Xerox SIGMA 7 Computer

Reference Manual

90 09 50J

XEROX

Xerox SIGMA 7 Computer

Reference Manual

90 09 50J

October 1973

XG46, Rev. 0
File No.: 1X03

Printed in U.S.A.

REVISION

This publication is a revision of the Xerox SIGMA 7 Computer Reference Manual, 90 09 501. It incorporates Publication Revision Package 90 09 501-1(7/71), which was a revision of Appendix D, Instruction Timing, pages 126 through 132. The change in text from that of the previous manual is indicated by vertical lines in the margins of these pages.

RELATED PUBLICATIONS

Title	Publication No.
Xerox Sigma Glossary of Computer Terminology	90 09 57
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

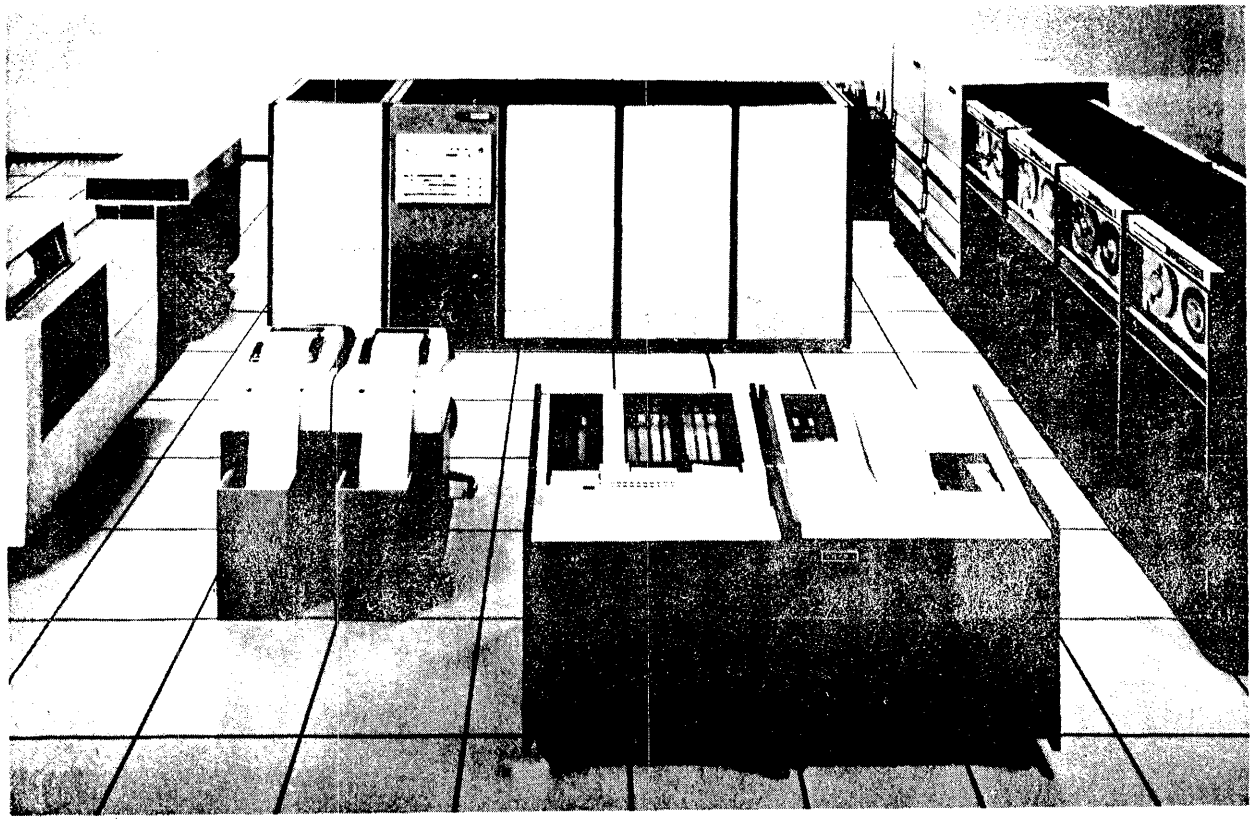
Manual Content Codes: BP – batch processing, LN – language, OPS – operations, RP – remote processing, RT – real-time, SM – system management, TS – time-sharing, UT – utilities.

CONTENTS

APPENDIXES

ILLUSTRATIONS

TABLES



SIGMA 7 Computer

1. SIGMA 7 SYSTEM

A typical SIGMA 7 system (see Figure 1) consists of the following major elements:

- A memory consisting of up to eight magnetic core storage units
- A central processing unit (CPU) that addresses core memory, fetches and stores information, performs arithmetic and logical operations, sequences and controls instruction execution, and controls the exchange of information between core memory and other elements of the system
- An input/output system controlled by one or more input/output processors (IOPs), each providing data transfer between core memory and peripheral input/output devices, and operating simultaneously with the CPU.

