs 2 and 4).

2. If the order specifies an input operation (e.g., Read, Read Backward, or Sense) and the Skip flag is coded as a 0, all parameters of the current IOCD may be applicable. As a result of receiving an appropriate input order, the device transmits data (Read, or Read Backward order) or information from special registers (Sense order) into data buffers of the associated I/O subchannel within the IOP.

   Depending upon the priority that prevails for accessing main memory, the IOP accesses a memory word location (as specified by the current memory byte address), transfers up to four bytes of data or information from the data buffers to a memory unit, provides a write key, and increments (or decrements, if Read Backward order) the memory byte address and decrements the byte count by one for each byte transferred out of the data buffers.

   The write key is evaluated against the preassigned write lock for the memory word location accessed. If the write key is valid for each memory word location accessed, the input operation continues, as described above, until it is completed or terminated by an "unusual end" condition, other than Write Lock Violation. If the write key is not valid, the memory unit (1) generates and transmits a Write Lock Violation (WLV) signal to the IOP, (2) rejects the new data, and (3) does not disturb the previous contents of the memory word location accessed.

   If the write key is invalid for any memory word location accessed and the HTE flag is coded as a 1, the input operation is terminated immediately upon receipt of a WLV signal (see "Termination Phase").

   If the HTE flag is coded as a 0, the memory unit may accept or reject the data or information, based on the write key/write lock evaluation for each memory word location accessed, without affecting the operations within the IOP, device controller, or device. The input operation continues until either completed or terminated by an "unusual end" condition, other than a Write Lock Violation.

Note: Since the same write key prevails for the entire IOCD and all memory locations within a memory page are assigned the same write lock, the write key/write lock relationship may change when the memory byte address is incremented (or decremented) across a memory page boundary.

3. If the order specifies an input operation (e.g., Read, Read Backward, or Sense) and the Skip flag is coded as a 1, all parameters within the IOCD, except the write key, may be applicable. As a result of receiving an appropriate input order, the device transmits data (Read or Read Backward order) or information from special registers (Sense order) into the data buffers within the I/O subchannel of the IOP. Because the Skip flag is coded as a 1, the IOP can not access main memory (the write key may be ignored and a Write Lock Violation can not occur). Although the data can not be stored in the main memory, the IOP increments the memory byte address (except during a Read Backward order, when it is decremented) and decrements the byte count by one for each byte transferred out of the data buffers. The device may continue to transmit data into the data buffers and the IOP may continue to update the memory byte address and byte count until the current order is either completed in a normal manner or terminated because of an "unusual end" condition (other than a Write Lock Violation).

4. If the order specifies an output operation (e.g., Write or Control) and if the Skip flag is coded as a 0, all parameters within the IOCD, except the write key, may be applicable. When transferring data (Write order) or information (Control order) out of main memory, the write key/write lock checking is not performed; hence, the write key may be ignored. Likewise, a Write Lock Violation will not occur. For an output operation, the IOP will access main memory (in accordance with the priority that prevails for accessing main memory) and transfer up to four bytes of data (or information), as specified by the current memory byte address, to the data buffers of the appropriate I/O subchannel. The IOP also increments the memory byte address and decrements the byte count by one for each byte of data transferred. Data is then transferred from the data buffers to the device. The IOP may continue to access main memory, transfer up to four bytes of data from main memory to the appropriate data buffers, and update the memory byte address and byte count. The device continues to output data until the order is either completed in a normal manner or terminated because of an "unusual end" condition.

5. If the order specifies an output operation (e.g., Write or Control) and if the Skip flag is coded as a 1, all parameters within the current IOCD, except the write key, may be applicable. Because the Skip flag is coded as a 1, the IOP can not access main memory for any data (or information). Instead, the IOP generates and loads zeros (X'00') into the data buffers of the appropriate I/O subchannel and increments the memory byte address and decrements the byte count by one for each byte loaded. The zeros are then transferred from the data buffer to the device. The IOP may continue to generate and load zeros into the data buffers and update the memory byte address and byte count, accordingly, and the device may continue to output zeros until the order is either completed in a normal manner or terminated because of an "unusual end" condition.

DATA CHAINING

An order may be continued from the current operational IOCD to the next operational IOCD, if data chaining is specified in the current IOCD. In this case, the IOP will automatically fetch the next operational IOCD, as described under "Fetching Phase", when the byte count of the current IOCD is reduced to zero. In the process of fetching the next operational IOCD, the IOP may fetch and execute a control IOCD containing a Transfer in Channel command without affecting the continuity of the order. The process of fetching and loading the next operational IOCD into the control registers of the I/O subchannel is transparent to the device. That is, the device continues to operate as if the order were defined by a single IOCD. Also, any changes in the status of the Skip flag or in the write key from one IOCD to the next is transparent to the device. The device continues to receive zeros, data, or information from the data buffers during an output operation, or continues to transmit data (or information) into the data buffers regardless of whether it is subsequently rejected or stored while performing an input operation.

During the execution phase, an I/O interrupt may be requested each time the byte count of an operational IOCD is reduced to zero if the Interrupt at Zero Byte Count (IZC) flag is coded as a 1. Thus, if data chaining is specified, the IOP may request an I/O interrupt without interfering with the process of fetching the next operational IOCD.

If the I/O interrupt level (location X'5C') within the interrupt system is armed, enabled, and not inhibited, the I/O interrupt may be processed by the BP in accordance with the priority that prevails within the interrupt system, the IOPs, and the device controllers connected to the IOP.

The order may be completed in a normal manner when the Data Chain flag of the current IOCD (the last IOCD of a logical record) is coded as a 0.

COMMAND CHAINING

An I/O operation may be continued from the current IOCD to the next IOCD if command chaining is specified in the current IOCD. Command chaining is commonly specified when reading (or writing) consecutive records of data from the same file. In which case, the current IOCD must be the last IOCD for the current record and the next IOCD must be the first IOCD of the next logical record. Although the device may execute the same functional order for both records, logically, it is equivalent to two separate orders.

Depending upon the characteristics of the device, command chaining may also be used to perform different operations on either different but consecutive records or upon the same record (e.g., a magnetic tape unit may be programmed to alternately read or write consecutive records or to read the same record backwards after writing). Refer to an appropriate Xerox peripheral equipment reference manual for further details.

If command chaining is specified, the device controller causes the IOP to fetch the next operational IOCD, as described under "Fetching Phase", when the device signals "channel end" (signifying that it is ready to accept and execute another order). In the process of fetching the next operational IOCD, the IOP may fetch and execute a control IOCD containing a Transfer in Channel command without affecting the continuity of the I/O operation (i.e., transparent to the device controller/device); however, the fetching of the next operational IOCD is not transparent to the device controller/device. The process of automatically fetching the next operational IOCD because data chaining and/or command chaining is specified in the current IOCD permits an I/O operation to continue normally until an IOCD is executed in which both chaining flags are coded as zeros (the last IOCD of the last record).

If data chaining and command chaining are both specified within an IOCD, data chaining is performed if the byte count of the current IOCD is reduced to zero before the device generates "channel end"; command chaining is performed if the device generates "channel end" before the byte count is reduced to zero.

During the execution phase, an I/O interrupt may also be requested each time a "channel end" occurs if the Interrupt at Channel End (ICE) flag is coded as a 1. Thus, if command chaining is specified, the IOP may request an I/O interrupt without interfering with the process of fetching the next operational IOCD.

TERMINATION PHASE

An I/O operation may be terminated in one of the following manners:

1. Aborted at any time because the BP executed either an HIO or RIO instruction.

2. Aborted when an unconditional "unusual end" condition was detected.

3. Aborted when a conditional "unusual end" condition was detected while the HTE control flag was coded as a 1.

4. Completed as specified by the command list but with an "unusual end" condition.

5. Completed as specified by the command list.

6. Aborted whenever a SUPER RESET, SYSTEM RESET, or I/O RESET command is entered from the System Control Console (SCC).

The progress of an I/O operation, including the termination, may be ascertained by evaluating the status information returned for I/O instructions, as described in Chapter 3. Depending upon programming considerations, these I/O instructions may be executed either singly or as part of an I/O handling routine and either imperatively at logical points of a BP-executed program or on an "as needed" basis when an I/O interrupt is requested by an IOP or device controller. Normally, an I/O interrupt is requested whenever a critical or significant event occurs within any I/O subchannel, device controller, or device. Typically, an I/O interrupt may be requested when the byte count of any IOCD is reduced to zero, whenever any device detects a "channel end" condition, or when the IOP or any device controller detects an "unusual end" condition, providing the appropriate control flag (IZC, ICE, and IUE) is coded as a 1.

Note: An I/O interrupt may also be requested by certain devices, e.g., a magnetic tape unit may be able to execute a Rewind and Interrupt order and other devices may request an I/O interrupt when executing a Stop order in which bit 0 is coded as a 1. Refer to an appropriate Xerox peripheral reference manual for further details.

Once an I/O interrupt request has been made by a device, that device, device controller, and I/O subchannel remain in an interrupt pending condition until the interrupt request is acknowledged, reset, or cleared.

Normally, an I/O interrupt request is acknowledged by the BP executing an AIO instruction, as part of an I/O interrupt-handling routine; reset by the BP executing either an HIO or an RIO instruction; or for certain devices cleared automatically, as a function of time. Refer to an appropriate Xerox peripheral equipment reference manual for further details.)

Since a multiple number of I/O interrupt requests may prevail simultaneously (one per each device controller) and all requests are serviced by a common I/O interrupt level (location X'5C'), the BP normally acknowledges an I/O interrupt request based on the priority that prevails within the interrupt system, the IOPs, and the I/O subchannels within an MIOP, if applicable. An interrupt pending condition prevents a new I/O operation from being initiated by an SIO instruction on a particular subchannel but does not affect the current I/O operation. (That is, if an I/O interrupt was requested as the result of a zero byte count or "channel end" condition, and data chaining or command chaining is specified, the I/O operation may continue as specified by the command list.)

5. OPERATIONAL CONTROL

EXTERNAL CONTROL SUBSYSTEM

The External Control Subsystem (ECS) is a group of elements used in this computer system that provide operational and diagnostic interfaces to control and maintain system hardware and software.

CENTRALIZED SYSTEM CONTROL

In many other computer systems "software-level" operator interactions are transacted through an operator's teletypewriter console while hardware level interactions are performed through a fixed panel of lamps and switches. In contrast, this Xerox computer system consolidates these interactions and controls into a console telecommunications device, designated as the System Control Console (SCC). Through the SCC, the operator has a single control point for all normal system control activities.

A Remote Diagnostic Interface (RDI) permits the local System Control Console to be augmented with a Remote Console that may have the same degree of system control. (Usage of the RDI and Remote Console as a Remote Assist feature is described below, under "Remote Console".)

A System Control Panel (SCP) contains indicators and basic controls that the operator may use during system startup or to establish connections with the remote location.

CONTROL CONSOLE DEVICES

The ECS provides an interface for two local (primary and alternate) communications consoles and a data set interface for remote diagnostic connection. Each communications console must have an EIA RS232 voltage interface and format characters in even parity ASCII code with control protocols of a Model 4691 KSR 35 Keyboard/Printer. Allowed communications rates are 10 and 30 characters per second.

PRIMARY CONSOLE

The primary console always has the functional capability of the System Control Console to communicate with software through I/O subchannel address X'01'. The communications rate of the primary console is either 120 characters per second or the same as the alternate and remote consoles depending on the setting of the FSELA switch on the Configuration Control Panel (see Chapter 6). If the REMOTE CHANNEL switch on the System Control Panel is in the SCC position (implying a remote diagnostic connection), the remote channel frequency is automatically enforced on the primary console.

REMOTE CHANNEL

The alternate and remote consoles share the same data paths. Both consoles receive the same output; either one of the consoles is selected for input by the ALTSEL switch on the Configuration Control Panel. The communications rates of 10 or 30 characters per second are selected for both consoles by the FSELB0 and FSELB1 switches on the Configuration Control Panel. Both consoles may function either strictly as I/O devices or as parallel System Control Consoles selected by the REMOTE CHANNEL switch on the System Control Panel. Description of communications rate selection is found in Chapter 6.

ALTERNATE CONSOLE

The alternate console normally functions as an output device residing at I/O subchannel address X'0B'. This console can create an edited system log, while the operator's console functions at a higher communications rate. (REMOTE CHANNEL and ALTSEL switches are both OFF.)

If the primary console fails, the alternate console may function as the System Control Console. In this case, the remote console connection is only inhibited by the operator at the data set. (REMOTE CHANNEL switch in SCC position; ALTSEL switch in ON position.)

REMOTE CONSOLE

Before the remote device can gain access to the Remote Diagnostic Interface (RDI), the operator must manually intervene to establish the connection at the data set and the System Control Panel. The data set (Bell 103A or equivalent) connection is inhibited while the REMOTE CHANNEL switch is in the OFF position.

The remote console may run on-line diagnostics while the rest of the system performs non-maintenance work. In this case, the remote console preempts I/O subchannel X'0B' and the alternate (local) console creates a log of the on-line mainentance if not turned off. The remote device does not have access to the SCC hardware controls, but may enter software-level control information through the I/O system (REMOTE CHANNEL switch in I/O position, ALTSEL switch in OFF position).

If the entire system is under the discretionary control of remote maintenance personnel, the operator may connect the remote console to the RDI as the System Control Console. The remote console is then connected logically in parallel, and assumes all the functional capability of the primary console, and shares I/O subchannel X'01'. (Note that conventions must be established to ensure that the primary and remote consoles do not generate overlapping input.) The remote console communications rate is automatically imposed on the primary console and the operator may have to

change the rate on the primary console to retain parallel control. The alternate (local) device creates a log of all SCC transactions. The normal (log) output on I/O subchannel X'0B' is suspended for the duration of the SCC assignment to the remote channel (REMOTE CHANNEL switch in SCC position; ALTSEL switch in OFF position).

CONTROL COMMANDS

A set of commands and display formats implements operator communication with hardware through the System Control Console. These hardware-control commands, called "SCC Functions", are independent, direct hardware controls as distinguished from the software-level operating controls activated from the SCC through the normal I/O system. A special micro-processor, working independently of the BP, senses and controls the execution of SCC functions. The flexibility of character-oriented communications equipment and micro-programmed control significantly enhance many system operating and diagnostic features.

The basic command format provides a four-level interlock on critical system controls by requiring a correct four-character sequence to initiate a command action. In addition, context analysis is provided to assure that commands are executed only in appropriate system states. This basic format requires that each command is preceded by the "control-Z" character (control and Z keys depressed simultaneously). Note that within this text, the control-Z character is represented with the symbol "Z^C".

A typical command sequence is to enter "Z^C HLT" from the SCC. The system responds by printing "(HLT)" on the next line of the SCC printout, and forcing the system to halt instruction execution and enter the IDLE state. If a command cannot be executed due to improper syntax or context, the system provides an advisory message following the command echo indicating the probable source of error. A typical example of the display format is "(RSY) *EVENT A1*", indicating that a reset command may not be executed prior to halting instruction execution. (Refer to Table 21 for a complete listing of event messages.)

The various control functions that may be exercised from the SCC may be generally classified into three categories: operator control commands, diagnostic control commands, and maintenance control commands.

OPERATOR CONTROL COMMANDS

These commands provide controls which an operator normally uses to control the computer system. By entering the appropriate command the operator may direct the computer system to load, run, halt, reset, read/set the sense switches, or issue a "console interrupt" to the operating software.

The sense switch control and console interrupt commands may interact with the software and are always operative. All other SCC functions may be enabled or disabled by the SCC FUNCTIONS switch on the SCP.

Table 21. Event Messages

Display	Significance
EVENT 00	System Initialization; POWER ON or SUPER RESET.
EVENT A0	Improper syntax for Z^C format command.
EVENT A1	Command not executed; Improper syntax or system may not be in IDLE mode.
EVENT A2	Command not executed; system not in maintenance mode.
EVENT A4	Command not executed; SCC FUNCTION switch is in DISABLE position.
EVENT A8	Power ride through; recoverable power line failure detected; power on trap requested.
EVENT F0	Trap requested occurred; inhibited in P-Mode.
EVENT F1	Basic processor error halt; watchdog timeout reset issued when watchdog timeout alarm bit set. (See "Processor Control Word".)
EVENT F4	Basic processor halt; Address Halt.
EVENT F6	Basic processor halt; Processor-Detected Fault (PDF).
EVENT F9	System failed micro-diagnostic test (followed by Single Clock Status Register display of the element that failed).

To prevent inadvertent activation from disrupting a running system, the SCC FUNCTIONS switch is placed in the DISABLE position.

The following operator control commands are standard features of this system:

Input	Display	Name of Command
Z^C I	(I)	Operator's Console Interrupt
Z^C SSW	(SSW=bbbb)	Read Sense Switches
Z^C SS#[t]	(SS#[t]=bbbb)	Set Sense Switches
Z^C LDN####[t,tt,ttt]	(LDN@####[t])	Load Normal
Z^C RSY[tt,ttt]	(RSY)	Reset System
Z^C RBP[tt,ttt]	(RBP)	Reset Basic Processor
Z^C RIO[tt,ttt]	(RIO)	Reset I/O System
Z^C HLT[tt]	(HLT)	System Halt
Z^C RUN[tt,ttt]	(RUN)	System Run

ZCI OPERATOR'S CONSOLE INTERRUPT

The Operator's Console (or SCC) INTERRUPT command permits the operator to interact with the executing software by setting interrupt level X'5D'. If this interrupt level is Armed when the INTERRUPT command is entered, the interrupt level is advanced to the Waiting state. If the interrupt level is already in the Active state or Disarmed, the INTERRUPT command has no effect upon the computer system. This command is always enabled.

ZCSSW READ SENSE SWITCHES

This command causes the status of the sense switches to be displayed as part of the command echo. For example, if all four sense switches were set to a 1, the console display would be "(SSW=1111)". The status of the sense switches is also displayed by indicators on the System Control Panel. The READ SENSE SWITCHES command is always enabled.

The status of the sense switches may also be read by executing a READ DIRECT instruction (see Chapter 3).

ZCSS# SET SENSE SWITCHES

This command causes the sense switches to be set to the value specified by the hexadecimal digit in the command (#). The new sense switch value is displayed as part of the command echo. For example, if the operator enters "ZCSS3" the SCC will print "(SS3=0011)". The new status is also displayed by indicators on the System Control Panel. The SET SENSE SWITCHES command is always operative.

The sense switches may also be set by executing a WRITE DIRECT instruction (see Chapter 3). The sense switches are initialized to zero during the power on and SUPER RESET sequences. While the ZCSS# command is active, the basic processor is momentarily put in the IDLE state. This prevents any conflict between the operator command and a WRITE DIRECT instruction.