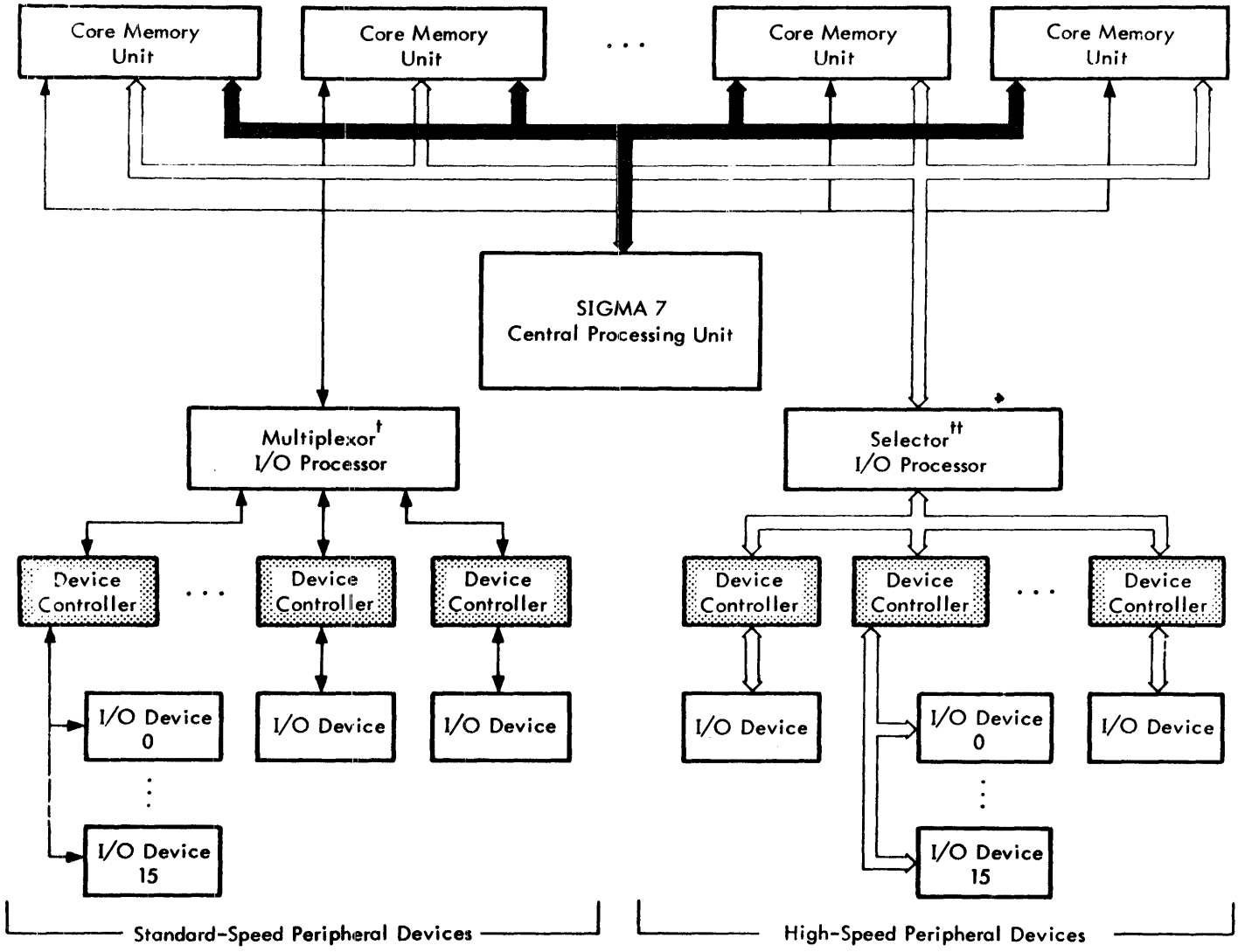


Figure 1. A Typical SIGMA 7 System

†Multiplexor IOP allows up to 24 devices (one per device controller) to operate simultaneously.

††Selector IOP allows one device at a time to operate at a high-speed transfer rate of up to one 32-bit word per microsecond. A selector IOP may service up to 32 high-speed devices, and two selector IOPs may share a single memory bus.

GENERAL CHARACTERISTICS

A SIGMA 7 computer system has features and operating characteristics that permit efficient functioning in real-time, general-purpose, time-sharing, and multi-usage computing environments:

- Word-oriented memory (32-bit word plus parity bit) which can be addressed and altered as byte (8-bit), halfword (2-byte), word (4-byte), and doubleword (8-byte) quantities.
- Full parity checking for both CPU/memory and input/output operations.
- Memory expandable from 8192 to 131,072 words (32,768 to 524,288 bytes) in increments of 3192 or 16,384 words.
- Direct addressing of the entire core memory, within the primary instruction word and without the need for base registers, indirect addressing, or indexing.
- Indirect addressing, with or without post-indexing.
- Displacement index registers, automatically self-adjusting for all data sizes.
- Immediate addressing of operands, for greater storage efficiency and increased speed.
- Sixteen general-purpose registers, expandable (in blocks of 16) to 512 to reduce transfer of data into and out of registers in a multiusage environment.
- Hardware memory mapping (optional), which obviates the problem of memory fragmentation and provides dynamic program relocation.
- Selective memory access protection with four modes (included with memory mapping) for system and information security and protection.
- Selective memory write protection (optional).
- Watchdog timer, assuring nonstop operation.
- Real-time priority interrupt system with automatic identification and priority assignment, fast response time, and up to 240 levels that can be individually armed, enabled, and triggered by program control.
- Interruptibility of long instructions, guaranteeing fast response to interrupts.
- Automatic traps, for error conditions and for simulation of optional instructions not physically implemented, all under program control.
- Power fail-safe, for automatic, safe shutdown in the event of a power failure.
- Multiple interval timers, with a choice of resolutions for independent time bases.
- Privileged instruction logic (master/slave modes), for concurrent, time-shared operation.
- Complete instruction set including:
  - Byte, halfword, word, and doubleword operations.
  - Use of all memory-referencing instructions for register-to-register operations, with or without indirect addressing and post-indexing, and within the normal instruction format.
  - Multiple register operations.
  - Fixed-point arithmetic operations in halfword, word, and doubleword modes.
  - Optional floating-point hardware operations, in short and long formats, with significance, zero, and normalization control and checking, all under 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 permitting up to 64 dynamically variable, user-defined instructions, and permitting a program to gain access to operating system functions without operating system intervention.
  - Optional decimal hardware operations, including arithmetic, edit, and pack/unpack.
  - 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.
  - An analyze instruction, for facilitating effective address computation.
  - An interpret instruction, for increased speed of interpretive programs.
  - Shift operations (left and right) or word or doubleword, including logical, circular, arithmetic, and floating-point modes.
- Independently operating input/output system with the following features:
  - Direct input/output of a full word, without the use of a channel.
  - Up to eight input/output processors (IOPs).

- Multiplexor input/output processors (MIOPs) for simultaneous operation of up to 24 standard-speed devices per IOP.
- Selector input/output processors (SIOPs) (8 or 32 bits wide) for data rates approaching 4 million bytes per second.
- Up to 32 device controllers can be connected to each SIOP.
- Both data and command chaining, for gather-read and scatter-write operations.
- Up to 32,000 output control signals and input test signals.

- Comprehensive complement of modular software:
  - Expands in capability and speed as system grows.
  - Basic system programming support: "Stand-Alone" Systems and Basic Control Monitor (BCM).
  - Operating systems: Real-time Batch Monitor (RBM), Batch Processing Monitor (BPM), Batch Time-Sharing Monitor (BTM), Universal Time-Sharing System (UTS), and Xerox Operating System (XOS). When larger computing capacity is required, UTS and XOS users can expand to the XDS SIGMA 9 Computer.
  - Language processors that include: FORTRAN IV-H, Extended XDS FORTRAN IV, XDS ANS COBOL, BASIC, FLAG, Symbol, Macro-Symbol, Meta-Symbol; also, utilities and applications software for both commercial and scientific users, e.g., Data Management System (DMS), Generalized Sort and Merge, Manage, 1401 Simulator, Functional Mathematical Programming System (FMPS), FMPS Matrix Generator/Report Writer (GAMMA 3), Simulation Language (SL-1), General Purpose Discrete Simulation package (GPDS), Graphic Display Library (GDL-1), Circuit Analysis Systems (CIRC-AC, CIRC-DC), etc.

- Standard and special-purpose peripheral equipment includes:
  - Rapid Access Data (RAD) files: Capacities to 6.2 million bytes per unit; transfer rates to 3 million bytes per second; average access times from 17 milliseconds.
  - Magnetic tape units: 7-track and 9-track systems, IBM-compatible; high-speed units operate at 150 inches per second with transfer rates up to 120,000 bytes per second; and other units operate at 37.5 inches per second with transfer rates up to 20,800 bytes per second and at 75 inches per second with transfer rates up to 60,000 bytes per second.
  - Displays: Graphic display has standard character generator, vector generator, and close-ups, as well as light pen and alphanumeric/function keyboard with a display rate of up to 100,000 characters per second.
  - Card equipment: Reading speeds of up to 1500 cards per minute; punching speeds of up to 300 cards per minute; intermixed binary and EBCDIC card codes.
  - Line printers: Fully buffered, with speeds of up to 1500 lines per minute; 132 print positions with 64 characters.
  - Keyboard/printers: Ten characters per second; also available with integral paper tape reader (20 characters per second) and punch (10 characters per second).
  - Paper tape equipment: Readers with speeds of up to 300 characters per second; punches with speeds of up to 120 characters per second.
  - Graph plotters: Digital incremental, providing drift-free plotting in two axes in up to 300 steps per second at speeds from 30 mm to 3 inches per second.
  - Data communications equipment: A complete line of character- and message-oriented equipment to connect remote user terminals to the computer system via common carrier lines and local terminals directly.

REAL-TIME FEATURES

Real-time applications are characterized by a need for hardware that provides quick response to an external environment, speed great enough to keep up with the real-time process itself, and sufficient input/output flexibility to handle a wide variety of data types at varying speeds. The SIGMA 7 system includes provisions for the following real-time computing features.

Multilevel, True Priority Interrupt System. The real-time oriented SIGMA 7 system provides for quick response to interrupts by means of up to 224 external interrupt levels. The source of each interrupt is automatically identified and responded to according to its priority. For further flexibility, each level can be individually disarmed (to discontinue accepting inputs to it) and disabled (to defer responding to it). Use of the disarm/disable feature makes programmed dynamic reassignment of priorities quick and easy, even while a real-time process is in progress. In establishing a configuration for the system, each group of 16 interrupt levels can have its priority assigned in different ways in order to meet the specific

needs of the problem; the way in which interrupt levels are programmed is not affected by the priority assignment.

Programs that deal with interrupts from specially designed equipment sometimes must be checked out before that equipment is actually available. To permit simulating this special equipment, any SIGMA 7 interrupt level can be triggered by the CPU itself through execution of a single instruction. This capability is also useful in establishing a 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 in order to respond to other critical interrupt levels. The interrupt routine can accomplish this by triggering a lower-priority level, which processes the remaining data only after other interrupts have been handled.

Certain instructions (READ DIRECT and WRITE DIRECT, described in Chapter 3) allow the program to completely interrogate 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 computer can sometimes become excessively delayed if the special device does not respond quickly. A built-in watchdog timer assures that the SIGMA 7 computer 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 – elapsed time since a given event, for example, or the current time of day. SIGMA 7 can contain two or four real-time clocks with varying degrees of resolution (1/60 second or 1/8 millisecond, for example) 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" costs in time. In SIGMA 7, each one of up to 32 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 a 64-bit program status doubleword (PSD). A single instruction stores the current PSD anywhere in memory and loads a new one from memory to establish a new environment, which includes information identifying a new block of general-purpose registers. A SIGMA 7 system can thus preserve and change its operating environment completely through the execution of a single instruction.

Simultaneous I/O Channel Operation. The use of a multiplexor input/output processor (MIOP) permits up to 24 channels with standard-speed devices to operate concurrently; the addition of more MIOPs increases this throughput.

High-Speed Channel Operation. The use of the selector input/output processor (SIOP) permits very high-speed data transfer – up to one 32-bit word per memory cycle. To meet special needs, data size can be 8 or 32 bits wide.

Memory Protection. Both foreground (real-time) and background programs can be run concurrently in a SIGMA 7 system because a foreground program is protected against destruction by an unchecked background program. Optional memory write-protection guarantees that protected areas of memory can be written into only under predefined conditions. Under operating system control, the optional memory access-protection feature also prevents accessing of memory for specified combinations of reading, writing, and instruction acquisition.

Variable Precision Arithmetic. Much data encountered in real-time systems are 16 bits or less. To permit this length of data to be processed efficiently, SIGMA 7 provides halfword arithmetic operations in addition to fullword operations. Doubleword arithmetic operations (for extended precision) are also included.

Direct Data Input/Output. For handling asynchronous I/O, a 32-bit word can be transferred directly to or from a general-purpose register, so that an I/O channel need not be occupied with relatively infrequent transmissions.

Interleave/Overlap. To increase processing speeds, memory banks overlap cycles automatically wherever possible. Core memory addresses can be interleaved modulo-2 or modulo-4 on a bank basis to increase the probability of overlapping.

GENERAL-PURPOSE FEATURES

General-purpose computing applications are characterized primarily by an emphasis on computation and internal data handling. Many operations are performed in floating-point format and on strings of characters. Other typical characteristics include decimal arithmetic operations, the need to convert binary numbers into decimal (for printing or display), and considerable input/output at standard speeds. The SIGMA 7 system includes the following general-purpose computer features.

Floating-Point Hardware (optional). Floating-point instructions are available in both short (32-bit) and long (64-bit) formats. Under program control, the user can select optional zero checking, normalization, and significance checking (which causes the computer to trap when a post operation shift of more than two hexadecimal places occurs in the fraction of a floating-point number). The significance checking feature permits the use of the short floating-point format (for high processing speed and storage economy) and the use of the long format when loss of significance is detected.

Decimal Arithmetic Hardware (optional). Decimal arithmetic instructions operate on up to 31 digits plus sign. This optional instruction set also 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 byte information with punctuation to display or print it).

Indirect Addressing. This feature provides for simple table linkages and permits the user to keep data sections of his program separate from procedure sections for ease of maintenance.

Displacement Indexing. The technique of indexing by means of a "floating" displacement permits the user to access the desired unit of data without the need to consider its size. The index registers automatically align themselves appropriately; thus, the same index register can be used on arrays with different data sizes. For example, in a matrix multiplication of any array of fullword, single-precision, fixed-point numbers, the results can 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 no matter which size of data element is used.

Powerful Instruction Set. The availability of more than 100 major instructions results in programs that are short, rapidly assembled, and quickly executed.

Translate Instruction. This instruction permits rapid translation between any two 8-bit codes (such as EBCDIC to ANSCII); thus data from a variety of input sources can be easily handled and reconverted 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. Four instructions permit handling up to 64 user-defined subroutines (as if they were built-in machine instructions) and gaining access to specified operating system services without requiring its intervention.

Interpret Instruction. This instruction simplifies and speeds interpretive operations such as compiling, thus reducing the space and time requirements for compilers.

Four-Bit Condition Code. This feature simplifies the checking of results 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.

TIME-SHARING FEATURES

Time-sharing is the ability of a computer system to share its resources among many users at the same time. Each user may perform a different task that requires a different share of the available resources and, in many instances, each may be on-line in an interactive ("conversational") mode with the computer. Other users may enter work to be batch processed. The SIGMA 7 system provides for the following time-sharing computer features.

Rapid Context Saving. When changing from one user to another, the operating environment can be switched quickly and easily. Stack-manipulating instructions permit from one to 16 general-purpose registers to be stored in a push-down stack by a single instruction—with automatic updating of stack status information—and to be retrieved (again, by a single instruction) when needed. The current program status doubleword (which contains the entire description of the current user's environment and mode of operation) can be stored anywhere in memory and a new program status doubleword loaded, all with a single instruction.