ZCLDN#### LOAD NORMAL

The loading operation is normally accomplished by readying the load device and entering the LOAD NORMAL command from the System Control Console. The four hexadecimal digits (represented as ####) specify the load device address. Successful completion of the command is signified by the command echo "(LDN@####)". A failure in the load sequence is indicated by a display of an appropriate error message (see Table F-) following the command echo. The LOAD NORMAL command is accepted only when the system is in the IDLE state.

This single command initiates the following sequence:

1. A series of internal micro-diagnostic tests are conducted to verify the operation of system paths and elements used in the loading sequence. Each test is preceded by a system reset. If a failure is detected during the micro-diagnostic tests, an error message "*EVENT F9*" is generated and followed by a Single Clock Status Register display identifying the failing element.

2. Upon completion of all micro-diagnostic tests, a system reset is issued.

3. All system memory locations are initialized to zero.

4. The basic processor loads a self-diagnostic program in memory locations X'100' through X'1FF' and loads the bootstrap loader (see Figure 14) in memory locations X'20' through X'29'. If an error is detected during the process, an error message "*EVENT F0*", is generated.

5. The system is placed in the RUN mode.

6. The basic processor executes the self-diagnostic program, beginning at location X'160'. The processor then executes the bootstrap loader, starting at location X'26'. If a failure occurs during the processor self-diagnostic program, the processor enters the WAIT state.

---

[t] Hexadecimal digits.

[tt] SCC FUNCTIONS switch of SCP must be in the ENABLE position.

[ttt] System must be in the IDLE state.

Location (hex)	(dec)	Hexadecimal	Symbolic form of instruction	
20	32	020000A8		
21	33	0E000058		
22	34	22110029	LI, 1	
23	35	64100023	BDR, 1	
24	36	68000028	BCR, 0	40
25	37	0000####†		
26	38	22000010	LI, 0	
27	39	CC000025	SIO, 0	*37
28	40	CD000025	TIO, 0	*37
29	41	69C00022	BCS, 12	34

†#### represents four hexadecimal digits that specify the load device address as entered by the LOAD NORMAL command.

Figure 14. Bootstrap Loader

Execution of the bootstrap program causes the following actions:

1. The first record on the selected peripheral is read into memrory locations X'2A' through X'3F' (the previous contents of general register 0 are destroyed as a result of executing the bootstrap program in locations X'26' through X'29').

2. After the record has been read, the next instruction is taken from location X'2A' (provided that no error condition has been detected by the device or the (IOP).

3. When the instruction in location X'2A' is executed, the unit device and device controller selected for the load operation can accept a new SIO instruction.

4. Further I/O operations from the load unit may be accomplished by coding subsequent I/O instructions to indirectly address location X'25'.

Following the successful completion of the load sequence, the computer system usually continues execution of the loaded program and begins issuing messages to the operator via the I/O system to the System Control Console.

Z^CRSY RESET SYSTEM

The RESET SYSTEM command performs the combined functions of the RESET BASIC PROCESSOR and RESET I/O SYSTEM commands, as well as the function described below:

1. The system control processor bus interface is initialized.

2. The processor memory bus and processor bus interfaces are initialized.

3. The system memory units are initialized. This process does not alter any memory locations.

4. All interrupt levels are reset to the Disarmed and Disabled state.

5. The system ALARM indicator is cleared.

This command is accepted only when the system is in the IDLE state.

Z^CRBP RESET BASIC PROCESSOR

The RESET BASIC PROCESSOR command initializes the basic processor by performing the following:

1. All bits in the Program Status Words, except the instruction address, are reset.

2. The program counter of the BP (register Q5) is set to a value of X'26'.

3. The BP remains in the IDLE state until allowed to begin execution at location X'26'.

This command is accepted only when the system is in the IDLE state.

Since all memory requests are inhibited during a reset, the RESET BASIC PROCESSOR command disrupts any simultaneous memory request from the standard I/O system.

Z^CRIO RESET I/O SYSTEM

When accepted, the RESET I/O SYSTEM command initializes the IOPs and device controllers of the standard I/O system. All peripheral devices under control of the system are reset to the "ready" condition and all status, interrupt, and control indicators in the I/O system are reset. This command is accepted only when the system is in the IDLE state. The RESET I/O SYSTEM command does not affect the External Direct Input/Output (DIO), the BP, or other non-input/output system elements.

Z^CHLT SYSTEM HALT

When the HALT command is entered, the BP ceases to execute instructions and is forced into the IDLE state; the RUN indicator on the System Control Panel is extinguished

and the IDLE indicator is illuminated. In the IDLE state the load commands, the reset commands, and the RUN command are enabled. The I/O system may continue to perform I/O operations initiated prior to the Z^CHLT command, even though the BP is halted. Note that the processor HALT status is not set by the Z^CHLT command, but is caused by internal processor conditions (see "Processor-Control Word").

Z^CRUN SYSTEM RUN

The RUN command is accepted only if the BP is in the IDLE state. When the FUN command is accepted, the BP is allowed to execute its instruction stream. On the SCP, the IDLE indicator is extinguished and the RUN indicator is illuminated, subject to the processor control word and system status.

When not in the IDLE state, the system does not accept any of the load and reset control commands. Attempting to enter any load and reset control command while the system is in the RUN mode results in an error message being displayed on the control console (see Table F-).

DIAGNOSTIC CONTROL COMMANDS

Diagnostic control commands facilitate isolating software and hardware problems by providing single-instruction execution, as well as permitting read/write access to many processor internal control registers and system memory locations. To perform diagnostic commands, BP instruction execution must be interrupted and the ECS control mode altered. This is accomplished by the ENTER P-MODE command (a "CONTROL-P" character generated by depressing CONTROL and P keys simultaneously). Once in P-Mode the system is forced into the IDLE state and the BP stores and fetches data or executes single instructions only upon request from the operator through the SCC.

Note: Within this text the control-P character is represented by the following symbol, P^C.

The diagnostic control (P-Mode) command format differs from the basic command format. Hexadecimal digits are immediately echoed and stored to be used as data or address depending on the following command. The system truncates the data stream to eight hexadecimal digits and assumes leading zeros if less than eight characters are entered. All non-hexadecimal characters, except basic (Z^C) format commands, are treated as P-Mode commands. If the character is not in the allowed command set, it is echoed followed by a question mark "N?" and no action results. Valid commands are echoed; the requested operation is then performed and a P-Mode data display of the form "P:DDDDDDDD @ AAAAAAAA" is generated on the next line of SCC printout. The "P" represents the processor address (normally 0); the "D" field (eight hexadecimal digits) represents the data in the location specified in the "A" field (eight hexadecimal digits). The first hexadecimal digit of the A field is X'0' for memory addresses and X'8' for internal register addresses. All valid commands, except EXIT P-MODE, produce a display in this format.

The allowed diagnostic command set is listed in Table 22. An example of the resulting printout is shown in the section entitled "Operating Procedures and Information".

P^C ENTER P-MODE

The ENTER P-MODE control command is generated by depressing the CONTROL and P keys, simultaneously (P^C). The system is forced into the IDLE state and the processor will execute diagnostic control commands entered from the System Control Console. The ECS remains in the P-Mode until an EXIT P-MODE command (described below) is entered or the Z^C format commands SYSTEM RUN or LOAD NORMAL are entered. Successful entry into the P-Mode is indicated by a P-Mode display on the SCC.

(P-Mode)
SELECT INTERNAL REGISTER ADDRESSING

(P-Mode)
/ SELECT MEMORY ADDRESSING

These commands specify the storage element whose contents are to be displayed and operated upon with subsequent commands. The "/" character following a hexadecimal data stream specifies a memory address; the "." character specifies an internal processor control register address. All address calculations and memory accesses are subject to the write lock keys, address mode, and mapping bits in the program status words.

(P-Mode)
+ ADD TO SELECTED LOCATION

The "+" character, following a hexadecimal data stream, causes the value of the data to be added to the contents of the selected storage element.

(P-Mode)
− SUBTRACT FROM SELECTED LOCATION

The "-" character, following a hexadecimal data stream, causes the value of the data to be subtracted from the contents of the selected storage element.

(P-Mode)
M STORE IN SELECTED LOCATION

The "M" character, following a hexadecimal data stream, causes the data to be stored in the selected storage element.

Table 22. Diagnostic Control (P-Mode) Commands

Character	Function
P^C	ENTER P-MODE.
####...##	Input data or address value (context determined by the succeeding operator. ####...## is any hex digit string).
.	SELECT INTERNAL REGISTER ADDRESSING.
/	SELECT MEMORY ADDRESSING.
+	ADD TO SELECTED LOCATION.
-	SUBTRACT FROM SELECTED LOCATION.
M	STORE IN SELECTED LOCATION.
L	SHIFT LEFT AND DISPLAY.
R	SHIFT RIGHT AND DISPLAY.
I	INCREMENT REFERENCED ADDRESS AND DISPLAY.
RUBOUT	DISPLAY ADDRESSED LOCATION.
S	INSTRUCTION SINGLE STEP.
G	SPECIAL INSTRUCTION SINGLE STEP.
X	EXIT P-MODE.

(P-Mode)
L SHIFT LEFT AND DISPLAY

This command causes an image of the contents of the presently selected memory location or Q register to be shifted one bit position to the left and then displayed. A zero is entered into the least significant bit of the location for each L command.

Actual contents of the memory or Q-register location referenced by the shift instruction are not altered.

(P-Mode)
R SHIFT RIGHT AND DISPLAY

This command causes an image of the contents of the presently selected memory location or Q register to be shifted one bit position to the right and then displayed. A zero is entered into the most significant bit of the memory location or Q register for each R command executed.

Actual contents of the memory or Q register location referenced by the shift instruction are not altered.

(P-Mode)
I INCREMENT REFERENCED ADDRESS AND DISPLAY

This command increments by +1 the address of the currently selected memory location or Q register (as specified by a previously executed SELECT MEMORY ADDRESSING or SELECT INTERNAL REGISTER ADDRESSING control command). The new address and contents are displayed on the next line.

(P-Mode)
RUB OUT DISPLAY ADDRESSED LOCATION

This command displays the contents of the currently addressed memory location or Q register (as specified by a previously executed SELECT MEMORY ADDRESSING or SELECT INTERNAL REGISTER ADDRESSING control command).

(P-Mode)
S INSTRUCTION SINGLE STEP

This command causes the BP to execute a single instruction as pointed to by the current contents of the program counter. Execution is precisely the same as if the system were running continuously. Upon completion, the BP returns to the IDLE state. If a trap occurs while the instruction is being executed, the instruction in the trap location is executed before the BP returns to the IDLE state. The resultant display shows the next instruction to be executed.

Condition codes resulting from the instruction execution are displayed as the second hexadecimal digit of the address field.

(P-Mode)

G SPECIAL INSTRUCTION SINGLE STEP

This command permits the contents of register Q5 to be interpreted as the current instruction, and execution by the BP proceeds as described for the INSTRUCTION SINGLE STEP command. The program counter is incremented by +1. This command thus allows any single instruction (contained in register Q5) to be executed in lieu of the instruction pointed to by the program counter without otherwise disturbing conditions within the system. The resultant display shows the next instruction to be executed.

Condition codes resulting from the instruction execution are displayed as the second hexadecimal digit of the address field.

(P-Mode)

X EXIT P-MODE

The EXIT P-MODE command terminates the P-Mode within the ECS. The BP resumes execution of instructions. If no SYSTEM RUN or LOAD NORMAL commands were in effect before entering the P-Mode, the system remains in the IDLE state.

MAINTENANCE CONTROL COMMANDS

Maintenance control commands facilitate isolation and analysis of system hardware malfunctions. The commands are accepted only if the SCC FUNCTIONS switch is in the ENABLE position. In addition, most critical maintenance controls can be activated only if the MAINT MODE switch on the SCP is in the ON position.

The primary features of the maintenance control commands are the provision of system clock control and single clock status displays. Status is obtained from read-only registers located in central system elements. These Single Clock Status Registers (SCSR) monitor the state of internal hardware signals. Each SCSR display is printed on the next line of SCC printout in the format "CE:DDDDDDDD CC". The "CE" field contains two hexadecimal digits that represent a cluster and an element address, respectively. The 8-digit D field displays the contents of the register, and the 2-digit "CC" field is a modulo 256 clock step counter. This information is valid only when the system clock is stopped.

The following maintenance control commands are included as standard features of this computer system:

Input	Display	Name of Command
$Z^{C}MM0$	(MM0)	CLEAR MM FEATURES
$Z^{C}MM1^{t}$	(MM1)	SET/CLEAR REPEAT CLOCKING MODE
$Z^{C}MM2$	(MM2)	SET/CLEAR CLUSTER DISPLAY MODE
$Z^{C}MM3$	(MM3)	SET/CLEAR P-MODE REPEAT MODE
$Z^{C}MM4^{tt}$	(MM echo interrupted)	SUPER RESET
$Z^{C}MM5^{tt}$	(MM5)	SET MICRO-DIAGNOSTIC LOOP MODE
$Z^{C}MM6^{tt}$	(MM6)	INITIATE ELEMENT MICRO-DIAGNOSTIC
$Z^{C}MM7^{tt}$	(MM7)	SET LOW CLOCK MARGINS
$Z^{C}MM8^{tt}$	(MM8)	SET HIGH CLOCK MARGINS
$Z^{C}MM9^{tt}$	(MM9)	SET MEMORY INTERLEAVE OVERRIDE
$Z^{C}MMA^{tt}$	(MM echo interrupted)	SET DISPLAY INHIBIT MODE
$Z^{C}CLK^{tt}$	(CLK)	SET SINGLE CLOCK MODE
'space'^{t}	'space'	SINGLE-CLOCK STEP
$Z^{C}C\#\#^{t}$	(C##)	MULTIPLE-CLOCK STEP
$Z^{C}KIL$	(KIL)	CLEAR SINGLE CLOCK MODE
$Z^{C}E\#\#$	(E##)	SELECT/DISPLAY SINGLE CLOCK STATUS REGISTER
$Z^{C}T$	(T)	SET/CLEAR TRANSPARENT TEXT MODE
$Z^{C}LDS\#\#\#\#$	(LDS@####)	LOAD SPECIAL
$Z^{C}LDT$	(LDT)	MEMORY SET

Z^{C}CLK SET SINGLE CLOCK MODE

This command sets the computer system to the "Single Clock Mode" by simultaneously stopping all central system clocks, except those required by the External Control Subsystem and the I/O system. When the system is in the Single Clock Mode all control commands may be entered and executed. Operations performed in the Single Clock Mode may differ from those performed when the clock is running

---

[t] All clock controls are inhibited unless the MAINT MODE switch is in the ON position.

[tt] These commands are accepted only if the system is in the MAINT MODE.

at its normal rate (e.g., fixed duration control sequences may not take effect and diagnostic control commands which operate upon BP's registers or memory locations require a large number of clock steps to complete the operation). The RESET SYSTEM, RESET I/O, and RESET BASIC PROCESSOR commands are effective in Single Clock Mode. When the Single Clock Mode is set, the contents of the currently selected Single Clock Status Register are displayed (see SELECT/DISPLAY SINGLE CLOCK STATUS REGISTER, Z^CE## command).

If the computer system is currently in the Single Clock Mode, Z^CCLK command resets the two-digit clock step counter to X'00'.

The Single Clock Mode may be reset by either a CLEAR SINGLE CLOCK MODE, Z^CKIL, command or a SUPER RESET, Z^CMM4, command.

Note: Entering a SET SINGLE CLOCK MODE command when the basic processor is performing normal data processing operations may have an adverse effect upon I/O operations. To prevent inadvertent entry into Single Clock Mode, the Z^CCLK command is not accepted unless the MAINT MODE switch is in the ON position. Attempting to enter a Z^CCLK command when the MAINT MODE switch is in the OFF position results in an error message (*EVENT A2*) being displayed and no further action.

␣ SINGLE-CLOCK STEP

When the system is in the Single Clock Mode, a single space character (depicted as ␣ within this text), without a control-Z, is interpreted as a SINGLE-CLOCK STEP command. For each space character received from the control console, the current command or instruction is partially executed (one clock's worth of execution for each space character).

The contents of the currently selected Single Clock Status Register are displayed and the clock step counter is incremented by +1 for each SINGLE-CLOCK STEP.

Z^C## MULTIPLE-CLOCK STEP

When the system is in the Single Clock Mode, the MULTIPLE-CLOCK STEP command causes the current instruction or command to be executed for 1 to 256 clock steps (as specified by the two hexadecimal digits "##" in this command). The Z^C00 command causes 256 clocks to be issued. The contents of the currently selected Single Clock Status Register are displayed. The clock step counter is incremented by the number of clock steps specified by this command.

The MULTIPLE-CLOCK STEP command allows precise stepping to a specific point in a micro-program sequence involving a large number of clock steps.

Z^CE## SELECT/DISPLAY SINGLE CLOCK STATUS REGISTER

This command causes the requested SCSR to be displayed. The "##" portion of this command (two hexadecimal digits) is stored within the ECS and used as a reference address in any command which displays the contents of the currently selected Single Clock Status Register. The first hexadecimal digit is the cluster address and the second digit is the element address.

In addition to being modified by subsequent Z^CE## commands, the cluster and element addresses may also be changed by the LOAD NORMAL command and the SET CLUSTER DISPLAY MODE command. The LOAD NORMAL command sets the address to X'00' and the SET CLUSTER DISPLAY MODE command causes the element address to be set to a zero following each cluster scan.

Z^CKIL CLEAR SINGLE CLOCK MODE

When this command is entered, the system clocks are restarted immediately. The Z^CKIL command may be issued at any time. If the system is not in the Single Clock Mode, the Z^CKIL command is ignored.

Z^CMM0 CLEAR MM FEATURES

Upon completion of maintenance operations, the CLEAR MM FEATURES command (as well as the SUPER RESET command, described below) may be used to restore the system to a standard configuration status. The CLEAR MM FEATURES command does the following:

1. Restores the system clock to its normal frequency.
2. Clears the "Repeat Clocking Mode" (see Z^CMM1 command).
3. Clears the "Cluster Display Mode" (see Z^CMM2 command).
4. Clears the "P-Mode Repeat Mode" (see Z^CMM3 command).
5. Clears the "Micro-diagnostic Repeat Mode" (see Z^CMM5 command).
6. Clears the "Override Interleave Mode" (see Z^CMM9 command).
7. Clears the "Display Inhibit Mode" (see Z^CMMA command).

Note that the CLEAR MM FEATURES command does not generate any resets (Z^CRIO, Z^CRSY, or Z^CRBP), does not clear P-Mode, nor does it clear Single Clock Mode (see Z^CCLK and Z^CKIL commands).

Z^CMM1 SET/CLEAR REPEAT CLOCKING MODE

This command may be used either to SET REPEAT CLOCKING MODE or to CLEAR REPEAT CLOCKING MODE. When the Repeat Clocking Mode is set, system clocks are repeatedly issued to the system.

If the Display Inhibit Mode, as described below, is also set, the clock rate during Repeat Clocking Mode is approximately 1600 Hertz. If the Display Inhibit Mode is not set (cleared), the clock rate is determined by the communications frequency of the System Control Console.