Multiple Register Blocks. The optional availability of up to 32 blocks of 16 general-purpose registers further improves response time by reducing the need to store and load register blocks. As needed, each user can be assigned a distinct block; the program status doubleword automatically points to the currently applicable register block.

User Protection. The slave mode of operation restricts each user to his own set of instructions while reserving to the operating system those instructions that could, if used incorrectly, destroy another user's program. An optional memory access-protection system prevents any user from accessing storage areas other than those assigned to him. This access protection permits the user to access certain areas for reading only, such as those containing public subroutines, while preventing him from reading, writing, or accessing instruction in areas set aside for other users.

Storage Management. SIGMA 7 memories are available in 16 sizes (from 8192 to 131,072 words) to provide the capacity needed, while assuring potential for expansion. To assure efficient use of available memory, the optional memory map hardware permits storing a user's program in fragments (as small as 512 words) wherever space is available; yet, all fragments appear as a single, contiguous block of storage at execution time. The memory map also automatically and dynamically handles 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 for the full-sized SIGMA 7 memory is provided as a single option no matter how small the actual memory may be. Thus, the system can always address a virtual memory of 131,072 words regardless of physical memory size.

Input/Output Capability. SIGMA 7 can control up to eight input/output processors (of two types) in various combinations. Each multiplexor I/O processor can have up to 24 standard-speed I/O channels operating simultaneously; selector I/O processors can have any one of up to 32 high-speed I/O devices operating on each processor. The I/O processors operate semi-independently of the central processor, leaving it free to provide faster response to overall system needs.

Nonstop Operation. A watchdog timer assures that the system continues to operate even if certain special I/O capabilities are used with special devices that can cause delays or halts if they fail. Multiple real-time clocks with varying resolutions permit establishing several independent time bases, thus allowing flexible allocation of time slices to each user.

MULTIUSAGE FEATURES

As implemented in the SIGMA 7 system, "multiusage" combines two or more computer application areas. The most difficult computing problems are associated with real-time applications. Similarly, the most difficult multiusage problems are associated with time-sharing applications that include one or more real-time processes. Because the SIGMA 7 system has been designed on a real-time base, it is especially qualified for a mixture of applications in a multiusage environment. Many of the hardware features that are required for specific application areas are equally useful in others, although in different ways. This multiple capability makes SIGMA 7 particularly effective for multiusage applications. The major SIGMA 7 multiusage computer features are:

Priority Interrupt. In a multiusage environment, many elements operate asynchronously. Thus, a true priority interrupt system is essential. It allows the computer system to respond quickly (and in proper order) to the many demands made on it, without the high overhead costs of complicated programming, lengthy execution time, and extensive storage allocations.

Quick Response. The many features that combine to produce a quick-response system – multiple register blocks, quick context saving, push-pull operations – benefit all users because more of the computer's resources are available for useful work.

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

Input/Output. Because of its wide range of capacities and speeds (with and without channels), the SIGMA 7 I/O system simultaneously satisfies the needs of many different application areas economically, both in terms of equipment and of programming.

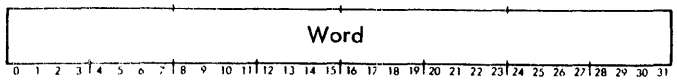
Instruction Set. The large SIGMA 7 instruction set provides the computational and data-handling capabilities required for widely differing application areas; therefore, each user's program length (thus running time) is decreased and the speed of obtaining results is increased.

2. SIGMA 7 SYSTEM ORGANIZATION

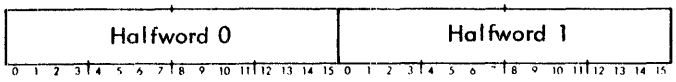
The primary elements in a basic SIGMA 7 system — a central processor, core memory, and input/output processor — are all designed around a central, double bus structure. Each primary element of the system operates asynchronously and semi-independently, automatically overlapping the operation of the other elements (when circumstances permit) for greater speed. The basic configuration can be expanded merely by increasing the number of core memory modules (up to eight), increasing the number of buses (up to six), increasing the number of input/output processors (up to eight), or by increasing the number of central processors.

INFORMATION FORMAT

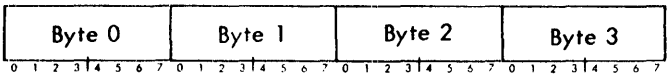
The basic element of SIGMA 7 information is a 32-bit word, in which the bit positions are numbered from 0 through 31, as follows:



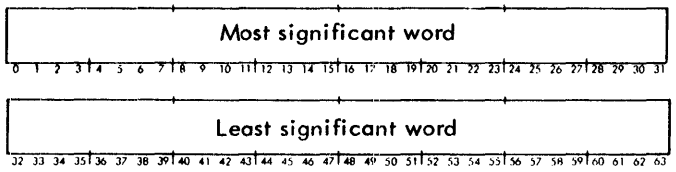
A SIGMA 7 word can be divided into two 16-bit parts (called halfwords) in which the bit positions are numbered from 0 through 15, as follows:



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



Two SIGMA 7 words can be combined to form a 64-bit element (called a doubleword) in which the bit positions are numbered from 0 through 63, as follows:



Four bits of information can be expressed by means of a single hexadecimal digit. Hexadecimal digits (and their binary and decimal equivalents) are expressed in the following notation:

Hexadecimal	Binary	Decimal
X'0'	0000	0
X'1'	0001	1
X'2'	0010	2
X'3'	0011	3
X'4'	0100	4
X'5'	0101	5
X'6'	0110	6
X'7'	0111	7
X'8'	1000	8
X'9'	1001	9
X'A'	1010	10
X'B'	1011	11
X'C'	1100	12
X'D'	1101	13
X'E'	1110	14
X'F'	1111	15

Thus, a byte can be expressed as a 2-digit hexadecimal number, a halfword as a 4-digit hexadecimal number, a word as an 8-digit hexadecimal number, and a doubleword as a 16-digit hexadecimal number. In this reference manual, a hexadecimal number is displayed as a string of hexadecimal digits enclosed by single quotation marks and preceded by the letter "X". For example, the binary number 01011010 is expressed hexadecimally as X'5A'.

CORE MEMORY

SIGMA 7 core memory systems use a 33-bit word (four 8-bit bytes, plus a parity bit) as the basic unit of information. All of memory is directly addressable by the CPU (except for memory locations 0 through 15) and by the IOPs. The SIGMA 7 addressing capability accommodates a maximum memory size of 131,072 words (524,288 bytes). Core memory is modular and is available in increments of 8192 words (32,768 bytes), or 16,384 words (65,536 bytes).

DEDICATED MEMORY LOCATIONS

Memory locations 0 through 319 are reserved by standard XDS software for special purposes as shown in Table 1.

INFORMATION BOUNDARIES

SIGMA 7 instructions assume that bytes, halfwords, and doublewords are located in storage according to the following boundary conventions:

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

The various information boundaries are illustrated in Figure 2.

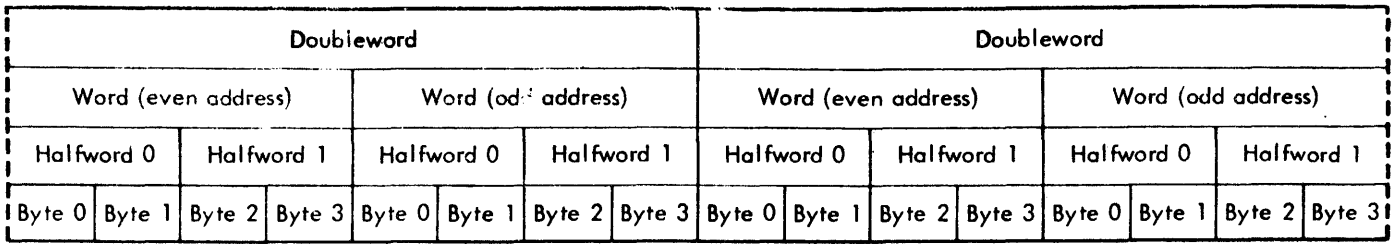


Figure 2. Information Boundaries

Table 1. SIGMA 7 Dedicated Memory Locations

Location		Function
Decimal	Hexadecimal	
0 : 15	0 : F	Addresses of general registers
16 : 31	10 : 1F	Reserved for future use
32 33	20 21	CPU/IOP communication
34 : 41	22 : 29	Program stored by LOAD switch on the processor panel
42 : 63	2A : 3F	First record read from peripheral device during a load operation
64 : 79	40 : 4F	Traps
80 : 87	50 : 57	Override interrupt levels
88 : 91	58 : 5B	Counter interrupt levels
92 93	5C 5D	Input/output interrupt levels
94 95	5E 5F	Reserved for future use
96 : 319	60 : 13F	External interrupt levels

COMPUTER MODES

The SIGMA 7 computer operates in either the master mode or the slave mode. The mode of operation is determined by the state of the master/slave mode control bit in the arithmetic and control unit.

MASTER MODE

The master mode is the basic operating mode of the computer. In this mode, all SIGMA 7 instructions are permissible. It is assumed that there is a resident executive program (operating in the master mode) that controls and supports the other programs operating in the master or slave mode.

SLAVE MODE

The slave mode is the problem-solving mode of the computer. In this mode, "privileged" instructions are prohibited. Privileged instructions are those relating to input/output and to changes in the basic control state of the computer. All privileged instructions are performed in the master mode only. Any attempt by a program to execute a privileged instruction while the computer is in the slave mode results in a return of control to the resident executive program.

The master/slave mode control bit can be changed only when the computer is in the master mode; thus, a slave program cannot directly change the computer mode from slave to master. However, the slave 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 executive program.

CPU FAST MEMORY

Several high-speed integrated circuit memories may be used in the SIGMA 7 CPU. These memories are capable of delivering information to (or receiving information from) the arithmetic and control unit simultaneously with the operation of core memory. These memories are not accessible to any other unit in a SIGMA 7 system.

CENTRAL PROCESSING UNIT

This section describes the organization and operation of the the SIGMA 7 central processing unit in terms of information processing and program control, instruction and data formats, indirect addressing and indexing, memory mapping and protection, overflow and trap conditions, and interrupt control. Basically, the SIGMA 7 CPU consists of a fast memory and an arithmetic and control unit (see Figure 3).

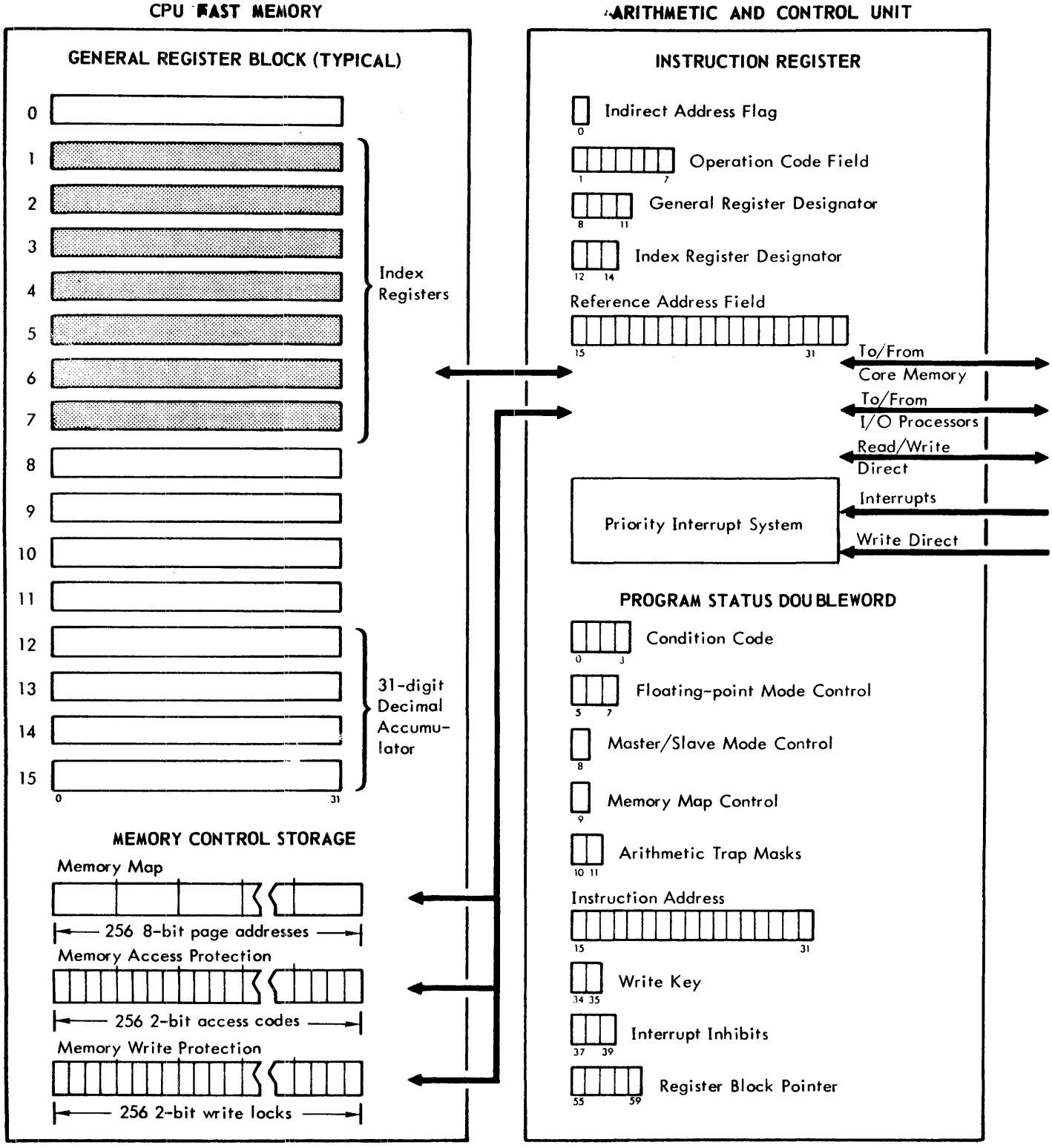


Figure 3. SIGMA 7 Central Processing Unit

GENERAL REGISTERS AND REGISTER BLOCK POINTER

A high-speed memory, consisting of sixteen 32-bit words, is contained within the basic SIGMA 7 CPU for general-purpose register usage; these 16 words of fast memory are referred to as a register block. A SIGMA 7 system may contain up to 32 such register blocks, and a 5-bit control field (called the register block pointer) in the arithmetic and control unit selects the block currently available to a program. The 16 general registers selected by the register block pointer are referred to as the current register block. The register block pointer can be changed only when the computer is in the master mode; thus, a slave program cannot change the register block pointer.