The amount of information displayed when the Repeat Clocking Mode is set is also dependent upon the Cluster Display Mode. If the Cluster Display Mode (see Z^CMM2 command) is not set, the contents of the selected Single-Clock Status Register is displayed after each clock.

If the Cluster Display Mode is set, the contents of all 16 SCSRs within a selected cluster are displayed after each clock.

The above display routine is interrupted during a Z^C format command. This mode is cleared by a CLEAR REPEAT CLOCK MODE, Z^CMM1, a CLEAR MM FEATURES, Z^CMM0, or a SUPER RESET, Z^CMM4, command.

Z^CMM2 SET/CLEAR CLUSTER DISPLAY MODE

This command may be used either to SET CLUSTER DISPLAY MODE or to CLEAR CLUSTER DISPLAY MODE. When the Cluster Display Mode is set, any console operation which causes the display of a Single-Clock Status Register (e.g., Z^CCLK, Z^CMM1, $Z^CE\#\#$, $Z^CC\#\#$, Z^CKIL, or "$\not{0}$" during Single Clock Mode) will cause the contents of all SCSRs in the selected cluster to be displayed in succession.

If the Cluster Display Mode is set, it may be reset by a CLEAR CLUSTER DISPLAY MODE, Z^CMM2, a CLEAR MM FEATURES, Z^CMM0, or a SUPER RESET, Z^CMM4, command.

Z^CMM3 SET/CLEAR P-MODE REPEAT MODE

This command may be used either to SET P-MODE REPEAT MODE or to CLEAR P-MODE REPEAT MODE.

When the P-Mode Repeat Mode is set, any P-Mode function character (see "Diagnostic Control Commands") entered from the control console is automatically repeated following each line of display. The repetition of P-Mode functions is halted by entering any character while repetition is active. Repetition is automatically resumed when another function is entered.

The P-Mode Repeat Mode may be reset by a CLEAR P-MODE REPEAT MODE, Z^CMM3, a CLEAR MM FEATURES, Z^CMM0, or a SUPER RESET, Z^CMM4, command.

The P-Mode repeat feature is particularly useful for scanning through sequential memory locations or Q registers (using the INCREMENT REFERENCED ADDRESS AND DISPLAY command or INSTRUCTION SINGLE STEP command described under "Diagnostic Control Commands").

Z^CMM4 SUPER RESET

The primary application of the SUPER RESET command is to restore the system to a predetermined condition during and following maintenance activities. The SUPER RESET command is accepted and executed only if the MAINT MODE switch on the SCP is in the ON position. Entering a SUPER RESET command when the system is not in the maintenance mode results in an error message without affecting the system.

If a SUPER RESET command is accepted, all reset signals (System, I/O, and BP) are issued. In addition, the ECS is reset and initialized, and the basic processor executes an initializing routine which clears the contents of the Q scratchpad prior to executing a normal reset.

After a SUPER RESET command is executed, the system remains in the IDLE state, and the ECS is automatically placed in P-Mode.

Z^CMM5 SET MICRO-DIAGNOSTIC LOOP MODE

This command allows maintenance personnel to repetitively loop the micro-diagnostic test of a single system element. The operator must ensure that the system is in the IDLE state prior to entering this command.

This mode may be cleared by either a SUPER RESET, Z^CMM4, or a CLEAR MM FEATURES, Z^CMM0, command.

Z^CMM6 INITIATE ELEMENT MICRO-DIAGNOSTIC

This command causes a single element micro-diagnostic test to be initiated for the selected Single Clock Status element, even if the system is in Single Clock Mode. The operator must ensure that the normal preconditions of micro-diagnostic test execution provided in the LOAD NORMAL sequence are met. This may be accomplished by the following command sequence:

CLEAR SINGLE CLOCK MODE (enables clocks for reset)

SYSTEM HALT (system must be in IDLE state)

RESET SYSTEM (test must be preceded by a system reset)

SELECT/DISPLAY SINGLE CLOCK STATUS REGISTER

SET SINGLE CLOCK MODE

INITIATE ELEMENT MICRO-DIAGNOSTIC

SELECT/DISPLAY SINGLE CLOCK STATUS REGISTER

SINGLE CLOCK STEP (step-through sequence)

Z^CMM7 SET LOW CLOCK MARGINS

This command causes the system clock frequency to be set to low margin. The CLOCK MARGIN indicator (see System Control Panel) is illuminated. If high and low clock margins are both set, an undefined intermediate frequency results.

The system clock is restored to a normal frequency by either a SUPER RESET, Z^CMM4, or a CLEAR MM FEATURES, Z^CMM0, command. The system clock also assumes the normal level following power on.

Z^CMM8 SET HIGH CLOCK MARGINS

This command causes the system clock frequency to be set to high margin. The CLOCK MARGIN indicator (see System Control Panel) is illuminated. If high and low clock margins are both set, an undefined intermediate frequency results.

The system clock is restored to a normal level by either a CLEAR MM FEATURES, Z^CMM0, command or a SUPER RESET, Z^CMM4, command. The system clock also assumes the normal level following power on.

Z^CMM9 SET MEMORY INTERLEAVE OVERRIDE

This command inhibits interleaving all memory units and is used primarily when running certain memory diagnostic programs. It is allowed only when the system is in the maintenance mode. The change in the manner in which memory is accessed when interleaving is inhibited versus when interleaving is permitted requires that programs be reloaded each time the interleave control is changed. Note that the SET MEMORY INTERLEAVE OVERRIDE command inhibits interleaving all memory units, whereas the INTERLEAVE switches of the Configuration Control Panel (described in Chapter 6) inhibit interleaving on an individual memory unit basis. The INTERLEAVE DISABLE indicator on the SCP is illuminated while INTERLEAVE OVERRIDE is in effect.

The SET MEMORY INTERLEAVE OVERRIDE command remains in effect (interleaving is inhibited) until either a CLEAR MM FEATURES, Z^CMM0, or a SUPER RESET, Z^CMM4, command is executed. Interleaving is automatically enabled following a power on/off cycle.

Z^CMMA SET/CLEAR DISPLAY INHIBIT MODE

This command may be used either to SET DISPLAY INHIBIT MODE or to CLEAR DISPLAY INHIBIT MODE. When the Display Inhibit Mode is set, all data output associated with the System Control Console (SCC) function is inhibited; however, data output generated by the software is not affected.

The Display Inhibit Mode is normally set to inhibit printout during execution of SCC functions which do not require a display.

The Display Inhibit Mode may be cleared by a CLEAR DISPLAY INHIBIT MODE, Z^CMMA, a CLEAR MM FEATURES, Z^CMM0, or a SUPER RESET, Z^CMM4, command. The Display Inhibit Mode is automatically cleared following power on.

Z^CLDS#### LOAD SPECIAL

The LOAD SPECIAL command is accepted only if the system is in the IDLE state. The LOAD SPECIAL command is used in situations where all of the functions performed by the LOAD NORMAL command are not desired (e.g., in loading a postmortem dump sequence). The LOAD SPECIAL command causes only the bootstrap loader program to be written into memory without diagnostics or memory clearing prior to the load. A "load device address", specified by the hexadecimal digits "####" in the command, is stored in Q register X'1E'. When using the LOAD SPECIAL command, the operator must also enter the SYSTEM RESET and SYSTEM RUN commands before loading will commence.

Z^CLDT MEMORY SET

This command causes all memory to be set to the value contained in the BP internal register Q26(X'1A'). The command may be entered only when the system is in the IDLE state.

Z^CT SET/CLEAR TRANSPARENT TEXT MODE

This command is used either to SET TRANSPARENT TEXT MODE or to CLEAR TRANSPARENT TEXT MODE. When the Transparent Text Mode is set, software-driven I/O from the System Control Console is inhibited, but all SCC FUNCTIONS are processed in the normal manner. This feature permits the operator to make marginal notes on the console printout for logging purposes. If two different control consoles are connected in parallel (i.e., remote device is connected as a System Control Console), the Transparent Text Mode permits messages to be exchanged between the two devices. If the SCC FUNCTIONS switch is in the DISABLE position, input data is passed into the I/O system regardless of the status of the Transparent Text Mode.

The Transparent Text Mode may also be cleared by a SUPER RESET, Z^CMM4, command.

SYSTEM CONTROL PANEL

The System Control Panel contains indicators and controls which are used primarily when maintenance and/or diagnostic activities are performed. The computer operator normally monitors certain indicators (as described below) to ascertain conditions within the computer system (e.g., status of primary power, status of sense switches, status of BP, and status of ALARM indicators).

The controls and indicators of the System Control Panel (see Figure 15) are functioanlly described below.

POWER ON

This alternate-action switch/indicator controls the application of power to the system. The indicator is illuminated only when the switch has been depressed and power is being applied to the system.

PRIMARY POWER

This indicator is illuminated whenever PRIMARY power is applied to the system.

POWER FAULT

This indicator is illuminated only if an abnormal power system condition exists. Maintenance action is required when the POWER FAULT indicator is lit.

MAINTENANCE MODE

This indicator is illuminated when the computer system is placed in the maintenance mode as the result of the MAINT MODE switch being in the ON position.

INTERLEAVE DISABLE

This indicator is illuminated whenever the two-way interleaving feature of the memory system is inhibited. (See SET MEMORY INTERLEAVE OVERRIDE command, under "Maintenance Control Commands".)

CLOCK MARGIN

This indicator is illuminated whenever the system clock frequency is above or below the normal value (usually as a result of maintenance and/or diagnostic activities; see SET LOW CLOCK MARGINS and SET HIGH CLOCK MARGINS commands, under "Maintenance Control Commands").

POWER MARGIN

This indicator is illuminated whenever any power supply within the computer system has its low margin switch set.

SENSE SWITCH

These four indicators display the status of the four sense switches. Each indicator is appropriately marked (1, 2, 3, 4) and is illuminated only when the corresponding sense switch is on.

ALARM

This indicator is illuminated whenever the system Alarm flip-flog has been set, signifying that a condition has occurred which requires the attention of the operator. This visual alarm may also be augmented with an audio alarm (see ALARM AUDIO, below).

IDLE

This indicator is illuminated whenever the BP operations have been interrupted by the ECS. When the system is in the IDLE state, the BP will fetch and store data or execute instructions only upon request from the System Control Console. The BP states (RUN, WAIT, or HALT) are only defined when the BP is executing instructions.

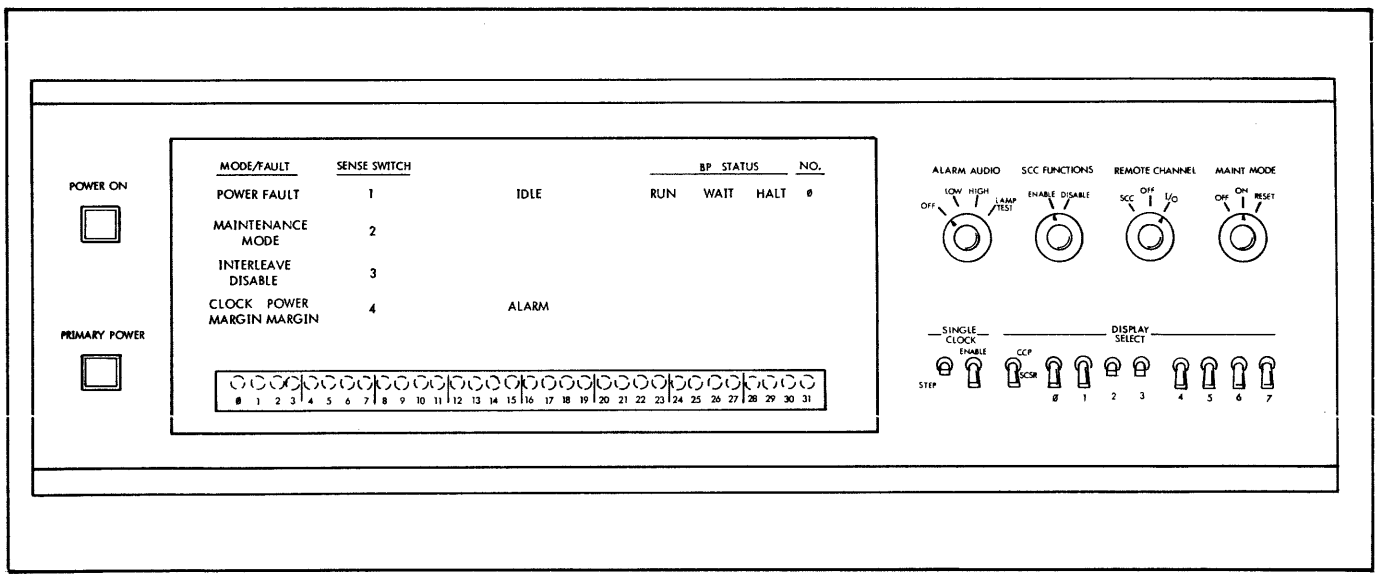


Figure 15. System Control Panel

BP STATUS AND NO.

This group of indicators permits the processor address (usually 0) and current internal state (RUN, WAIT, or HALT) of the BP to be displayed. While executing instructions, the BP is normally in the RUN or WAIT state. The HALT state is entered only when an address halt occurs, the processor disable is on (see "Operating Procedures and Information") or an irrecoverable processor fault occurs. When the system is in the IDLE state as a result of power on, a Z^{C}HLT command, or a p^{C} command, only the processor address is lighted and RUN, WAIT, and HALT indicators are extinguished.

ALARM AUDIO

This 4-position rotary switch controls the connection and volume of a loudspeaker, and also permits all indicators (except POWER ON and PRIMARY POWER) on the SCP to be tested. When this switch is in the OFF position, the loudspeaker is disconnected. Note that this switch does not inhibit the ALARM indicator. When this switch is in the LOW position, the loudspeaker is connected and the volume is set to a low level. When this switch is in the HIGH position, the loudspeaker is connected and the volume is set to a high level. When this switch is held in the LAMP TEST position, all back-lighted indicators should illuminate, simultaneously. The switch returns to the HIGH position when released.

SCC FUNCTIONS

This switch controls the functional capabilities of the System Control Console(s). When this switch is in the ENABLE position, the SCC device(s) may perform the various control functions attributed to a System Control Console. Certain control functions require the system to be in the IDLE state while others (as described under "Maintenance Control Commands") require the MAINT MODE switch to be in the ON position.

When the SCC FUNCTIONS switch is in the DISABLE position, the control functions that may be entered from the control console (to interact with operating software) are limited to the following:

1. Operator requested interrupt (Z^{C}I),
2. Read Sense Switches (Z^{C}SSW),
3. Set Sense Switches (Z^{C}SS#).

The operator may lock out potentially disruptive control commands when the operating software is running by setting the SCC FUNCTIONS switch to DISABLE.

REMOTE CHANNEL

This 3-position rotary switch controls the manner in which the alternate and remote consoles may operate. When this switch is in the SCC position, the alternate and remote consoles may be connected in parallel with the System Control Console and may perform the same control functions as the local control device. The remote console also requires a manual connection through the RD1 data set. Note that any restrictions upon the control functions imposed upon the local control device by the SCC FUNCTIONS switch being in the DISABLE position also apply to both consoles. Either the alternate or remote console is selected for input by the ALTSEL switch on the Configuration Control Panel (see Chapter 6).

When the REMOTE CHANNEL switch is in the OFF position, the remote device is disconnected from the ECS at the data set (if present). The alternate console functions in the I/O mode.

When this switch is in the I/O position, the alternate and remote consoles operate strictly as I/O devices communicating with the computer system via IOP subchannel address X'0B'. Only one device is selected for input at a time by the ALTSEL switch on the Configuration Control Panel (see Chapter 6).

MAINT MODE

During normal operations, this switch is placed in the OFF position. During maintenance and/or diagnostic activies, this switch may be placed in the ON position or momentarily held in the RESET position (switch automatically returns to the ON position when released). In addition to causing the MAINTENANCE MODE indicator to become illuminated when placed in the ON position, the switch also enables certain hardware controls and allows their associated control commands to be entered from the operator's control console (see "Maintenance Control Commands"). This interlocking feature prevents inadvertent adverse effects upon the current program.

Caution should be exercised in activating RESET, since this position (equivalent to the SCC SUPER RESET command) causes all components of the system to be reset and initialized.

SELECT DISPLAY

These nine switches, labeled CCP/SCSR and 0 through 7, are used to specify the binary address of any one of up to 256 Single Clock Status Registers and up to 32 Configuration Status Registers or Read Direct Mode 9 Status Registers whose content is to be displayed by the 32 binary indicators, labeled 0 through 31.

When the CCP/SCSR switch is in the SCSR position, switches 0 through 3 specify the cluster address and switches 4 through 7 specify the element address of the Single Clock Status Register whose content is to be displayed.

When the CCP/SCSR switch is in the CCP position and switch 0 is in the "0" position, switches 3 through 7 specify the binary address of the cabinet whose Read Direct Mode 9 Status Register is to be displayed by the 32 panel indicators.

When the CCP/SCSR switch is in the CCP position and switch 0 is in the "1" position, switches 3 through 7 specify the binary address of the cabinet whose 16-bit Configuration Status Register is to be displayed by the 16 lower-order indicators.

SINGLE CLOCK ENABLE

This switch stops all central system clocks in the same manner as the Z^CCLK command. Activating this switch when the basic processor is performing normal data processing may have an adverse effect on any active I/O operations. To prevent inadvertent activation of this control, it is disabled unless the MAINT MODE switch is in the ON position.

SINGLE CLOCK STEP

This switch is active only when in Single Clock Mode or when the Single Clock Enable switch is active. When active, this switch causes one system clock to be issued each time it is placed in the STEP position. The new single clock status, as selected by the MODE and SELECT switches, may be monitored via the 32 binary indicators on the System Control Panel; no display is generated on the System Control Console by activation of the SCP Single Clock Step switch.

OPERATING PROCEDURES AND INFORMATION

This section contains reference information which may be required by either the operator or maintenance/diagnostic personnel.

LOAD OPERATION DETAILS

The first executed instruction of the bootstrap program (in location X'26') loads general register 0 with the address of the first I/O command doubleword (IOCD). The I/O address for the SIO instruction in location X'27' is the 13 low-order bits of location X'25' (which have been set equal to the load unit address as a result of the NORMAL LOAD, Z^CLDN####, command). During execution of the SIO instruction, general register 0 points to locations X'20' and X'21' as the first IOCD for the selected device. This IOCD contains an order to the selected peripheral device to read 88 (X'58') bytes of data into consecutive memory locations beginning at word location X'2A' (byte location X'A8'). At the end of the Read operation, neither data chaining nor command chaining is called for in the IOCD. The Suppress Incorrect Length (SIL) flag is set to 1 so that an incorrect length indication will not cause a Transmission Error Halt. After the SIO instruction has been executed, the basic processor executes a TIO instruction with the same effective address as the SIO instruction. The TIO instruction is coded to accept only condition code data from the IOP. The BCS instruction (in location X'29') will cause a branch to X'22' (a LOAD IMMEDIATE instruction), if either CC1 or CC2 is set to 1. Execution of the LOAD IMMEDIATE instruction at X'22' loads a count of X'10029' into general register 1. The following BDR instruction at location X'23' uses this as a "delay" count before executing the BCR instruction in location X'24', which unconditionally branches to the TIO instruction in location X'28'. In normal operations, CC1 is reset to 0 and CC2 remains set to 1 until the device can accept another SIO instruction. At that time, the next instruction is taken from location X'2A'.

If a Transmission Error or equipment malfunction is detected by either the device or the IOP, the IOP instructs the device to halt and to initiate an "unusual end" interrupt signal (as specified by appropriate flags in the IOCD, described in Chapter 4). The "unusual end" interrupt will be ignored since all interrupt levels have been Disarmed and Disabled by the system reset during the load sequence. The device will not accept another SIO while the interrupt is pending and the BCS instruction in location X'29' will continue to branch to location X'22'. The correct operator action at this point is to repeat the NORMAL LOAD, Z^CLDN####, command. If there is no I/O address recognition of the load unit, the SIO instruction will not cause any I/O action and CC1 will continue to be set to 1 by the SIO and TIO instructions causing the BCS instruction to branch.

FETCHING and STORING DATA

The following examples illustrate how diagnostic control (P-Mode) commands may be used to display and alter the contents of specified memory locations and control registers within the system. Control commands, as entered from a keyboard device functioning as the System Control Console, are shown in the first column. The resulting printouts are shown in the second column. The third column of information is an explanation of the functions performed by the different control commands.

Input	Printout	Function
P^C	0:DDDDDDDD @ 80000000	Enter P-Mode of operations; contents of Q register 0 is normally displayed.
100/	100/ 0:DDDDDDDD @ 00000100	Select and display contents of memory location X'100'.

Input	Printout	Function
5M	5M 0:00000005 @ 00000100	Store X'5' into the currently selected memory location.
I	I 0:DDDDDDDD @ 00000101	Increment address of currently selected memory location and display.

Note that all P-Mode accesses are qualified by address mapping bits and Write Lock keys in the Program Status Words.

PROCESSOR CONTROL WORD

The Processor Control Word resides in the processor internal addressable register, Q30. This register may be loaded with appropriate control information to perform maintenance or diagnostic functions, such as halting and resetting the basic processor, setting address hold, and activating various fault detection controls. During normal operations it should not be necessary to access this word. The contents of the Processor Control Word are not affected by either processor or system reset, but are automatically set to zero (default condition) during power-on sequencing and by the SUPER RESET command. The bit assignments of the Processor Control Word (register Q30) are listed and described in Table 23.

ADDRESS COMPARE WORD

The Address Compare Word is located in register Q31 and contains parameters defining the type of comparison and the desired action (alarm, halt, or none) on detecting an address compare. (See Table 24.)

Table 23. Bit Assignments and Description, Processor Control Word, Register Q30 (X'1E')

Bit Position	Description
0	Retry Inhibit: If this bit is a 0, the basic processor will automatically retry the instruction which caused the trap to location X'4C'; if this bit is a 1, the basic processor is inhibited from retrying the instruction which caused the trap to location X'4C'.
1	Parity Check Inhibit: If this bit is a 0, parity checking of R register transactions is enabled; if this bit is a 1, parity checking of R register transactions is inhibited.
2	Watchdog Timer Override: If this bit is a 0, the watchdog timer is allowed to count; if this bit is a 1, the watchdog timer is inhibited from counting and the machine will not execute the Watchdog Timer Trap.
3	Watchdog Timer Alarm: If this bit is a 0, the Watchdog Timer Trap is enabled; if this bit is a 1, the Watchdog Timer Trap is inhibited. When a timeout occurs, a system reset is generated and the system will run to timeout again. This provides a dynamic loop for isolating the cause of the timeout.
4-5	Reserved (must be coded as zeros).
6	Address Hold: If this bit is a 0, the address hold is disabled; if this bit is a 1, the program counter is inhibited from counting (incrementing) causing the machine to loop on the selected instruction (i.e., when the machine is returned to RUN mode, the instruction pointed to by the program counter is executed continuously).
7	Processor Halt: If this bit is a 0, the processor is allowed to run under the control of system and P-Mode controls; If this bit is a 1, the processor is forced into the HALT condition.
8-15	Reserved.
16-31	Load device address.

Table 24. Bit Assignments, Address Compare Register Q31 (X'1F')

Bit Position	Status	Significance
0	1	Selects mapped address comparison.
	0	Selects unmapped address comparison.
1	1	Selects address comparison during instruction access only.
	0	Selects address comparison for all memory cycles.
2	1	Selects comparisons only during memory write cycle.
	0	Selects all memory cycle comparisons.
3	1	Selects page comparisons.
	0	Selects word comparisons.
4	1	System turns on audible alarm for 220 microseconds each time an Address Compare occurs (maximum frequency 1KHz).
	0	Address match alarm is disabled.
5	1	The processor is forced into the HALT state when an Address Compare occurs.
	0	Address Halt disabled.
6-7	-	Reserved.
8-31	-	Comparison address field.

6. SYSTEM CONFIGURATION CONTROL

Pooled resources along with flexible configuration control provide a high degree of continuous availability. If a problem occurs in an individual unit of a resource pool, the system may allow that unit to be isolated with a loss only in capacity but no loss of functional capability, assuming there is an additional unit of that type in the system that can absorb the added load. Specialized units can be duplicated (with all units being normally used, where possible) and configuration controls used to divert the input from one to the other in the event of a failure.

Chapter 2 describes the system organization and Chapter 5 discusses system operational control. This chapter describes the Configuration Control Panel (CCP), which serves as the principal element for controlling and modifying the configuration of the system.

CONFIGURATION CONTROL PANEL (CCP)

The CCP provides the capability for enabling and disabling units in the system. It accomplishes this with centrally located manual selection switches used for the following functions:

1. Establish starting addresses for all memory units in the system.
2. Enable or disable memory ports.
3. Enable or disable individual units and clusters.
4. Control the power throughout the system.

The Configuration Control Panel is mounted within the endbell assembly at the end of the row of cabinets containing the chassis assemblies for the MUs, BP, and other system components. On the outer surface of the endbell assembly is the System Control Panel (described and illustrated in Chapter 5). Access is gained to the CCP by opening the hinged endbell assembly (see Figure 16).

A CCP has six rows of 22 toggle switches and two lamp indicators each. A row may control a memory unit, processor cluster, or system control processor. (See Figure 17, and Tables 25 and 26.) The active logic associated with each row of switches and indicators is located within each processor cluster or memory unit itself. Above each row is a marker strip that identifies the function of each switch. The configuration control is designed in a modular manner. As the system grows, previously unused rows on the panel can be used (up to the panel's maximum of six), and an additional panel may be added. Two panels represent the maximum configuration for one endbell assembly.

Note: The Configuration Control Panel does not contain operational controls as the System Control Panel does. The CCP switches are initially positioned during system configuration and are not normally repositioned during system operation.

CONFIGURATION CONTROL STATUS WORD

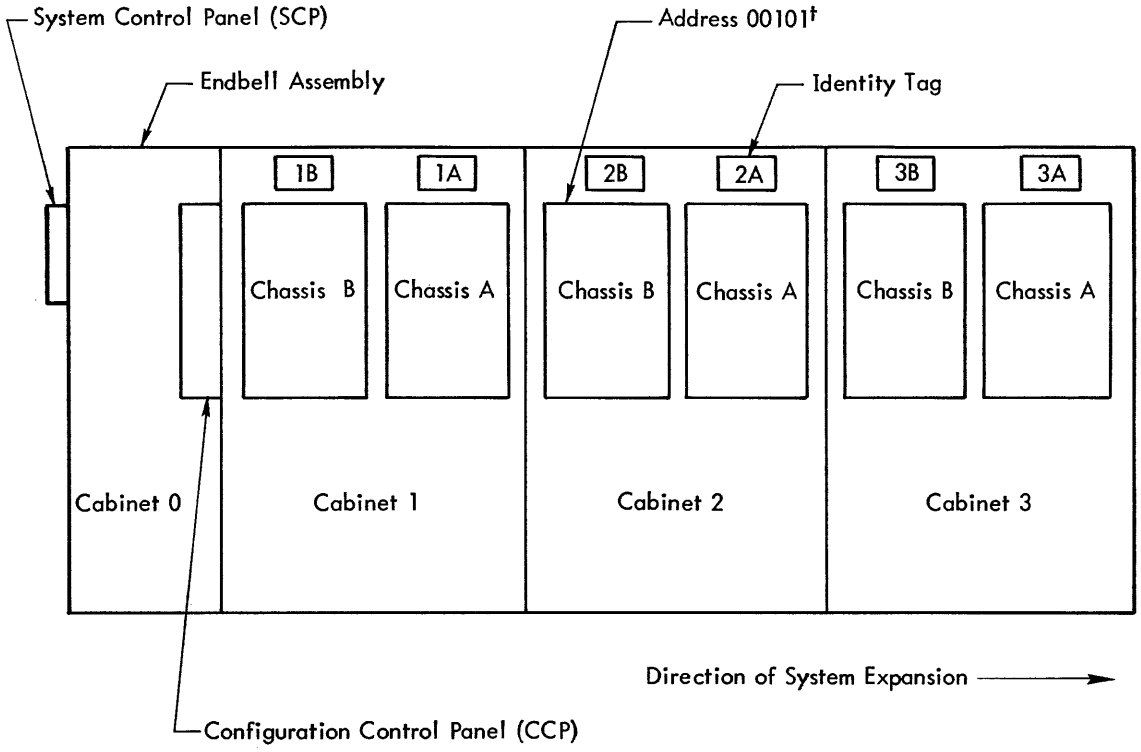
A program may read settings of the panel switches, type of unit, and options installed. A READ DIRECT (RD) instruction using the chassis address of the cluster or unit as an address allows the program to determine the configuration status of a particular processor cluster or memory unit, for example. (The chassis address assignment represents the chassis' physical location relative to the endbell assembly.) (See Figure 16.)

The configuration control status of a panel read by the RD instruction is a 32-bit status word consisting of panel switch settings and type information. (The RD instruction is described in Chapter 3, "Control Instructions".) The logic for these program provisions resides in each unit.

In addition to reading configuration status information via a READ DIRECT (RD) instruction in a program, the status information may also be obtained by manual switch selection on the System Control Panel; the 32-bit status word is displayed on a bank of lamp indicators. (See Chapter 5 for a discussion of the System Control Panel features.)

CONFIGURATION BUS

The configuration bus connects to each processor cluster and provides a path for the system control processor to select and read the switch settings on the CCP for the selected unit via an RD instruction.



†Starting from the endbell as cabinet number 0, the most significant four bits designate physical cabinet number. The least significant bit designates the back panel location in the halves of the cabinet.

Figure 16. Chassis Physical Configuration

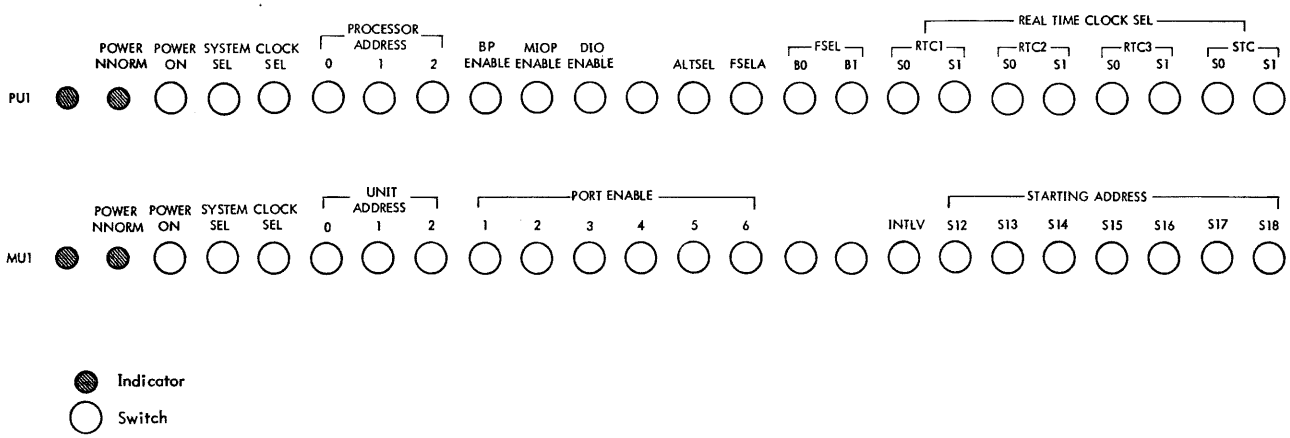


Figure 17. Sample Rows of CCP Switches

Table 25. Functions of Processor Cluster Configuration Control Panel Row

Label	Switch/Indicator	Function
POWER NNORM	1 indicator	Lighted when unit power is shut down due to abnormal operational condition.
POWER ON	1 switch	When in up or middle position, enables power-on control in the unit power supply. (Middle position inhibits the unit reset signal.) When in down position, power to unit is off.
SYSTEM SEL	1 switch	Selects the processor bus to which the processor cluster is to be connected (up is processor bus A, down is B).
CLOCK SEL	1 switch	Selects the clock source (up, A or down, B) for the unit clock subsystem.
PROCESSOR ADDRESS	3 switches	Establishes the logical address of the cluster within a group of processor clusters. Note: The 5-bit chassis location number and not the processor address is used in addressing the configuration switches for a given unit by the RD instruction directed to the Configuration Control Panel.
BP ENABLE	1 switch	When in down position, inhibits the BP from operating on the internal bus.
MIOP ENABLE	1 switch	When in down position, inhibits the MIOP from operating on the internal bus.
DIO ENABLE	1 switch	When in down position, inhibits external DIO interface.
ALTSEL	1 switch	Selects either the remote console (down position) or alternate operator's console (up position) to enter data on the Remote Channel Interface.
FSELA	1 switch	Selects communications frequency for the primary operator's console as follows: up = same frequency as remote channel down = 1200 baud Note: The 1200 baud selection is effective only if the REMOTE CHANNEL switch on the System Control Panel is not in the SCC position.
FSELB0/FSELB1	2 switches	Selects communications frequency for the alternate operator's console and the Remote Diagnostic Interface (Remote Channel) as follows: B0 B1 Rate (baud) 0 0 110 0 1 1200 1 0 unspecified 1 1 300

Table 25. Functions of Processor Cluster Configuration Control Panel Row (cont.)

Label	Switch/Indicator	Function
REAL-TIME CLOCK SEL	8 switches	Four groups (labeled RTC1, RTC2, RTC3, and STC) of 2 switches each (labeled S0 and S1), used for selecting the real-time clock frequencies; each of the three real-time clock counters and the one subjective clock counter may have their frequencies selected by the proper combination of the two switches associated with each counter: S0 S1 Frequency (Hz) 0 0 500 0 1 External real-time clock source 1 0 2000 1 1 Power line

Table 26. Functions of Memory Unit Configuration Control Panel Row

Label	Switch/Indicator	Function
POWER NNORM	1 indicator	Lighted when unit power is shut down due to abnormal operational condition.
POWER ON	1 switch	When in up or middle position, enables power-on control in the unit power supply. (Middle position inhibits the unit reset signal.) When in down position, power to unit is off.
SYSTEM SEL	1 switch	Determines to which central shared resources the reset signal is connected.
CLOCK SEL	1 switch	Selects which clock the memory shall use; up position selects system clock A, down position selects system clock B.
UNIT ADDRESS	3 switches	Establishes the logical address of the unit within a group of memory units.
PORT ENABLE	6 switches	Down position disables, up enables, corresponding port when setting up different configuration in the system. Switch 1 (leftmost) corresponds to port 1, etc., and switch 6 corresponds to port 6.
INTLV	1 switch	Up position selects interleave addressing mode in each memory unit; down position means no interleaving in any memory unit. Only two-way interleaving is allowed. The unit interleaved with this memory unit must have its interleave switch on and be in the appropriate addressing range. The interleave logic operates only for memory units with a number corresponding to a power of 2: i.e., 16K, 32K words; if other than a power of 2, the interleave signal it receives is ignored. Interleaving is permissible only: 1. Between two memory units of the same size. 2. Provided the two memory units cover an addressing range that is continuous and starts at a multiple of the size of the two memory units taken together.

Table 26. Functions of Memory Unit Configuration Control Panel Row (cont.)

Label	Switch/Indicator	Function
STARTING ADDRESS	7 switches	Used to set the starting addresses of the memory units. From left to right the switches are labeled S12, S13, S14, S15, S16, S17, and S18. Using the switches as a 7-position binary field, this allows memory to address up to 1 million words. When the memory system comprises memory units of different sizes, some precautions are necessary to prevent false address recognition as well as to prevent gaps in the memory range. 1. If all memory units have sizes that are powers of two, they can all be different; they must, however, be assigned in order of decreasing size in the address continuum. For instance, three memory units can be used in this manner: Memory Unit No. / Size / Address Range 1 / 64K words / 0 to 64K words 2 / 32K words / 64 to 96K words 3 / 16K words / 96 to 112K words 2. If a memory unit has a size that is not a power of two, it must be situated in a memory system that satisfies the following rules: a. All other memory units must have sizes that are a power of two. b. The starting address of the non-power-of-two unit must be a multiple of the next power of two above the size of that unit. c. The memory unit whose size is not a power of two must be at the upper end of the address range.

APPENDIX A. REFERENCE TABLES

This appendix contains the following reference material:

Title

Standard Symbols and Codes

Standard 8-Bit Computer Codes (EBCDIC)

Standard 7-Bit Communication Codes (ANSCII)

Standard Symbol-Code Correspondences

Hexadecimal Arithmetic

- Addition Table
- Multiplication Table
- Table of Powers of Sixteen$_{10}$
- Table of Powers of Ten$_{16}$

Hexadecimal-Decimal Integer Conversion Table

Hexadecimal-Decimal Fraction Conversion Table

Table of Powers of Two

Mathematical Constants

STANDARD SYMBOLS AND CODES

The symbol and code standards described in this publication are applicable to all Xerox computer products, both hardware and software. They may be expanded or altered from time to time to meet changing requirements.

The symbols listed here include two types: graphic symbols and control characters. Graphic symbols are displayable and printable; control characters are not. Hybrids are SP, the symbol for a blank space; and DEL, the delete code, which is not considered a control command.

Three types of code are shown: (1) the 8-bit Xerox Standard Computer Code, i.e., the Extended Binary-Coded-Decimal Interchange Code (EBCDIC); (2) the 7-bit American National Standard Code for Information Interchange (ANSCII); and (3) the Xerox standard card code.