Each general register in a current register block is identified by a 4-bit code in the range 0000 through 1111 (0 through 15 in decimal, or X'0' through X'F' in hexadecimal notation). Any general register can be used as a fixed-point accumulator, floating-point accumulator, temporary storage, or can contain control information such as a data address, count, pointer, etc. Any (or all) of general registers 1 through 7 can be used as index registers. Registers 12 through 15 are used as a decimal accumulator that is capable of containing 31 decimal digits plus sign. The use of registers 12 through 15 is automatic when a decimal instruction is executed; however, these registers may be used for other purposes by instructions not in the decimal instruction set.

MEMORY CONTROL STORAGE

Three optional, high-speed integrated-circuit memories are available for storage of a memory map, a set of memory access protection codes, and a set of memory write-protection codes, all of which can be changed only when the computer is in the master mode.

Memory Map and Access Protection

The optional memory map feature includes high-speed memories for both the memory map and the access-protection codes. When the memory map is implemented in a SIGMA 7 computer, use of the map is determined by the state of the memory map control bit in the arithmetic and control unit.

Memory Map. Two terms are essential to a proper understanding of the memory mapping concept: virtual address and actual address.

A virtual address is a value used by a machine-level program to designate the location of an instruction, the location of an element of data, the location of a data address (indirect address), or to designate an explicit quantity, such as a count. 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. Thus, virtual addresses include all instruction addresses, data addresses, indirect addresses, and addresses used as counts within a stored program, as well as those addresses computed by the program.

An actual address is a value used by the CPU to access memory for storage or retrieval of information, as required by the execution sequence of an instruction. Thus, actual addresses designate wired-in hardware storage locations.

When the memory map is not implemented in a SIGMA 7 computer (or when the map is implemented but not in use, as determined by the memory map control bit), all virtual address values above 15 are used by the CPU as actual addresses. Virtual addresses in the range 0 through 15 are always used by the CPU as general register addresses rather than as core memory addresses. Thus, for example, if an instruction uses a virtual address of 5 as the address where a result is to be stored, the result is stored in general register 5 in the current register block instead of in core memory location 5.

When the computer is operating with the memory map (i.e., the memory map is implemented and in effect), virtual addresses in the range 0 through 15 are still used as general register addresses. However, all 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 replacement process is described in the section "Memory Address Control".

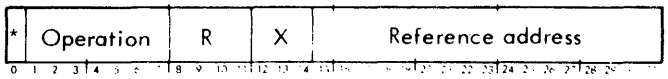
Memory Access Protection. When the computer is operating in the slave mode with the memory map, the access-protection codes determine whether or not the program may access instructions from, read from, or write into specific regions of the virtual address continuum (virtual memory). If the slave program attempts to access a region of virtual memory that is so protected, program control is returned to the executive program. (The access-protection codes are described in the section "Memory Address Control".)

Memory Write Protection

The optional memory write-protection feature operates independently of the memory map and access protection. The memory write-protection option includes the high-speed memory for the memory write locks. These locks operate in conjunction with a 2-bit field, called the write key, in the arithmetic and control unit. The locks and the key determine whether or not the program (slave or master) may alter the contents of specific regions of core memory as accessed by actual addresses. The write key can be changed only when the computer is in the master mode; thus the current write key cannot be changed by a slave program. (The functions of the locks and key are described in the section "Memory Address Control".)

INSTRUCTION FORMAT

The normal SIGMA 7 memory-addressing instruction has the following format:



* This bit position indicates whether or not indirect addressing is to be performed. Indirect addressing is performed (one level only) if this

bit position contains a 1, and is not performed if this bit position contains a 0.

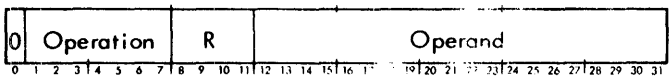
Operation — This 7-bit field contains the code that designates the operation to be performed.

R — This 4-bit field designates any of the 16 registers of the current register block as an operand source, result destination, or both.

X — This 3-bit field designates any one of registers 1-7 of the current register block as an index register. X = 0 designates no indexing; hence, register 0 cannot be used as an index register.

Reference address — This 17-bit field contains the initial virtual address of the instruction operand. Although the contents of this field is always, in itself, a word address, the reference address field allows any word, doubleword, left halfword, or leftmost byte within a word in memory to be directly addressed. Halfword and byte operations require additional address bits for halfwords and bytes that do not begin on a word boundary. Thus, to address the second halfword of a word, the X field of the instruction must designate a register that contains a 1 in its low-order bit position. To address bytes 1, 2, or 3 of a word, the X field of the instruction must designate a register that contains 01, 10, or 11, respectively, in its two low-order bit positions. See "Indexing and Index Registers" for a more complete description of the SIGMA 7 indexing process.

Some SIGMA 7 instructions are of the immediate-addressing type. The format of these instructions provides for an operand within the instruction word itself, as shown below. The functions of the Operation and R fields are identical to those of the normal instruction format.



0 — This bit position is shown coded with a 0 because indirect addressing cannot be used with this type of instruction. If indirect addressing is attempted, the computer treats the instruction as a nonexistent instruction.

Operand — This field contains an operand that is 20 bits in length, with negative numbers represented in two's-complement form.

There are several methods by which an instruction word may specify the source of an operand or the destination of a result. These methods are explained below.

IMMEDIATE OPERAND

The operation code of an immediate operand instruction specifies that an operand is to be found in the operand field (bit positions 12-31) of the instruction word itself, and not in a general register or core memory location. The operand field of this type of instruction cannot be modified by indexing. The following SIGMA 7 instructions are of the immediate operand type:

Instruction Name	Mnemonic
Load Immediate	LI
Load Conditions and Floating Control Immediate	LCFI
Add Immediate	AI
Multiply Immediate	MI
Compare Immediate	CI

The byte string instructions (see page 58) are similar to those of the immediate operand type in that they cannot be modified by indexing. However, the operand field of these instructions contains a byte address displacement (or a byte address) that is a virtual address subject to modification by the memory map. If an immediate or byte string instruction is indirectly addressed, it is treated as a nonexistent instruction by the computer.