STANDARD CHARACTER SETS

1. EBCDIC

   57-character set: uppercase letters, numerals, space, and & - / . < > ( ) + | $ * : ; , % # @ ' =

   63-character set: same as above plus ¢ ! _ ? " ¬

   89-character set: same as 63-character set plus lowercase letters

2. ANSCII

   64-character set: uppercase letters, numerals, space, and ! " $ % & ' ( ) * + , - . / \ ; : = < > ? @ _ [ ] ^ #

   95-character set: same as above plus lowercase letters and { } ¦ ~ `

CONTROL CODES

In addition to the standard character sets listed above, the symbol repertoire includes 37 control codes and the hybrid code DEL (hybrid code SP is considered part of all character sets). These are listed in the table titled Standard Symbol-Code Correspondences.

SPECIAL CODE PROPERTIES

The following two properties of all standard codes will be retained for future standard code extensions:

1. All control codes, and only the control codes, have their two high-order bits equal to "00". DEL is not considered a control code.

2. No two graphic EBCDIC codes have their seven low-order bits equal.

STANDARD 8-BIT COMPUTER CODES (EBCDIC)

Least Significant Digits	Hexadecimal	Binary	Most Significant Digits 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
			0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0	0000	NUL	DLE	ds		SP	&	-									0
1	1	0001	SOH	DC1	ss				/		a	j		\ [1]	A	J		1
2	2	0010	STX	DC2	fs						b	k	s	{ [1]	B	K	S	2
3	3	0011	ETX	DC3	si						c	l	t	} [1]	C	L	T	3
4	4	0100	EOT	DC4							d	m	u	[[1]	D	M	U	4
5	5	0101	HT	LF NL			Will not be assigned				e	n	v	] [1]	E	N	V	5
6	6	0110	ACK	SYN							f	o	w		F	O	W	6
7	7	0111	BEL	ETB							g	p	x		G	P	X	7
8	8	1000	EOM BS	CAN							h	q	y		H	Q	Y	8
9	9	1001	ENQ	EM							i	r	z		I	R	Z	9
A	A	1010	NAK	SUB			¢ [2]	!	^ [1]	:								
B	B	1011	VT	ESC			.	$	,	#								
C	C	1100	FF	FS			<	*	%	@					Will not be assigned			
D	D	1101	CR	GS			(	)	_	'								
E	E	1110	SO	RS			+	;	>	=								
F	F	1111	SI	US			| [2]	¬ [2]	?	"								DEL
			3				4				5							

NOTES:

1 The characters ^ \ { } [ ] are ANSCII characters that do not appear in any of the EBCDIC-based character sets, though they are shown in the EBCDIC table.

2 The characters ¢ | ¬ appear in the 63- and 89-character EBCDIC sets but not in either of the ANSCII-based sets. However, Xerox software translates the characters into ANSCII characters as follows:

EBCDIC	=	ANSCII
¢		` (6-0)
|		¦ (7-12)
¬		~ (7-14)

3 The EBCDIC control codes in columns 0 and 1 and their binary representation are exactly the same as those in the ANSCII table, except for two interchanges: LF/NL with NAK, and HT with ENQ.

4 Characters enclosed in heavy lines are included only in the standard 63- and 89-character EBCDIC sets.

5 These characters are included only in the standard 89-character EBCDIC set.

STANDARD 7-BIT COMMUNICATION CODES (ANSCII) [1]

Least Significant Digits	Decimal (rows) (col's.) →	Binary [1]	Most Significant Digits 0	1	2	3	4	5	6	7
	↓		x000	x001	x010	x011	x100	x101	x110	x111
0	0	0000	NUL	DLE	SP	0	@	P	`	p
1	1	0001	SOH	DC1	! [5]	1	A	Q	a	q
2	2	0010	STX	DC2	"	2	B	R	b	r
3	3	0011	ETX	DC3	#	3	C	S	c	s
4	4	0100	EOT	DC4	$	4	D	T	d	t
5	5	0101	ENQ	NAK	%	5	E	U	e	u
6	6	0110	ACK	SYN	&	6	F	V	f	v
7	7	0111	BEL	ETB	'	7	G	W	g	w
8	8	1000	BS	CAN	(	8	H	X	h	x
9	9	1001	HT	EM	)	9	I	Y	i	y
10	A	1010	LF NL	SUB	*	:	J	Z	j	z
11	B	1011	VT	ESC	+	;	K	[	k	{
12	C	1100	FF	FS	,	<	L	\	l	|
13	D	1101	CR	GS	-	=	M	]	m	}
14	E	1110	SO	RS	.	>	N	^ [4]	n	~ [4]
15	F	1111	SI	US	/	?	O	_ [4]	o	DEL
			2		3					

NOTES:

1 Most significant bit, added for 8-bit format, is either 0 or even parity.

2 Columns 0-1 are control codes.

3 Columns 2-5 correspond to the 64-character ANSCII set. Columns 2-7 correspond to the 95-character ANSCII set.

4 On many current teletypes, the symbol

^ is ↑ (5-14)

_ is ← (5-15)

~ is ESC or ALTMODE control (7-14)

and none of the symbols appearing in columns 6-7 are provided. Except for the three symbol differences noted above, therefore, such teletypes provide all the characters in the 64-character ANSCII set. (The Xerox 7015 Remote Keyboard Printer provides the 64-character ANSCII set also, but prints ^ as ∧.)

STANDARD SYMBOL-CODE CORRESPONDENCES

EBCDIC† Hex.	EBCDIC† Dec.	Symbol	Card Code	ANSCII††	Meaning	Remarks
00	0	NUL	12-0-9-8-1	0-0	null	00 through 23 and 2F are control codes.
01	1	SOH	12-9-1	0-1	start of header	
02	2	STX	12-9-2	0-2	start of text	
03	3	ETX	12-9-3	0-3	end of text	
04	4	EOT	12-9-4	0-4	end of transmission	
05	5	HT	12-9-5	0-9	horizontal tab	
06	6	ACK	12-9-6	0-6	acknowledge (positive)	
07	7	BEL	12-9-7	0-7	bell	
08	8	BS or EOM	12-9-8	0-8	backspace or end of message	
09	9	ENQ	12-9-8-1	0-5	enquiry	
0A	10	NAK	12-9-8-2	1-5	negative acknowledge	
0B	11	VT	12-9-8-3	0-11	vertical tab	
0C	12	FF	12-9-8-4	0-12	form feed	
0D	13	CR	12-9-8-5	0-13	carriage return	
0E	14	SO	12-9-8-6	0-14	shift out	
0F	15	SI	12-9-8-7	0-15	shift in	
10	16	DLE	12-11-9-8-1	1-0	data link escape	
11	17	DC1	11-9-1	1-1	device control 1	
12	18	DC2	11-9-2	1-2	device control 2	
13	19	DC3	11-9-3	1-3	device control 3	
14	20	DC4	11-9-4	1-4	device control 4	
15	21	LF or NL	11-9-5	0-10	line feed or new line	
16	22	SYN	11-9-6	1-6	sync	
17	23	ETB	11-9-7	1-7	end of transmission block	
18	24	CAN	11-9-8	1-8	cancel	
19	25	EM	11-9-8-1	1-9	end of medium	
1A	26	SUB	11-9-8-2	1-10	substitute	Replaces characters with parity error.
1B	27	ESC	11-9-8-3	1-11	escape	
1C	28	FS	11-9-8-4	1-12	file separator	
1D	29	GS	11-9-8-5	1-13	group separator	
1E	30	RS	11-9-8-6	1-14	record separator	
1F	31	US	11-9-8-7	1-15	unit separator	
20	32	ds	11-0-9-8-1		digit selector	20 through 23 are used with EDIT BYTE STRING (EBS) instruction — not input/output control codes. 24 through 2E are unassigned.
21	33	ss	0-9-1		significance start	
22	34	fs	0-9-2		field separation	
23	35	si	0-9-3		immediate significance start	
24	36		0-9-4			
25	37		0-9-5			
26	38		0-9-6			
27	39		0-9-7			
28	40		0-9-8			
29	41		0-9-8-1			
2A	42		0-9-8-2			
2B	43		0-9-8-3			
2C	44		0-9-8-4			
2D	45		0-9-8-5			
2E	46		0-9-8-6			
2F	47		0-9-8-7			
30	48		12-11-0-9-8-1			30 through 3F are unassigned.
31	49		9-1			
32	50		9-2			
33	51		9-3			
34	52		9-4			
35	53		9-5			
36	54		9-6			
37	55		9-7			
38	56		9-8			
39	57		9-8-1			
3A	58		9-8-2			
3B	59		9-8-3			
3C	60		9-8-4			
3D	61		9-8-5			
3E	62		9-8-6			
3F	63		9-8-7			

†Hexadecimal and decimal notation.

††Decimal notation (column-row).

STANDARD SYMBOL-CODE CORRESPONDENCES (cont.)

EBCDIC† Hex.	EBCDIC† Dec.	Symbol	Card Code	ANSCII††	Meaning	Remarks
40	64	SP	blank	2-0	blank	
41	65		12-0-9-1			41 through 49 will not be assigned.
42	66		12-0-9-2			
43	67		12-0-9-3			
44	68		12-0-9-4			
45	69		12-0-9-5			
46	70		12-0-9-6			
47	71		12-0-9-7			
48	72		12-0-9-8			
49	73		12-8-1			
4A	74	¢ or ‘	12-8-2	6-0	cent or accent grave	Accent grave used for left single quote.
4B	75	.	12-8-3	2-14	period	
4C	76	<	12-8-4	3-12	less than	
4D	77	(	12-8-5	2-8	left parenthesis	
4E	78	+	12-8-6	2-11	plus	
4F	79	| or ¦	12-8-7	7-12	vertical bar or broken bar	
50	80	&	12	2-6	ampersand	
51	81		12-11-9-1			51 through 59 will not be assigned.
52	82		12-11-9-2			
53	83		12-11-9-3			
54	84		12-11-9-4			
55	85		12-11-9-5			
56	86		12-11-9-6			
57	87		12-11-9-7			
58	88		12-11-9-8			
59	89		11-8-1			
5A	90	!	11-8-2	2-1	exclamation point	
5B	91	$	11-8-3	2-4	dollars	
5C	92	*	11-8-4	2-10	asterisk	
5D	93	)	11-8-5	2-9	right parenthesis	
5E	94	;	11-8-6	3-11	semicolon	
5F	95	~ or ¬	11-8-7	7-14	tilde or logical not	
60	96	-	11	2-13	minus, dash, hyphen	
61	97	/	0-1	2-15	slash	
62	98		11-0-9-2			62 through 69 will not be assigned.
63	99		11-0-9-3			
64	100		11-0-9-4			
65	101		11-0-9-5			
66	102		11-0-9-6			
67	103		11-0-9-7			
68	104		11-0-9-8			
69	105		0-8-1			
6A	106	^	12-11	5-14	circumflex	On Model 7015 ^ is ∧ (caret).
6B	107	,	0-8-3	2-12	comma	
6C	108	%	0-8-4	2-5	percent	
6D	109	_	0-8-5	5-15	underline	Underline is sometimes called "break character"; may be printed along bottom of character line.
6E	110	>	0-8-6	3-14	greater than	
6F	111	?	0-8-7	3-15	question mark	
70	112		12-11-0			70 through 79 will not be assigned.
71	113		12-11-0-9-1			
72	114		12-11-0-9-2			
73	115		12-11-0-9-3			
74	116		12-11-0-9-4			
75	117		12-11-0-9-5			
76	118		12-11-0-9-6			
77	119		12-11-0-9-7			
78	120		12-11-0-9-8			
79	121		8-1			
7A	122	:	8-2	3-10	colon	
7B	123	#	8-3	2-3	number	
7C	124	@	8-4	4-0	at	
7D	125	'	8-5	2-7	apostrophe (right single quote)	
7E	126	=	8-6	3-13	equals	
7F	127	"	8-7	2-2	quotation mark	

†Hexadecimal and decimal notation.

††Decimal notation (column-row).

STANDARD SYMBOL-CODE CORRESPONDENCES (cont.)

EBCDIC† Hex.	EBCDIC† Dec.	Symbol	Card Code	ANSCII††	Meaning	Remarks
80	128		12-0-8-1			80 is unassigned.
81	129	a	12-0-1	6-1		81-89, 91-99, A2-A9 comprise the lowercase alphabet. Available only in standard 89- and 95-character sets.
82	130	b	12-0-2	6-2		
83	131	c	12-0-3	6-3		
84	132	d	12-0-4	6-4		
85	133	e	12-0-5	6-5		
86	134	f	12-0-6	6-6		
87	135	g	12-0-7	6-7		
88	136	h	12-0-8	6-8		
89	137	i	12-0-9	6-9		
8A	138		12-0-8-2			8A through 90 are unassigned.
8B	139		12-0-8-3			
8C	140		12-0-8-4			
8D	141		12-0-8-5			
8E	142		12-0-8-6			
8F	143		12-0-8-7			
90	144		12-11-8-1			
91	145	j	12-11-1	6-10		
92	146	k	12-11-2	6-11		
93	147	l	12-11-3	6-12		
94	148	m	12-11-4	6-13		
95	149	n	12-11-5	6-14		
96	150	o	12-11-6	6-15		
97	151	p	12-11-7	7-0		
98	152	q	12-11-8	7-1		
99	153	r	12-11-9	7-2		
9A	154		12-11-8-2			9A through A1 are unassigned.
9B	155		12-11-8-3			
9C	156		12-11-8-4			
9D	157		12-11-8-5			
9E	158		12-11-8-6			
9F	159		12-11-8-7			
A0	160		11-0-8-1			
A1	161		11-0-1			
A2	162	s	11-0-2	7-3		
A3	163	t	11-0-3	7-4		
A4	164	u	11-0-4	7-5		
A5	165	v	11-0-5	7-6		
A6	166	w	11-0-6	7-7		
A7	167	x	11-0-7	7-8		
A8	168	y	11-0-8	7-9		
A9	169	z	11-0-9	7-10		
AA	170		11-0-8-2			AA through B0 are unassigned.
AB	171		11-0-8-3			
AC	172		11-0-8-4			
AD	173		11-0-8-5			
AE	174		11-0-8-6			
AF	175		11-0-8-7			
B0	176		12-11-0-8-1			
B1	177	\	12-11-0-1	5-12	backslash	
B2	178	{	12-11-0-2	7-11	left brace	
B3	179	}	12-11-0-3	7-13	right brace	
B4	180	[	12-11-0-4	5-11	left bracket	
B5	181	]	12-11-0-5	5-13	right bracket	
B6	182		12-11-0-6			B6 through BF are unassigned.
B7	183		12-11-0-7			
B8	184		12-11-0-8			
B9	185		12-11-0-9			
BA	186		12-11-0-8-2			
BB	187		12-11-0-8-3			
BC	188		12-11-0-8-4			
BD	189		12-11-0-8-5			
BE	190		12-11-0-8-6			
BF	191		12-11-0-8-7			

†Hexadecimal and decimal notation.

††Decimal notation (column-row).

STANDARD SYMBOL-CODE CORRESPONDENCES (cont.)

EBCDIC† Hex.	EBCDIC† Dec.	Symbol	Card Code	ANSCII††	Meaning	Remarks
C0	192		12-0			C0 is unassigned.
C1	193	A	12-1	4-1		C1-C9, D1-D9, E2-E9 comprise the uppercase alphabet.
C2	194	B	12-2	4-2		
C3	195	C	12-3	4-3		
C4	196	D	12-4	4-4		
C5	197	E	12-5	4-5		
C6	198	F	12-6	4-6		
C7	199	G	12-7	4-7		
C8	200	H	12-8	4-8		
C9	201	I	12-9	4-9		
CA	202		12-0-9-8-2			CA through CF will not be assigned.
CB	203		12-0-9-8-3			
CC	204		12-0-9-8-4			
CD	205		12-0-9-8-5			
CE	206		12-0-9-8-6			
CF	207		12-0-9-8-7			
D0	208		11-0			D0 is unassigned.
D1	209	J	11-1	4-10		
D2	210	K	11-2	4-11		
D3	211	L	11-3	4-12		
D4	212	M	11-4	4-13		
D5	213	N	11-5	4-14		
D6	214	O	11-6	4-15		
D7	215	P	11-7	5-0		
D8	216	Q	11-8	5-1		
D9	217	R	11-9	5-2		
DA	218		12-11-9-8-2			DA through DF will not be assigned.
DB	219		12-11-9-8-3			
DC	220		12-11-9-8-4			
DD	221		12-11-9-8-5			
DE	222		12-11-9-8-6			
DF	223		12-11-9-8-7			
E0	224		0-8-2			E0, E1 are unassigned.
E1	225		11-0-9-1			
E2	226	S	0-2	5-3		
E3	227	T	0-3	5-4		
E4	228	U	0-4	5-5		
E5	229	V	0-5	5-6		
E6	230	W	0-6	5-7		
E7	231	X	0-7	5-8		
E8	232	Y	0-8	5-9		
E9	233	Z	0-9	5-10		
EA	234		11-0-9-8-2			EA through EF will not be assigned.
EB	235		11-0-9-8-3			
EC	236		11-0-9-8-4			
ED	237		11-0-9-8-5			
EE	238		11-0-9-8-6			
EF	239		11-0-9-8-7			
F0	240	0	0	3-0		
F1	241	1	1	3-1		
F2	242	2	2	3-2		
F3	243	3	3	3-3		
F4	244	4	4	3-4		
F5	245	5	5	3-5		
F6	246	6	6	3-6		
F7	247	7	7	3-7		
F8	248	8	8	3-8		
F9	249	9	9	3-9		
FA	250		12-11-0-9-8-2			FA through FE will not be assigned.
FB	251		12-11-0-9-8-3			
FC	252		12-11-0-9-8-4			
FD	253		12-11-0-9-8-5			
FE	254		12-11-0-9-8-6			
FF	255	DEL	12-11-0-9-8-7		delete	Special – neither graphic nor control symbol.

†Hexadecimal and decimal notation.

††Decimal notation (column-row).