MEMORY REFERENCE ADDRESSES

Core memory locations 0 through 15 are not accessible to the programmer because memory addresses 0 through 15 are reserved as register designators for "register-to-register" operations. Thus, an instruction can treat any register of the current register block as if it were a location in core memory. Furthermore, the register block can be used to hold an instruction (or a series of up to 16 instructions) for execution just as if the instruction (or instructions) were in core memory. The only restriction upon the use of the register block for instruction storage is:

> If an instruction accessed from a general register uses the R field of the instruction word to designate the next higher-numbered register and execution of the instruction would alter the contents of the register so designated, the contents of that register should not be used as the next instruction in sequence because the operation of the instruction in the affected register would be unpredictable.

In the maximum core memory configuration (131,072 words), memory addresses "wrap around" with address 0 (general register 0) being the next consecutive memory address after X'1FFFF'(131,071). Core memory location 16 follows general register 15 as the next location in ascending sequence.

Direct Reference Address. If neither indirect addressing nor indexing is called for by the instruction, the reference address field of the instruction is a direct reference address.

Indirect Reference Address. If indirect addressing is called for by the instruction (a 1 in bit position 0 of the instruction word), the reference address field is used to access a word location that contains the direct reference address in bit

positions 15-31. The direct reference address then replaces the indirect reference address. Indirect addressing is limited to one level.

Index Reference Address. If indexing is called for by the instruction (a nonzero value in bit positions 12-14 of the instruction), the direct 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 address of the instruction. If indirect addressing and indexing are both called for in an instruction, the index displacement is not used to modify the indirect reference address, but is used to modify the direct reference address obtained from the location pointed to by the indirect reference address. This method of indexing after indirect addressing is called post-indexing.

Register Address. If any instruction produces a virtual address that is a memory reference (i.e., a direct, indirect or indexed reference address) in the range 0 through 15, the CPU does not attempt to read from or write into core memory. Instead, the 4 low-order bits of the reference address are used as a general register address, and the general register (of the current register block) corresponding to this address is used as the operand location or result destination. Thus, the instruction can use any register 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 referred to as a "register-to-register" operation.

Actual Address. An actual address is the address value actually used by the CPU to access core memory. If the memory map option is not implemented, or if the computer is not operating with the memory map, all virtual addresses above 15 automatically become actual address. However, if the computer is operating with the memory map feature, all virtual address above 15 are transformed (usually into alternate addresses in a different memory page) by the memory map, and these then become actual addresses. Virtual addresses below 16 are never transformed by the memory map and thus always refer to a general register for a register-to-register operation.

Effective Address. The effective address is defined as the final virtual address computed for an instruction. The effective address is usually used as the virtual address of an operand location or result destination. However, 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).

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

Effective Operand. An effective operand is defined to be the contents of an actual location (in core 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 definition of an effective operand also presupposes the transformation of virtual address into actual addresses.

Address Modification

Indirect Addressing. The 7-bit operation code field of the SIGMA 7 instruction word provides for up to 128 instructions, nearly all of which can use indirect addressing (the exceptions, already mentioned, are the immediate and byte string instructions). The indirect addressing operation, as called for by the indirect address bit (bit position 0) of the instruction word, is limited to one level. Indirect addressing does not proceed to further levels, 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 reference address field of the instruction is used to obtain a word, and the 17 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.

Indexing and Index Registers. The X field of the normal instruction word permits any one of registers 1 through 7 in the current register block to be designated as an index register. The contents of this register are then treated as a displacement value.

Figure 4 shows how the indexing operation takes place. As the instruction is brought from memory, it is loaded into a 34-bit instruction register that initially contains 0's 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 addressing type of the instruction. That is; if it is a byte operation, the displacement is lined up so that its low-order bit is aligned with the least significant bit of the 34-bit instruction register. The displacement is 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, which is referred to as the effective address of the instruction. High-order bits of the 32-bit displacement field 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). However, the displacement value 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's (i.e., if the displacement is a negative integer in two's complement form).

The effective address of an instruction is always a 19-bit byte address value; however, this value is automatically adjusted

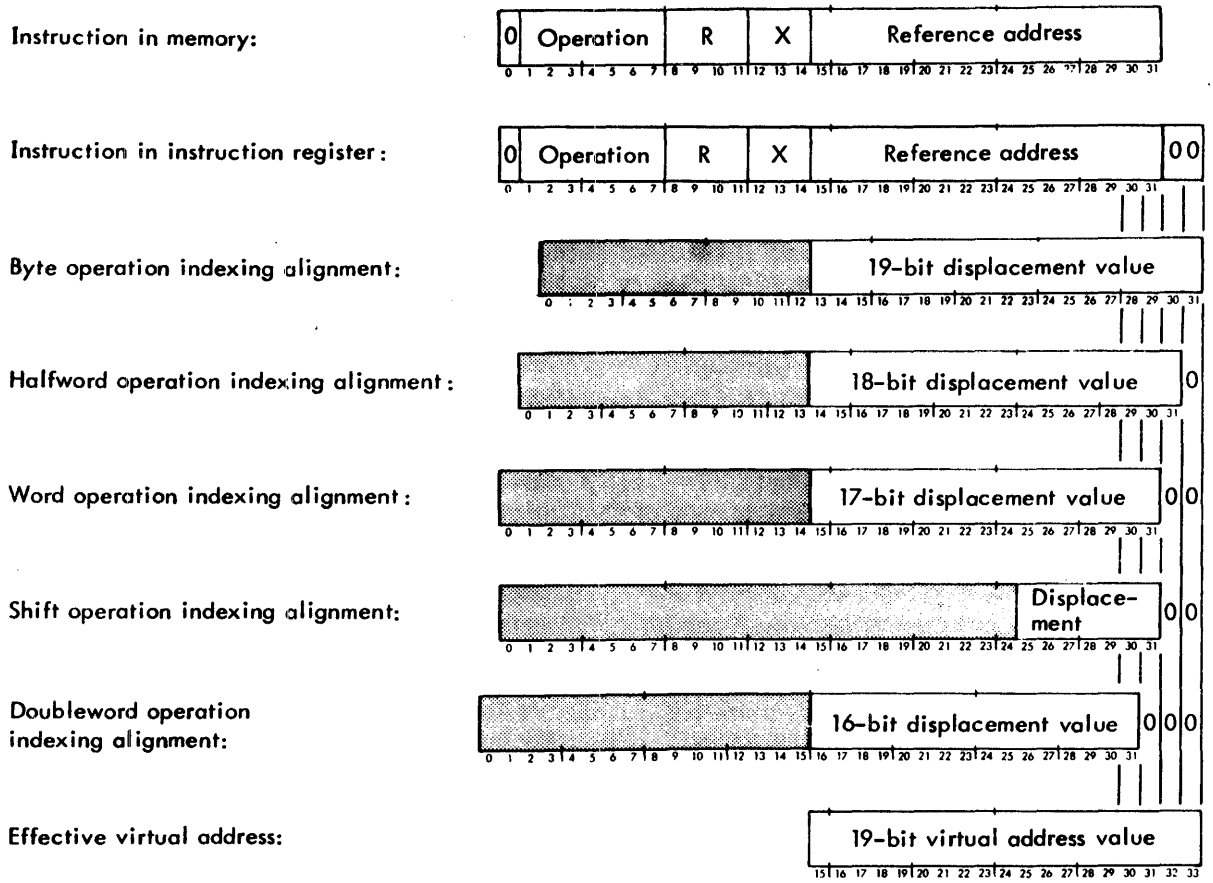


Figure 4. Index Displacement Alignment

to the SIGMA 7 information boundary conventions. Thus, for halfword operations, the low-order bit of the effective halfword address is 0; for word operations, the two low-order bits of the effective word address are 0's; and for doubleword operations, the 3 low-order bits of the effective doubleword address are 0's.