HEXADECIMAL ARITHMETIC

ADDITION TABLE

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

TABLE OF POWERS OF SIXTEEN $_{10}$

16^n	n	16^{-n}
1	0	0.10000 00000 00000 00000 × 10
16	1	0.62500 00000 00000 00000 × 10^{-1}
256	2	0.39062 50000 00000 00000 × 10^{-2}
4 096	3	0.24414 06250 00000 00000 × 10^{-3}
65 536	4	0.15258 78906 25000 00000 × 10^{-4}
1 048 576	5	0.95367 43164 06250 00000 × 10^{-6}
16 777 216	6	0.59604 64477 53906 25000 × 10^{-7}
268 435 456	7	0.37252 90298 46191 40625 × 10^{-8}
4 294 967 296	8	0.23283 06436 53869 62891 × 10^{-9}
68 719 476 736	9	0.14551 91522 83668 51807 × 10^{-10}
1 099 511 627 776	10	0.90949 47017 72928 23792 × 10^{-12}
17 592 186 044 416	11	0.56843 41886 08080 14870 × 10^{-13}
281 474 976 710 656	12	0.35527 13678 80050 09294 × 10^{-14}
4 503 599 627 370 496	13	0.22204 46049 25031 30808 × 10^{-15}
72 057 594 037 927 936	14	0.13877 78780 78144 56755 × 10^{-16}
1 152 921 504 606 846 976	15	0.86736 17379 88403 54721 × 10^{-18}

TABLE OF POWERS OF TEN $_{16}$

10^n	n	10^{-n}
1	0	1.0000 0000 0000 0000
A	1	0.1999 9999 9999 999A
64	2	0.28F5 C28F 5C28 F5C3 × 16^{-1}
3E8	3	0.4189 374B C6A7 EF9E × 16^{-2}
2710	4	0.68DB 8BAC 710C B296 × 16^{-3}
1 86A0	5	0.A7C5 AC47 1B47 8423 × 16^{-4}
F 4240	6	0.10C6 F7A0 B5ED 8D37 × 16^{-4}
98 9680	7	0.1AD7 F29A BCAF 4858 × 16^{-5}
5F5 E100	8	0.2AF3 1DC4 6118 73BF × 16^{-6}
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC × 16^{-7}
2 540B E400	10	0.6DF3 7F67 5EF6 EADF × 16^{-8}
17 4876 E800	11	0.AFEB FF0B CB24 AAFF × 16^{-9}
E8 D4A5 1000	12	0.1197 9981 2DEA 1119 × 16^{-9}
918 4E72 A000	13	0.1C25 C268 4976 81C2 × 16^{-10}
5AF3 107A 4000	14	0.2D09 370D 4257 3604 × 16^{-11}
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D × 16^{-12}
23 86F2 6FC1 0000	16	0.734A CA5F 6226 F0AE × 16^{-13}
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 × 16^{-14}
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1 × 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 × 16^{-15}

HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE

The table below provides for direct conversions between hexadecimal integers in the range 0—FFF and decimal integers in the range 0—4095. For conversion of larger integers, the table values may be added to the following figures:

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Hexadecimal fractions may be converted to decimal fractions as follows:

1. Express the hexadecimal fraction as an integer times 16^{-n}, where n is the number of significant hexadecimal places to the right of the hexadecimal point.

$$0.\,CA9BF3_{16} = CA9\ BF3_{16} \times 16^{-6}$$

2. Find the decimal equivalent of the hexadecimal integer

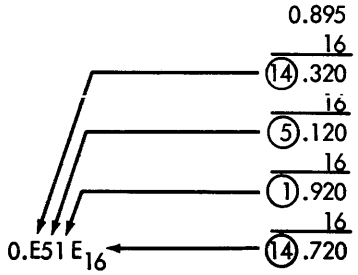
$$CA9\ BF3_{16} = 13\ 278\ 195_{10}$$

3. Multiply the decimal equivalent by 16^{-n}

$$\begin{array}{l} 13\ 278\ 195 \\ \times\ 596\ 046\ 448 \times 10^{-16} \\ \hline 0.791\ 442\ 096_{10} \end{array}$$

Decimal fractions may be converted to hexadecimal fractions by successively multiplying the decimal fraction by 16_{10}. After each multiplication, the integer portion is removed to form a hexadecimal fraction by building to the right of the hexadecimal point. However, since decimal arithmetic is used in this conversion, the integer portion of each product must be converted to hexadecimal numbers.

Example: Convert 0.895_{10} to its hexadecimal equivalent



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

HEXADECIMAL-DECIMAL FRACTION CONVERSION TABLE

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.40 00 00 00	.25000 00000	.80 00 00 00	.50000 00000	.C0 00 00 00	.75000 00000
.01 00 00 00	.00390 62500	.41 00 00 00	.25390 62500	.81 00 00 00	.50390 62500	.C1 00 00 00	.75390 62500
.02 00 00 00	.00781 25000	.42 00 00 00	.25781 25000	.82 00 00 00	.50781 25000	.C2 00 00 00	.75781 25000
.03 00 00 00	.01171 87500	.43 00 00 00	.26171 87500	.83 00 00 00	.51171 87500	.C3 00 00 00	.76171 87500
.04 00 00 00	.01562 50000	.44 00 00 00	.26562 50000	.84 00 00 00	.51562 50000	.C4 00 00 00	.76562 50000
.05 00 00 00	.01953 12500	.45 00 00 00	.26953 12500	.85 00 00 00	.51953 12500	.C5 00 00 00	.76953 12500
.06 00 00 00	.02343 75000	.46 00 00 00	.27343 75000	.86 00 00 00	.52343 75000	.C6 00 00 00	.77343 75000
.07 00 00 00	.02734 37500	.47 00 00 00	.27734 37500	.87 00 00 00	.52734 37500	.C7 00 00 00	.77734 37500
.08 00 00 00	.03125 00000	.48 00 00 00	.28125 00000	.88 00 00 00	.53125 00000	.C8 00 00 00	.78125 00000
.09 00 00 00	.03515 62500	.49 00 00 00	.28515 62500	.89 00 00 00	.53515 62500	.C9 00 00 00	.78515 62500
.0A 00 00 00	.03906 25000	.4A 00 00 00	.28906 25000	.8A 00 00 00	.53906 25000	.CA 00 00 00	.78906 25000
.0B 00 00 00	.04296 87500	.4B 00 00 00	.29296 87500	.8B 00 00 00	.54296 87500	.CB 00 00 00	.79296 87500
.0C 00 00 00	.04687 50000	.4C 00 00 00	.29687 50000	.8C 00 00 00	.54687 50000	.CC 00 00 00	.79687 50000
.0D 00 00 00	.05078 12500	.4D 00 00 00	.30078 12500	.8D 00 00 00	.55078 12500	.CD 00 00 00	.80078 12500
.0E 00 00 00	.05468 75000	.4E 00 00 00	.30468 75000	.8E 00 00 00	.55468 75000	.CE 00 00 00	.80468 75000
.0F 00 00 00	.05859 37500	.4F 00 00 00	.30859 37500	.8F 00 00 00	.55859 37500	.CF 00 00 00	.80859 37500
.10 00 00 00	.06250 00000	.50 00 00 00	.31250 00000	.90 00 00 00	.56250 00000	.D0 00 00 00	.81250 00000
.11 00 00 00	.06640 62500	.51 00 00 00	.31640 62500	.91 00 00 00	.56640 62500	.D1 00 00 00	.81640 62500
.12 00 00 00	.07031 25000	.52 00 00 00	.32031 25000	.92 00 00 00	.57031 25000	.D2 00 00 00	.82031 25000
.13 00 00 00	.07421 87500	.53 00 00 00	.32421 87500	.93 00 00 00	.57421 87500	.D3 00 00 00	.82421 87500
.14 00 00 00	.07812 50000	.54 00 00 00	.32812 50000	.94 00 00 00	.57812 50000	.D4 00 00 00	.82812 50000
.15 00 00 00	.08203 12500	.55 00 00 00	.33203 12500	.95 00 00 00	.58203 12500	.D5 00 00 00	.83203 12500
.16 00 00 00	.08593 75000	.56 00 00 00	.33593 75000	.96 00 00 00	.58593 75000	.D6 00 00 00	.83593 75000
.17 00 00 00	.08984 37500	.57 00 00 00	.33984 37500	.97 00 00 00	.58984 37500	.D7 00 00 00	.83984 37500
.18 00 00 00	.09375 00000	.58 00 00 00	.34375 00000	.98 00 00 00	.59375 00000	.D8 00 00 00	.84375 00000
.19 00 00 00	.09765 62500	.59 00 00 00	.34765 62500	.99 00 00 00	.59765 62500	.D9 00 00 00	.84765 62500
.1A 00 00 00	.10156 25000	.5A 00 00 00	.35156 25000	.9A 00 00 00	.60156 25000	.DA 00 00 00	.85156 25000
.1B 00 00 00	.10546 87500	.5B 00 00 00	.35546 87500	.9B 00 00 00	.60546 87500	.DB 00 00 00	.85546 87500
.1C 00 00 00	.10937 50000	.5C 00 00 00	.35937 50000	.9C 00 00 00	.60937 50000	.DC 00 00 00	.85937 50000
.1D 00 00 00	.11328 12500	.5D 00 00 00	.36328 12500	.9D 00 00 00	.61328 12500	.DD 00 00 00	.86328 12500
.1E 00 00 00	.11718 75000	.5E 00 00 00	.36718 75000	.9E 00 00 00	.61718 75000	.DE 00 00 00	.86718 75000
.1F 00 00 00	.12109 37500	.5F 00 00 00	.37109 37500	.9F 00 00 00	.62109 37500	.DF 00 00 00	.87109 37500
.20 00 00 00	.12500 00000	.60 00 00 00	.37500 00000	.A0 00 00 00	.62500 00000	.E0 00 00 00	.87500 00000
.21 00 00 00	.12890 62500	.61 00 00 00	.37890 62500	.A1 00 00 00	.62890 62500	.E1 00 00 00	.87890 62500
.22 00 00 00	.13281 25000	.62 00 00 00	.38281 25000	.A2 00 00 00	.63281 25000	.E2 00 00 00	.88281 25000
.23 00 00 00	.13671 87500	.63 00 00 00	.38671 87500	.A3 00 00 00	.63671 87500	.E3 00 00 00	.88671 87500
.24 00 00 00	.14062 50000	.64 00 00 00	.39062 50000	.A4 00 00 00	.64062 50000	.E4 00 00 00	.89062 50000
.25 00 00 00	.14453 12500	.65 00 00 00	.39453 12500	.A5 00 00 00	.64453 12500	.E5 00 00 00	.89453 12500
.26 00 00 00	.14843 75000	.66 00 00 00	.39843 75000	.A6 00 00 00	.64843 75000	.E6 00 00 00	.89843 75000
.27 00 00 00	.15234 37500	.67 00 00 00	.40234 37500	.A7 00 00 00	.65234 37500	.E7 00 00 00	.90234 37500
.28 00 00 00	.15625 00000	.68 00 00 00	.40625 00000	.A8 00 00 00	.65625 00000	.E8 00 00 00	.90625 00000
.29 00 00 00	.16015 62500	.69 00 00 00	.41015 62500	.A9 00 00 00	.66015 62500	.E9 00 00 00	.91015 62500
.2A 00 00 00	.16406 25000	.6A 00 00 00	.41406 25000	.AA 00 00 00	.66406 25000	.EA 00 00 00	.91406 25000
.2B 00 00 00	.16796 87500	.6B 00 00 00	.41796 87500	.AB 00 00 00	.66796 87500	.EB 00 00 00	.91796 87500
.2C 00 00 00	.17187 50000	.6C 00 00 00	.42187 50000	.AC 00 00 00	.67187 50000	.EC 00 00 00	.92187 50000
.2D 00 00 00	.17578 12500	.6D 00 00 00	.42578 12500	.AD 00 00 00	.67578 12500	.ED 00 00 00	.92578 12500
.2E 00 00 00	.17968 75000	.6E 00 00 00	.42968 75000	.AE 00 00 00	.67968 75000	.EE 00 00 00	.92968 75000
.2F 00 00 00	.18359 37500	.6F 00 00 00	.43359 37500	.AF 00 00 00	.68359 37500	.EF 00 00 00	.93359 37500
.30 00 00 00	.18750 00000	.70 00 00 00	.43750 00000	.B0 00 00 00	.68750 00000	.F0 00 00 00	.93750 00000
.31 00 00 00	.19140 62500	.71 00 00 00	.44140 62500	.B1 00 00 00	.69140 62500	.F1 00 00 00	.94140 62500
.32 00 00 00	.19531 25000	.72 00 00 00	.44531 25000	.B2 00 00 00	.69531 25000	.F2 00 00 00	.94531 25000
.33 00 00 00	.19921 87500	.73 00 00 00	.44921 87500	.B3 00 00 00	.69921 87500	.F3 00 00 00	.94921 87500
.34 00 00 00	.20312 50000	.74 00 00 00	.45312 50000	.B4 00 00 00	.70312 50000	.F4 00 00 00	.95312 50000
.35 00 00 00	.20703 12500	.75 00 00 00	.45703 12500	.B5 00 00 00	.70703 12500	.F5 00 00 00	.95703 12500
.36 00 00 00	.21093 75000	.76 00 00 00	.46093 75000	.B6 00 00 00	.71093 75000	.F6 00 00 00	.96093 75000
.37 00 00 00	.21484 37500	.77 00 00 00	.46484 37500	.B7 00 00 00	.71484 37500	.F7 00 00 00	.96484 37500
.38 00 00 00	.21875 00000	.78 00 00 00	.46875 00000	.B8 00 00 00	.71875 00000	.F8 00 00 00	.96875 00000
.39 00 00 00	.22265 62500	.79 00 00 00	.47265 62500	.B9 00 00 00	.72265 62500	.F9 00 00 00	.97265 62500
.3A 00 00 00	.22656 25000	.7A 00 00 00	.47656 25000	.BA 00 00 00	.72656 25000	.FA 00 00 00	.97656 25000
.3B 00 00 00	.23046 87500	.7B 00 00 00	.48046 87500	.BB 00 00 00	.73046 87500	.FB 00 00 00	.98046 87500
.3C 00 00 00	.23437 50000	.7C 00 00 00	.48437 50000	.BC 00 00 00	.73437 50000	.FC 00 00 00	.98437 50000
.3D 00 00 00	.23828 12500	.7D 00 00 00	.48828 12500	.BD 00 00 00	.73828 12500	.FD 00 00 00	.98828 12500
.3E 00 00 00	.24218 75000	.7E 00 00 00	.49218 75000	.BE 00 00 00	.74218 75000	.FE 00 00 00	.99218 75000
.3F 00 00 00	.24609 37500	.7F 00 00 00	.49609 37500	.BF 00 00 00	.74609 37500	.FF 00 00 00	.99609 37500

HEXADECIMAL-DECIMAL FRACTION CONVERSION TABLE (cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 40 00 00	.00097 65625	.00 80 00 00	.00195 31250	.00 C0 00 00	.00292 96875
.00 01 00 00	.00001 52587	.00 41 00 00	.00099 18212	.00 81 00 00	.00196 83837	.00 C1 00 00	.00294 49462
.00 02 00 00	.00003 05175	.00 42 00 00	.00100 70800	.00 82 00 00	.00198 36425	.00 C2 00 00	.00296 02050
.00 03 00 00	.00004 57763	.00 43 00 00	.00102 23388	.00 83 00 00	.00199 89013	.00 C3 00 00	.00297 54638
.00 04 00 00	.00006 10351	.00 44 00 00	.00103 75976	.00 84 00 00	.00201 41601	.00 C4 00 00	.00299 07226
.00 05 00 00	.00007 62939	.00 45 00 00	.00105 28564	.00 85 00 00	.00202 94189	.00 C5 00 00	.00300 59814
.00 06 00 00	.00009 15527	.00 46 00 00	.00106 81152	.00 86 00 00	.00204 46777	.00 C6 00 00	.00302 12402
.00 07 00 00	.00010 68115	.00 47 00 00	.00108 33740	.00 87 00 00	.00205 99365	.00 C7 00 00	.00303 64990
.00 08 00 00	.00012 20703	.00 48 00 00	.00109 86328	.00 88 00 00	.00207 51953	.00 C8 00 00	.00305 17578
.00 09 00 00	.00013 73291	.00 49 00 00	.00111 38916	.00 89 00 00	.00209 04541	.00 C9 00 00	.00306 70166
.00 0A 00 00	.00015 25878	.00 4A 00 00	.00112 91503	.00 8A 00 00	.00210 57128	.00 CA 00 00	.00308 22753
.00 0B 00 00	.00016 78466	.00 4B 00 00	.00114 44091	.00 8B 00 00	.00212 09716	.00 CB 00 00	.00309 75341
.00 0C 00 00	.00018 31054	.00 4C 00 00	.00115 96679	.00 8C 00 00	.00213 62304	.00 CC 00 00	.00311 27929
.00 0D 00 00	.00019 83642	.00 4D 00 00	.00117 49267	.00 8D 00 00	.00215 14892	.00 CD 00 00	.00312 80517
.00 0E 00 00	.00021 36230	.00 4E 00 00	.00119 01855	.00 8E 00 00	.00216 67480	.00 CE 00 00	.00314 33105
.00 0F 00 00	.00022 88818	.00 4F 00 00	.00120 54443	.00 8F 00 00	.00218 20068	.00 CF 00 00	.00315 85693
.00 10 00 00	.00024 41406	.00 50 00 00	.00122 07031	.00 90 00 00	.00219 72656	.00 D0 00 00	.00317 38281
.00 11 00 00	.00025 93994	.00 51 00 00	.00123 59619	.00 91 00 00	.00221 25244	.00 D1 00 00	.00318 90869
.00 12 00 00	.00027 46582	.00 52 00 00	.00125 12207	.00 92 00 00	.00222 77832	.00 D2 00 00	.00320 43457
.00 13 00 00	.00028 99169	.00 53 00 00	.00126 64794	.00 93 00 00	.00224 30419	.00 D3 00 00	.00321 96044
.00 14 00 00	.00030 51757	.00 54 00 00	.00128 17382	.00 94 00 00	.00225 83007	.00 D4 00 00	.00323 48632
.00 15 00 00	.00032 04345	.00 55 00 00	.00129 69970	.00 95 00 00	.00227 35595	.00 D5 00 00	.00325 01220
.00 16 00 00	.00033 56933	.00 56 00 00	.00131 22558	.00 96 00 00	.00228 88183	.00 D6 00 00	.00326 53808
.00 17 00 00	.00035 09521	.00 57 00 00	.00132 75146	.00 97 00 00	.00230 40771	.00 D7 00 00	.00328 06396
.00 18 00 00	.00036 62109	.00 58 00 00	.00134 27734	.00 98 00 00	.00231 93359	.00 D8 00 00	.00329 58984
.00 19 00 00	.00038 14697	.00 59 00 00	.00135 80322	.00 99 00 00	.00233 45947	.00 D9 00 00	.00331 11572
.00 1A 00 00	.00039 67285	.00 5A 00 00	.00137 32910	.00 9A 00 00	.00234 98535	.00 DA 00 00	.00332 64160
.00 1B 00 00	.00041 19873	.00 5B 00 00	.00138 85498	.00 9B 00 00	.00236 51123	.00 DB 00 00	.00334 16748
.00 1C 00 00	.00042 72460	.00 5C 00 00	.00140 38085	.00 9C 00 00	.00238 03710	.00 DC 00 00	.00335 69335
.00 1D 00 00	.00044 25048	.00 5D 00 00	.00141 90673	.00 9D 00 00	.00239 56298	.00 DD 00 00	.00337 21923
.00 1E 00 00	.00045 77636	.00 5E 00 00	.00143 43261	.00 9E 00 00	.00241 08886	.00 DE 00 00	.00338 74511
.00 1F 00 00	.00047 30224	.00 5F 00 00	.00144 95849	.00 9F 00 00	.00242 61474	.00 DF 00 00	.00340 27099
.00 20 00 00	.00048 82812	.00 60 00 00	.00146 48437	.00 A0 00 00	.00244 14062	.00 E0 00 00	.00341 79687
.00 21 00 00	.00050 35400	.00 61 00 00	.00148 01025	.00 A1 00 00	.00245 66650	.00 E1 00 00	.00343 32275
.00 22 00 00	.00051 87988	.00 62 00 00	.00149 53613	.00 A2 00 00	.00247 19238	.00 E2 00 00	.00344 84863
.00 23 00 00	.00053 40576	.00 63 00 00	.00151 06201	.00 A3 00 00	.00248 71826	.00 E3 00 00	.00346 37451
.00 24 00 00	.00054 93164	.00 64 00 00	.00152 58789	.00 A4 00 00	.00250 24414	.00 E4 00 00	.00347 90039
.00 25 00 00	.00056 45751	.00 65 00 00	.00154 11376	.00 A5 00 00	.00251 77001	.00 E5 00 00	.00349 42626
.00 26 00 00	.00057 98339	.00 66 00 00	.00155 63964	.00 A6 00 00	.00253 29589	.00 E6 00 00	.00350 95214
.00 27 00 00	.00059 50927	.00 67 00 00	.00157 16552	.00 A7 00 00	.00254 82177	.00 E7 00 00	.00352 47802
.00 28 00 00	.00061 03515	.00 68 00 00	.00158 69140	.00 A8 00 00	.00256 34765	.00 E8 00 00	.00354 00390
.00 29 00 00	.00062 56103	.00 69 00 00	.00160 21728	.00 A9 00 00	.00257 87353	.00 E9 00 00	.00355 52978
.00 2A 00 00	.00064 08691	.00 6A 00 00	.00161 74316	.00 AA 00 00	.00259 39941	.00 EA 00 00	.00357 05566
.00 2B 00 00	.00065 61279	.00 6B 00 00	.00163 26904	.00 AB 00 00	.00260 92529	.00 EB 00 00	.00358 58154
.00 2C 00 00	.00067 13867	.00 6C 00 00	.00164 79492	.00 AC 00 00	.00262 45117	.00 EC 00 00	.00360 10742
.00 2D 00 00	.00068 66455	.00 6D 00 00	.00166 32080	.00 AD 00 00	.00263 97705	.00 ED 00 00	.00361 63330
.00 2E 00 00	.00070 19042	.00 6E 00 00	.00167 84667	.00 AE 00 00	.00265 50292	.00 EE 00 00	.00363 15917
.00 2F 00 00	.00071 71630	.00 6F 00 00	.00169 37255	.00 AF 00 00	.00267 02880	.00 EF 00 00	.00364 68505
.00 30 00 00	.00073 24218	.00 70 00 00	.00170 89843	.00 B0 00 00	.00268 55468	.00 F0 00 00	.00366 21093
.00 31 00 00	.00074 76806	.00 71 00 00	.00172 42431	.00 B1 00 00	.00270 08056	.00 F1 00 00	.00367 73681
.00 32 00 00	.00076 29394	.00 72 00 00	.00173 95019	.00 B2 00 00	.00271 60644	.00 F2 00 00	.00369 26269
.00 33 00 00	.00077 81982	.00 73 00 00	.00175 47607	.00 B3 00 00	.00273 13232	.00 F3 00 00	.00370 78857
.00 34 00 00	.00079 34570	.00 74 00 00	.00177 00195	.00 B4 00 00	.00274 65820	.00 F4 00 00	.00372 31445
.00 35 00 00	.00080 87158	.00 75 00 00	.00178 52783	.00 B5 00 00	.00276 18408	.00 F5 00 00	.00373 84033
.00 36 00 00	.00082 39746	.00 76 00 00	.00180 05371	.00 B6 00 00	.00277 70996	.00 F6 00 00	.00375 36621
.00 37 00 00	.00083 92333	.00 77 00 00	.00181 57958	.00 B7 00 00	.00279 23583	.00 F7 00 00	.00376 89208
.00 38 00 00	.00085 44921	.00 78 00 00	.00183 10546	.00 B8 00 00	.00280 76171	.00 F8 00 00	.00378 41796
.00 39 00 00	.00086 97509	.00 79 00 00	.00184 63134	.00 B9 00 00	.00282 28759	.00 F9 00 00	.00379 94384
.00 3A 00 00	.00088 50097	.00 7A 00 00	.00186 15722	.00 BA 00 00	.00283 81347	.00 FA 00 00	.00381 46972
.00 3B 00 00	.00090 02685	.00 7B 00 00	.00187 68310	.00 BB 00 00	.00285 33935	.00 FB 00 00	.00382 99560
.00 3C 00 00	.00091 55273	.00 7C 00 00	.00189 20898	.00 BC 00 00	.00286 86523	.00 FC 00 00	.00384 52148
.00 3D 00 00	.00093 07861	.00 7D 00 00	.00190 73486	.00 BD 00 00	.00288 39111	.00 FD 00 00	.00386 04736
.00 3E 00 00	.00094 60449	.00 7E 00 00	.00192 26074	.00 BE 00 00	.00289 91699	.00 FE 00 00	.00387 57324
.00 3F 00 00	.00096 13037	.00 7F 00 00	.00193 78662	.00 BF 00 00	.00291 44287	.00 FF 00 00	.00389 09912

HEXADECIMAL-DECIMAL FRACTION CONVERSION TABLE (cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 40 00	.00000 38146	.00 00 80 00	.00000 76293	.00 00 C0 00	.00001 14440
.00 00 01 00	.00000 00596	.00 00 41 00	.00000 38743	.00 00 81 00	.00000 76889	.00 00 C1 00	.00001 15036
.00 00 02 00	.00000 01192	.00 00 42 00	.00000 39339	.00 00 82 00	.00000 77486	.00 00 C2 00	.00001 15633
.00 00 03 00	.00000 01788	.00 00 43 00	.00000 39935	.00 00 83 00	.00000 78082	.00 00 C3 00	.00001 16229
.00 00 04 00	.00000 02384	.00 00 44 00	.00000 40531	.00 00 84 00	.00000 78678	.00 00 C4 00	.00001 16825
.00 00 05 00	.00000 02980	.00 00 45 00	.00000 41127	.00 00 85 00	.00000 79274	.00 00 C5 00	.00001 17421
.00 00 06 00	.00000 03576	.00 00 46 00	.00000 41723	.00 00 86 00	.00000 79870	.00 00 C6 00	.00001 18017
.00 00 07 00	.00000 04172	.00 00 47 00	.00000 42319	.00 00 87 00	.00000 80466	.00 00 C7 00	.00001 18613
.00 00 08 00	.00000 04768	.00 00 48 00	.00000 42915	.00 00 88 00	.00000 81062	.00 00 C8 00	.00001 19209
.00 00 09 00	.00000 05364	.00 00 49 00	.00000 43511	.00 00 89 00	.00000 81658	.00 00 C9 00	.00001 19805
.00 00 0A 00	.00000 05960	.00 00 4A 00	.00000 44107	.00 00 8A 00	.00000 82254	.00 00 CA 00	.00001 20401
.00 00 0B 00	.00000 06556	.00 00 4B 00	.00000 44703	.00 00 8B 00	.00000 82850	.00 00 CB 00	.00001 20997
.00 00 0C 00	.00000 07152	.00 00 4C 00	.00000 45299	.00 00 8C 00	.00000 83446	.00 00 CC 00	.00001 21593
.00 00 0D 00	.00000 07748	.00 00 4D 00	.00000 45895	.00 00 8D 00	.00000 84042	.00 00 CD 00	.00001 22189
.00 00 0E 00	.00000 08344	.00 00 4E 00	.00000 46491	.00 00 8E 00	.00000 84638	.00 00 CE 00	.00001 22785
.00 00 0F 00	.00000 08940	.00 00 4F 00	.00000 47087	.00 00 8F 00	.00000 85234	.00 00 CF 00	.00001 23381
.00 00 10 00	.00000 09536	.00 00 50 00	.00000 47683	.00 00 90 00	.00000 85830	.00 00 D0 00	.00001 23977
.00 00 11 00	.00000 10132	.00 00 51 00	.00000 48279	.00 00 91 00	.00000 86426	.00 00 D1 00	.00001 24573
.00 00 12 00	.00000 10728	.00 00 52 00	.00000 48875	.00 00 92 00	.00000 87022	.00 00 D2 00	.00001 25169
.00 00 13 00	.00000 11324	.00 00 53 00	.00000 49471	.00 00 93 00	.00000 87618	.00 00 D3 00	.00001 25765
.00 00 14 00	.00000 11920	.00 00 54 00	.00000 50067	.00 00 94 00	.00000 88214	.00 00 D4 00	.00001 26361
.00 00 15 00	.00000 12516	.00 00 55 00	.00000 50663	.00 00 95 00	.00000 88810	.00 00 D5 00	.00001 26957
.00 00 16 00	.00000 13113	.00 00 56 00	.00000 51259	.00 00 96 00	.00000 89406	.00 00 D6 00	.00001 27553
.00 00 17 00	.00000 13709	.00 00 57 00	.00000 51856	.00 00 97 00	.00000 90003	.00 00 D7 00	.00001 28149
.00 00 18 00	.00000 14305	.00 00 58 00	.00000 52452	.00 00 98 00	.00000 90599	.00 00 D8 00	.00001 28746
.00 00 19 00	.00000 14901	.00 00 59 00	.00000 53048	.00 00 99 00	.00000 91195	.00 00 D9 00	.00001 29342
.00 00 1A 00	.00000 15497	.00 00 5A 00	.00000 53644	.00 00 9A 00	.00000 91791	.00 00 DA 00	.00001 29938
.00 00 1B 00	.00000 16093	.00 00 5B 00	.00000 54240	.00 00 9B 00	.00000 92387	.00 00 DB 00	.00001 30534
.00 00 1C 00	.00000 16689	.00 00 5C 00	.00000 54836	.00 00 9C 00	.00000 92983	.00 00 DC 00	.00001 31130
.00 00 1D 00	.00000 17285	.00 00 5D 00	.00000 55432	.00 00 9D 00	.00000 93579	.00 00 DD 00	.00001 31726
.00 00 1E 00	.00000 17881	.00 00 5E 00	.00000 56028	.00 00 9E 00	.00000 94175	.00 00 DE 00	.00001 32322
.00 00 1F 00	.00000 18477	.00 00 5F 00	.00000 56624	.00 00 9F 00	.00000 94771	.00 00 DF 00	.00001 32918
.00 00 20 00	.00000 19073	.00 00 60 00	.00000 57220	.00 00 A0 00	.00000 95367	.00 00 E0 00	.00001 33514
.00 00 21 00	.00000 19669	.00 00 61 00	.00000 57816	.00 00 A1 00	.00000 95963	.00 00 E1 00	.00001 34110
.00 00 22 00	.00000 20265	.00 00 62 00	.00000 58412	.00 00 A2 00	.00000 96559	.00 00 E2 00	.00001 34706
.00 00 23 00	.00000 20861	.00 00 63 00	.00000 59008	.00 00 A3 00	.00000 97155	.00 00 E3 00	.00001 35302
.00 00 24 00	.00000 21457	.00 00 64 00	.00000 59604	.00 00 A4 00	.00000 97751	.00 00 E4 00	.00001 35898
.00 00 25 00	.00000 22053	.00 00 65 00	.00000 60200	.00 00 A5 00	.00000 98347	.00 00 E5 00	.00001 36494
.00 00 26 00	.00000 22649	.00 00 66 00	.00000 60796	.00 00 A6 00	.00000 98943	.00 00 E6 00	.00001 37090
.00 00 27 00	.00000 23245	.00 00 67 00	.00000 61392	.00 00 A7 00	.00000 99539	.00 00 E7 00	.00001 37686
.00 00 28 00	.00000 23841	.00 00 68 00	.00000 61988	.00 00 A8 00	.00001 00135	.00 00 E8 00	.00001 38282
.00 00 29 00	.00000 24437	.00 00 69 00	.00000 62584	.00 00 A9 00	.00001 00731	.00 00 E9 00	.00001 38878
.00 00 2A 00	.00000 25033	.00 00 6A 00	.00000 63180	.00 00 AA 00	.00001 01327	.00 00 EA 00	.00001 39474
.00 00 2B 00	.00000 25629	.00 00 6B 00	.00000 63776	.00 00 AB 00	.00001 01923	.00 00 EB 00	.00001 40070
.00 00 2C 00	.00000 26226	.00 00 6C 00	.00000 64373	.00 00 AC 00	.00001 02519	.00 00 EC 00	.00001 40666
.00 00 2D 00	.00000 26822	.00 00 6D 00	.00000 64969	.00 00 AD 00	.00001 03116	.00 00 ED 00	.00001 41263
.00 00 2E 00	.00000 27418	.00 00 6E 00	.00000 65565	.00 00 AE 00	.00001 03712	.00 00 EE 00	.00001 41859
.00 00 2F 00	.00000 28014	.00 00 6F 00	.00000 66161	.00 00 AF 00	.00001 04308	.00 00 EF 00	.00001 42455
.00 00 30 00	.00000 28610	.00 00 70 00	.00000 66757	.00 00 B0 00	.00001 04904	.00 00 F0 00	.00001 43051
.00 00 31 00	.00000 29206	.00 00 71 00	.00000 67353	.00 00 B1 00	.00001 05500	.00 00 F1 00	.00001 43647
.00 00 32 00	.00000 29802	.00 00 72 00	.00000 67949	.00 00 B2 00	.00001 06096	.00 00 F2 00	.00001 44243
.00 00 33 00	.00000 30398	.00 00 73 00	.00000 68545	.00 00 B3 00	.00001 06692	.00 00 F3 00	.00001 44839
.00 00 34 00	.00000 30994	.00 00 74 00	.00000 69141	.00 00 B4 00	.00001 07288	.00 00 F4 00	.00001 45435
.00 00 35 00	.00000 31590	.00 00 75 00	.00000 69737	.00 00 B5 00	.00001 07884	.00 00 F5 00	.00001 46031
.00 00 36 00	.00000 32186	.00 00 76 00	.00000 70333	.00 00 B6 00	.00001 08480	.00 00 F6 00	.00001 46627
.00 00 37 00	.00000 32782	.00 00 77 00	.00000 70929	.00 00 B7 00	.00001 09076	.00 00 F7 00	.00001 47223
.00 00 38 00	.00000 33378	.00 00 78 00	.00000 71525	.00 00 B8 00	.00001 09672	.00 00 F8 00	.00001 47819
.00 00 39 00	.00000 33974	.00 00 79 00	.00000 72121	.00 00 B9 00	.00001 10268	.00 00 F9 00	.00001 48415
.00 00 3A 00	.00000 34570	.00 00 7A 00	.00000 72717	.00 00 BA 00	.00001 10864	.00 00 FA 00	.00001 49011
.00 00 3B 00	.00000 35166	.00 00 7B 00	.00000 73313	.00 00 BB 00	.00001 11460	.00 00 FB 00	.00001 49607
.00 00 3C 00	.00000 35762	.00 00 7C 00	.00000 73909	.00 00 BC 00	.00001 12056	.00 00 FC 00	.00001 50203
.00 00 3D 00	.00000 36358	.00 00 7D 00	.00000 74505	.00 00 BD 00	.00001 12652	.00 00 FD 00	.00001 50799
.00 00 3E 00	.00000 36954	.00 00 7E 00	.00000 75101	.00 00 BE 00	.00001 13248	.00 00 FE 00	.00001 51395
.00 00 3F 00	.00000 37550	.00 00 7F 00	.00000 75697	.00 00 BF 00	.00001 13844	.00 00 FF 00	.00001 51991

HEXADECIMAL-DECIMAL FRACTION CONVERSION TABLE (cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 00 40	.00000 00149	.00 00 00 80	.00000 00298	.00 00 00 C0	.00000 00447
.00 00 00 01	.00000 00002	.00 00 00 41	.00000 00151	.00 00 00 81	.00000 00300	.00 00 00 C1	.00000 00449
.00 00 00 02	.00000 00004	.00 00 00 42	.00000 00153	.00 00 00 82	.00000 00302	.00 00 00 C2	.00000 00451
.00 00 00 03	.00000 00006	.00 00 00 43	.00000 00155	.00 00 00 83	.00000 00305	.00 00 00 C3	.00000 00454
.00 00 00 04	.00000 00009	.00 00 00 44	.00000 00158	.00 00 00 84	.00000 00307	.00 00 00 C4	.00000 00456
.00 00 00 05	.00000 00011	.00 00 00 45	.00000 00160	.00 00 00 85	.00000 00309	.00 00 00 C5	.00000 00458
.00 00 00 06	.00000 00013	.00 00 00 46	.00000 00162	.00 00 00 86	.00000 00311	.00 00 00 C6	.00000 00461
.00 00 00 07	.00000 00016	.00 00 00 47	.00000 00165	.00 00 00 87	.00000 00314	.00 00 00 C7	.00000 00463
.00 00 00 08	.00000 00018	.00 00 00 48	.00000 00167	.00 00 00 88	.00000 00316	.00 00 00 C8	.00000 00465
.00 00 00 09	.00000 00020	.00 00 00 49	.00000 00169	.00 00 00 89	.00000 00318	.00 00 00 C9	.00000 00467
.00 00 00 0A	.00000 00023	.00 00 00 4A	.00000 00172	.00 00 00 8A	.00000 00321	.00 00 00 CA	.00000 00470
.00 00 00 0B	.00000 00025	.00 00 00 4B	.00000 00174	.00 00 00 8B	.00000 00323	.00 00 00 CB	.00000 00472
.00 00 00 0C	.00000 00027	.00 00 00 4C	.00000 00176	.00 00 00 8C	.00000 00325	.00 00 00 CC	.00000 00474
.00 00 00 0D	.00000 00030	.00 00 00 4D	.00000 00179	.00 00 00 8D	.00000 00328	.00 00 00 CD	.00000 00477
.00 00 00 0E	.00000 00032	.00 00 00 4E	.00000 00181	.00 00 00 8E	.00000 00330	.00 00 00 CE	.00000 00479
.00 00 00 0F	.00000 00034	.00 00 00 4F	.00000 00183	.00 00 00 8F	.00000 00332	.00 00 00 CF	.00000 00481
.00 00 00 10	.00000 00037	.00 00 00 50	.00000 00186	.00 00 00 90	.00000 00335	.00 00 00 D0	.00000 00484
.00 00 00 11	.00000 00039	.00 00 00 51	.00000 00188	.00 00 00 91	.00000 00337	.00 00 00 D1	.00000 00486
.00 00 00 12	.00000 00041	.00 00 00 52	.00000 00190	.00 00 00 92	.00000 00339	.00 00 00 D2	.00000 00488
.00 00 00 13	.00000 00044	.00 00 00 53	.00000 00193	.00 00 00 93	.00000 00342	.00 00 00 D3	.00000 00491
.00 00 00 14	.00000 00046	.00 00 00 54	.00000 00195	.00 00 00 94	.00000 00344	.00 00 00 D4	.00000 00493
.00 00 00 15	.00000 00048	.00 00 00 55	.00000 00197	.00 00 00 95	.00000 00346	.00 00 00 D5	.00000 00495
.00 00 00 16	.00000 00051	.00 00 00 56	.00000 00200	.00 00 00 96	.00000 00349	.00 00 00 D6	.00000 00498
.00 00 00 17	.00000 00053	.00 00 00 57	.00000 00202	.00 00 00 97	.00000 00351	.00 00 00 D7	.00000 00500
.00 00 00 18	.00000 00055	.00 00 00 58	.00000 00204	.00 00 00 98	.00000 00353	.00 00 00 D8	.00000 00502
.00 00 00 19	.00000 00058	.00 00 00 59	.00000 00207	.00 00 00 99	.00000 00356	.00 00 00 D9	.00000 00505
.00 00 00 1A	.00000 00060	.00 00 00 5A	.00000 00209	.00 00 00 9A	.00000 00358	.00 00 00 DA	.00000 00507
.00 00 00 1B	.00000 00062	.00 00 00 5B	.00000 00211	.00 00 00 9B	.00000 00360	.00 00 00 DB	.00000 00509
.00 00 00 1C	.00000 00065	.00 00 00 5C	.00000 00214	.00 00 00 9C	.00000 00363	.00 00 00 DC	.00000 00512
.00 00 00 1D	.00000 00067	.00 00 00 5D	.00000 00216	.00 00 00 9D	.00000 00365	.00 00 00 DD	.00000 00514
.00 00 00 1E	.00000 00069	.00 00 00 5E	.00000 00218	.00 00 00 9E	.00000 00367	.00 00 00 DE	.00000 00516
.00 00 00 1F	.00000 00072	.00 00 00 5F	.00000 00221	.00 00 00 9F	.00000 00370	.00 00 00 DF	.00000 00519
.00 00 00 20	.00000 00074	.00 00 00 60	.00000 00223	.00 00 00 A0	.00000 00372	.00 00 00 E0	.00000 00521
.00 00 00 21	.00000 00076	.00 00 00 61	.00000 00225	.00 00 00 A1	.00000 00374	.00 00 00 E1	.00000 00523
.00 00 00 22	.00000 00079	.00 00 00 62	.00000 00228	.00 00 00 A2	.00000 00377	.00 00 00 E2	.00000 00526
.00 00 00 23	.00000 00081	.00 00 00 63	.00000 00230	.00 00 00 A3	.00000 00379	.00 00 00 E3	.00000 00528
.00 00 00 24	.00000 00083	.00 00 00 64	.00000 00232	.00 00 00 A4	.00000 00381	.00 00 00 E4	.00000 00530
.00 00 00 25	.00000 00086	.00 00 00 65	.00000 00235	.00 00 00 A5	.00000 00384	.00 00 00 E5	.00000 00533
.00 00 00 26	.00000 00088	.00 00 00 66	.00000 00237	.00 00 00 A6	.00000 00386	.00 00 00 E6	.00000 00535
.00 00 00 27	.00000 00090	.00 00 00 67	.00000 00239	.00 00 00 A7	.00000 00388	.00 00 00 E7	.00000 00537
.00 00 00 28	.00000 00093	.00 00 00 68	.00000 00242	.00 00 00 A8	.00000 00391	.00 00 00 E8	.00000 00540
.00 00 00 29	.00000 00095	.00 00 00 69	.00000 00244	.00 00 00 A9	.00000 00393	.00 00 00 E9	.00000 00542
.00 00 00 2A	.00000 00097	.00 00 00 6A	.00000 00246	.00 00 00 AA	.00000 00395	.00 00 00 EA	.00000 00544
.00 00 00 2B	.00000 00100	.00 00 00 6B	.00000 00249	.00 00 00 AB	.00000 00398	.00 00 00 EB	.00000 00547
.00 00 00 2C	.00000 00102	.00 00 00 6C	.00000 00251	.00 00 00 AC	.00000 00400	.00 00 00 EC	.00000 00549
.00 00 00 2D	.00000 00104	.00 00 00 6D	.00000 00253	.00 00 00 AD	.00000 00402	.00 00 00 ED	.00000 00551
.00 00 00 2E	.00000 00107	.00 00 00 6E	.00000 00256	.00 00 00 AE	.00000 00405	.00 00 00 EE	.00000 00554
.00 00 00 2F	.00000 00109	.00 00 00 6F	.00000 00258	.00 00 00 AF	.00000 00407	.00 00 00 EF	.00000 00556
.00 00 00 30	.00000 00111	.00 00 00 70	.00000 00260	.00 00 00 B0	.00000 00409	.00 00 00 F0	.00000 00558
.00 00 00 31	.00000 00114	.00 00 00 71	.00000 00263	.00 00 00 B1	.00000 00412	.00 00 00 F1	.00000 00561
.00 00 00 32	.00000 00116	.00 00 00 72	.00000 00265	.00 00 00 B2	.00000 00414	.00 00 00 F2	.00000 00563
.00 00 00 33	.00000 00118	.00 00 00 73	.00000 00267	.00 00 00 B3	.00000 00416	.00 00 00 F3	.00000 00565
.00 00 00 34	.00000 00121	.00 00 00 74	.00000 00270	.00 00 00 B4	.00000 00419	.00 00 00 F4	.00000 00568
.00 00 00 35	.00000 00123	.00 00 00 75	.00000 00272	.00 00 00 B5	.00000 00421	.00 00 00 F5	.00000 00570
.00 00 00 36	.00000 00125	.00 00 00 76	.00000 00274	.00 00 00 B6	.00000 00423	.00 00 00 F6	.00000 00572
.00 00 00 37	.00000 00128	.00 00 00 77	.00000 00277	.00 00 00 B7	.00000 00426	.00 00 00 F7	.00000 00575
.00 00 00 38	.00000 00130	.00 00 00 78	.00000 00279	.00 00 00 B8	.00000 00428	.00 00 00 F8	.00000 00577
.00 00 00 39	.00000 00132	.00 00 00 79	.00000 00281	.00 00 00 B9	.00000 00430	.00 00 00 F9	.00000 00579
.00 00 00 3A	.00000 00135	.00 00 00 7A	.00000 00284	.00 00 00 BA	.00000 00433	.00 00 00 FA	.00000 00582
.00 00 00 3B	.00000 00137	.00 00 00 7B	.00000 00286	.00 00 00 BB	.00000 00435	.00 00 00 FB	.00000 00584
.00 00 00 3C	.00000 00139	.00 00 00 7C	.00000 00288	.00 00 00 BC	.00000 00437	.00 00 00 FC	.00000 00586
.00 00 00 3D	.00000 00142	.00 00 00 7D	.00000 00291	.00 00 00 BD	.00000 00440	.00 00 00 FD	.00000 00589
.00 00 00 3E	.00000 00144	.00 00 00 7E	.00000 00293	.00 00 00 BE	.00000 00442	.00 00 00 FE	.00000 00591
.00 00 00 3F	.00000 00146	.00 00 00 7F	.00000 00295	.00 00 00 BF	.00000 00444	.00 00 00 FF	.00000 00593

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

MATHEMATICAL CONSTANTS

Constant	Decimal Value	Hexadecimal Value
π	3.14159 26535 89793	3.243F 6A89
π^{-1}	0.31830 98861 83790	0.517C C1B7
$\sqrt{\pi}$	1.77245 38509 05516	1.C5BF 891C
$\ln \pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.B7E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10} e$	0.43429 44819 03252	0.6F2D EC55
$\log_2 e$	1.44269 50408 88963	1.7154 7653
γ	0.57721 56649 01533	0.93C4 67E4
$\ln \gamma$	-0.54953 93129 81645	-0.8CAE 9BC1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
$\ln 2$	0.69314 71805 59945	0.B172 17F8
$\log_{10} 2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.298B 075C
$\ln 10$	2.30258 40929 94046	2.4D76 3777

APPENDIX B. GLOSSARY OF SYMBOLIC TERMS

Term	Meaning
( )	Contents of.
∩	AND (logical product, where 0 ∩ 0 = 0, 0 ∩ 1 = 0, 1 ∩ 0 = 0, and 1 ∩ 1 = 1).
∪	OR (logical inclusive OR, where 0 ∪ 0 = 0, 0 ∪ 1 = 1, 1 ∪ 0 = 1, and 1 ∪ 1 = 1).
Ⓤ	EOR (logical exclusive OR, where 0 Ⓤ 0 = 0, 0 Ⓤ 1 = 1, 1 Ⓤ 0 = 1, and 1 Ⓤ 1 = 0).
AM	Fixed-point arithmetic trap mask—bit position 11 of PSWs. If set (=1), basic processor traps to location X'43' after executing an instruction causing fixed-point overflow; if not set, basic processor does not trap.
CC	Condition code – 4-bit value (bit positions labeled CC1, CC2, CC3, and CC4), established as part of the execution of most instructions.
CI	Counter interrupt group inhibit – bit position 37 of PSWs. If set (=1), all interrupt levels within this group are inhibited.
DA	Destination address—in byte-string instructions, address of the destination byte string.
DBS	Destination byte string—operand specified by byte-string instruction.
DECA	Decimal accumulator – general registers 12, 13, 14, and 15 in decimal instructions.
DM	Decimal arithmetic trap mask—bit position 10 of PSWs. When set (=1), decimal arithmetic fault trap is in effect.
EB	Effective byte – 8-bit contents of effective byte location (EBL).
EBL	Effective byte location – byte location pointed to by effective virtual address of an instruction for byte operation.
ED	Effective doubleword – 64-bit contents of effective doubleword location (EDL).
EDL	Effective doubleword location—doubleword location pointed to by effective virtual address of an instruction for a doubleword operation. If odd-numbered word location is specified, low-order bit of effective address field (bit position 31) is automatically
EDL (cont.)	forced to 0. Hence, odd-numbered word address (referring to middle of doubleword) designates same doubleword as even-numbered word address when used for a doubleword operation.
EDO	Effective decimal operand.
EH	Effective halfword – 16-bit contents of effective halfword location, or (EHL).
EHL	Effective halfword location—halfword location pointed to by effective virtual address of an instruction for halfword operation.
EI	External interrupt group inhibit – bit position 39 of PSWs. If set (=1), all interrupt levels within this group are inhibited.
ESA	Effective source address – in byte-string instructions, address of the source byte string.
EVA	Effective virtual address – virtual address value obtained as result of indirect addressing and/or indexing. This address value is independent of the program's actual location in main memory, and is final address value before memory mapping is performed.
EW	Effective word – 32-bit contents of effective word location (EWL).
EWL	Effective word location – word location pointed to by effective virtual address of an instruction for a word operation.
FN	Floating normalize mode control—bit position 7 of PSWs. If not set, results of floating-point additions and subtractions are to be normalized; if set (=1), results are not normalized.
FR	Floating round mode control—bit position 4 of PSWs. If set (=1), basic processor rounds floating-point results. If not set, results are truncated.
FS	Floating significance mode control—bit position 5 of PSWs. If set (=1), basic processor traps to location X'44' when more than two hexadecimal places of postnormalization shifting are required for a floating-point addition or subtraction; if not set, no significance checking is performed.

Term	Meaning
FZ	Floating zero mode control—bit position 6 of the PSWs. If set (=1), basic processor traps to location X'44' when either characteristic underflow or zero result occurs for a floating-point multiplication or division; if not set, characteristic underflow and zero result are treated as normal conditions.
I	Instruction register—internal basic processor register that holds instructions obtained from memory while they are being decoded.
IA	Instruction address—17-bit value that defines virtual address of instruction immediately prior to the time that it is executed.
II	I/O interrupt group inhibit—bit position 38 of the PSWs. If set (=1), all interrupt levels within this group are inhibited.
L	Numeric value of bits 8-11 of decimal instruction word (value of 0 is 16 bytes).
MA	Mode altered—bit position 61 of PSWs. This bit is set (=1) during master-protected mode of operation and during real extended type of addressing.
MM	Memory map mode control—position 9 of PSWs. When set (=1), the memory map is in effect.
MS	Master/slave mode control—bit position 8 of PSWs. When set (=1), basic processor is in slave mode; when not set, basic processor may be in master or master-protected mode as determined by bit 40.
PSWs	Program status words—collection of separate registers and flip-flops treated as an internal basic processor register to store and display critical control information.
R	General register address value—4-bit contents of bit positions 8-11 (R field) of instruction word, also expressed symbolically as $(I)_{8-11}$. In instruction descriptions, register R is general register (of current register block) that corresponds to R field address value.
RA	Register altered—bit position 60 of PSWs. When trap occurs, this bit set (=1) when general register or memory location altered in execution of instruction causing the trap.
Ref. Add.	Reference address—contents of bit positions 15-31 of instruction word, a 17-bit field capable of directly addressing any general register in current register block (by using a value in range 0-15) or any word in main memory in address range 16 through 131,071. This address value is initial address value for any subsequent address computations, memory mapping, or both computation and mapping.
RP	Register pointer—bit positions 58 and 59 of PSWs; these bits select one of four possible register blocks.
Ru1	Odd register address value—register Ru1 is general register pointed to by value obtained by logically ORing 0001 into address for register R. Thus, if R field of instruction contains even value, Ru1 = R + 1 and if R field contains odd value, Ru1 = R.
SA	Source address—in byte-string instructions, contents of specified R register.
SBS	Source byte string—operand specified by byte string instruction.
SE	Sign extension—some instructions operate on two operands of different lengths; they are made equal in length by extending sign of shorter operand by required number of bit positions. For positive operands, result of sign extension is high-order 0's prefixed to the operand; for negative operands, high-order 1's are prefixed to operand. Sign extension process is performed after operand accessed from memory and before operation called for by instruction code is performed.
SPD	Stack pointer doubleword—contains the context (TSA, space count, word count, and TS, TW inhibit bits) of the push-down instructions.
TCC	Trap condition code—4-bit value (bit positions labeled TCC1, TCC2, TCC3, and TCC4), established as part of trap operations.
TS	Trap-on-space inhibit bit—conditions push-down stack limit trap for impending overflow or underflow of space count.
TSA	Top-of-stack address—pointer that points to highest-numbered address of operand stack in push-down instructions.
TW	Trap-on-word inhibit bit—conditions push-down stack limit trap for impending overflow or underflow of word count.

Term	Meaning	Term	Meaning
WK	Write key – bit positions 32, 33, 34, and 35 of PSWs; they are evaluated by the memory write-protect feature to determine accessibility of real memory by current program.	X (cont.)	if X ≠ 0, indexing is performed (after indirect addressing if indirect addressing is called for) with general register X in current register block.
X	Index register address value – 3-bit contents of bit positions 12-14 (X field) of instruction word. In instruction word, if X = 0, no indexing is performed;	X'n'	Hexadecimal qualifier – hexadecimal value (n) is unsigned string of hexadecimal digits (0 through 9 and A through F) surrounded by single quotation marks and preceded by the qualifier "X" (for example, $7B0_{16}$ is written X'7B0'.

APPENDIX C. FAULT STATUS REGISTERS

Table C-1. Fault Status Registers

Bit Position		Status Registers – Faults Detected By:					
		Basic Processor	MIOP	RMP	MI	PI	System Control Processor
0	16	PFI	PFI	PFI	PFI	PFI	PFI
1	17	General register parity error	Bus Check Fault (BCF)	BCF	Map or access-protect register parity error	Cluster bus parity error	Parity error on processor bus
2	18	Control register parity error	Control Check Fault (CCF)	CCF	Cluster bus parity error	Processor bus parity error	Operation code error
3	19	Internal basic processor bus parity error	Control Memory Fault (CMF)	CMF	Reserved	Unrecognized operation code	Reserved
4	20	Cluster bus parity error	CMF I/O adapter	ECE	Reserved	Reserved	Reserved
5	21	Processor-Detected Fault flag (PDF)	MIE	MIE	Cluster bus sequence check fault	Reserved	Reserved
6	22	Memory parity error	Data/order indicator†	Order type†	Reserved	Reserved	Reserved
7	23	Memory Interface Error (MIE)	Out indicator†	Order type†	Reserved	Multiple error	Reserved
8	24	Processor interface sequence check fault	Control Memory Fault (CMF) address bit 0	Reserved	Reserved	Control Memory Fault (CMF) address bit 0	Reserved
9	25	Extended arithmetic sequence check fault	CMF address bit 1	Reserved	Reserved	CMF address bit 1	Reserved
10	26	Basic processor sequence check fault	CMF address bit 2	Reserved	Reserved	CMF address bit 2	Reserved
11	27	Successful instruction retry	CMF address bit 3	Reserved	Reserved	CMF address bit 3	Reserved
12	28	Control memory parity error (BPE module)	CMF address bit 4	Reserved	Reserved	CMF address bit 4	Reserved
13	29	Control memory parity error (BPF module)	CMF address bit 5	Reserved	Reserved	CMF address bit 5	Reserved
14	30	Control memory parity error (BPG module)	CMF address bit 6	Reserved	Reserved	CMF address bit 6	Reserved
15	31	Control memory parity error (BPH module)	CMF address bit 7	Reserved	Reserved	CMF address bit 7	Reserved

†This is a 2-bit code indicating type of service call, as follows:

Bits 6	Bits 7	MIOP Significance	RMP Significance
0	0	Data In	Sense
0	1	Data Out	Write
1	0	Order In	Read
1	1	Order Out	Control

Table C-2. Memory Unit Status Register

Bit Position	Faults Detected by Memory Unit:
0-21	Fault address snapshot
22	Reserved
23	Memory unit parity error
24	Storage module selection error
25	Address In parity error
26	Data In parity error
27	Write-lock memory storage parity error
28	Port selection error
29	Operation mode undefined
30	Control sequence check fault error
31	Multiple error

XEROX

Publication Revision Sheet

JANUARY, 1974

CORRECTIONS TO XEROX 560 COMPUTER REFERENCE MANUAL

PUBLICATION NO. 90 30 76A, JANUARY, 1974

Page 9 should be replaced with the page attached to this revision sheet. Page 10 is a backup page with no change. The revision bar in the margin of the page indicates that it is corrected information.

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication.

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

☐ Learning ☐ Installing ☐ Sales
☐ Reference ☐ Maintaining ☐ Operating

Is the material presented effectively?

☐ Fully Covered ☐ Well Illustrated ☐ Well Organized ☐ Clear

What is your overall rating of this publication?

☐ Very Good ☐ Fair ☐ Very Poor
☐ Good ☐ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, Please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your Name & Return Address

2190(12/72)

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

Staple

Staple

Fold

First Class
Permit No. 229
El Segundo,
California

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

Postage will be paid by

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

Attn: Programming Publications

Fold

XEROX

701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511