If no indexing is used with a byte operation, the effective byte is the first byte (bit positions 0-7) of a word location; if no indexing is used with a halfword operation, the effective halfword is the first halfword (bit positions 0-15) of a word location. A doubleword operation always involves a word at an even-numbered word address and the word at the next sequential (odd-numbered) word address. 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 0. Thus, an odd-numbered word address (referring to the middle of a doubleword) designates the same doubleword as an even-numbered word address, when used for a doubleword operation.

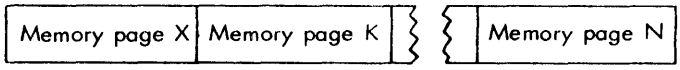
MEMORY ADDRESS CONTROL

With a SIGMA 7 computer, two optional methods are available for controlling the use of core memory by a program; they are the memory map and the memory write locks. The memory map provides for dynamic relocatability of programs and for access protection through inhibitions imposed on slave mode programs. The memory write locks provide memory write protection for both master and slave mode programs.

MEMORY MAP AND ACCESS PROTECTION

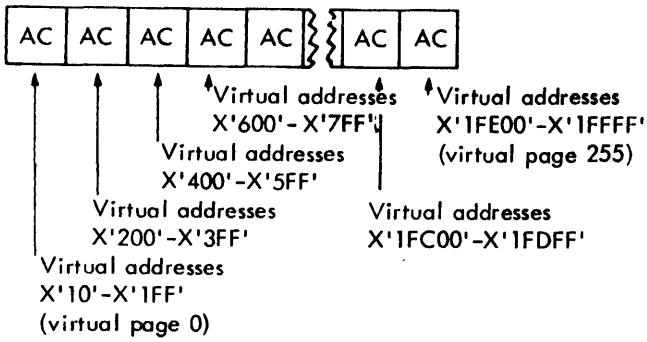
The memory map can be represented as a series of 256 8-bit registers, each of which contains an 8-bit actual memory page address code for a specific 512-word page of virtual addresses, and a series of 256 2-bit registers, each of which contains a 2-bit access control code for a specific 512-word page of virtual addresses. (The access control codes are applicable only to programs operating in the slave mode with the memory map.)

The memory page address codes are assigned to pages of virtual addresses as follows:



Virtual addresses	Virtual addresses	Virtual addresses
X'10'-X'1FF'	X'200'-X'3FF'	X'1FE00'-X'1FFFF'
(virtual page 0)	(virtual page 1)	(virtual page 255)

The access control codes are assigned as follows:



The memory page addresses and access control codes can be changed only by the privileged instruction MOVE TO MEMORY CONTROL (see "Control Instructions").

When the CPU is operating in the mapping mode, 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 core memory; instead, the 4 low-order bits of the virtual address comprise a general register address. However, if an instruction produces a virtual address greater than 15, the 8 high-order bits of the virtual address are used to obtain the appropriate memory page address and 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 and access control codes are used; and so on, through the 256th page address and 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, the CPU performs a test to determine whether or not there are any inhibitions on using the virtual address by a slave program. (If the CPU is in the master mode, this test is not performed.) The 2-bit access control code is interpreted as follows:

AC	Function
00	The slave program can write into, read from, or access instructions from this page of virtual addresses.
01	The slave program cannot write into, but can read from or access instructions from this page of virtual addresses.
10	The slave program cannot write into or access instructions from, but can read from this page of virtual addresses.
11	The slave program is denied any access to this page of virtual addresses.

If the instruction being executed by the slave program fails this test, the instruction execution is aborted and the computer traps to location X'40', the "nonallowed operation" trap (see "Trap System").

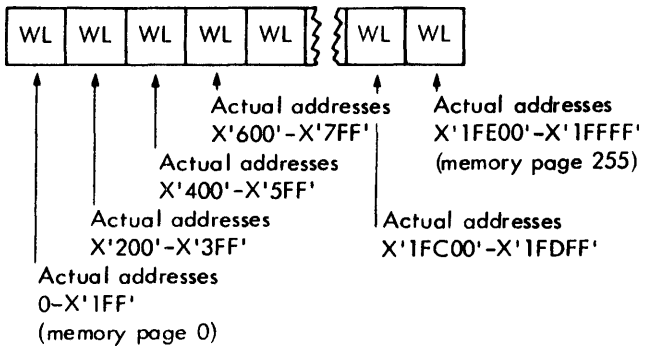
If the instruction being executed by the slave program passes this test (or the CPU is in the master mode), the page address bits in the accessed byte of the memory map replace the 8 high-order bits of the virtual address, to produce the actual address of the core memory location to be used by the instruction.

If the page address bits in the accessed byte of the memory map are all 0's, and when combined with 9 low-order bits of the virtual address, an actual address is produced that corresponds to a word address in the range 0 through 15, 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 actual core memory locations rather than general registers.

Figure 5 illustrates the address modification and mapping process for an indirectly addressed, indexed, halfword operation. As the figure shows, word address 1 is the contents of the reference address field in the instruction stored in memory. The instruction is brought into the instruction register, and word address 1 (assumed to be greater than 15) is converted from a virtual address to an actual address by the memory map. The 17 low-order bits of the core memory location pointed to by word address 1, labeled word address 2, then replaces word address 1 in the instruction register. The index register designated in the X field of the instruction is then aligned for incrementing at the halfword-address level, the final virtual (effective) address is formed, and the effective address (assumed to be greater than 15) is also transformed, through the memory map. The final 19-bit core memory address, which automatically contains a low-order 0, is then used to access the halfword to be used as an operand for the instruction.

MEMORY WRITE LOCKS

The access control bits in the memory map provide access protection, through inhibitions imposed on slave programs. However, this protection is only available when the memory map is in effect, and is only operative with respect to slave programs. An optional memory protection feature, independent of the map option, is provided by a lock and key technique. A 2-bit write-protect lock (WL) is provided for each 512-word page of actual core memory addresses. The write-protect locks consist of 256 2-bit write locks, each assigned to a 512-word page of actual addresses as follows:



The write-protect locks can be changed only by the execution of the privileged instruction MOVE TO MEMORY CONTROL (see Control Instructions).

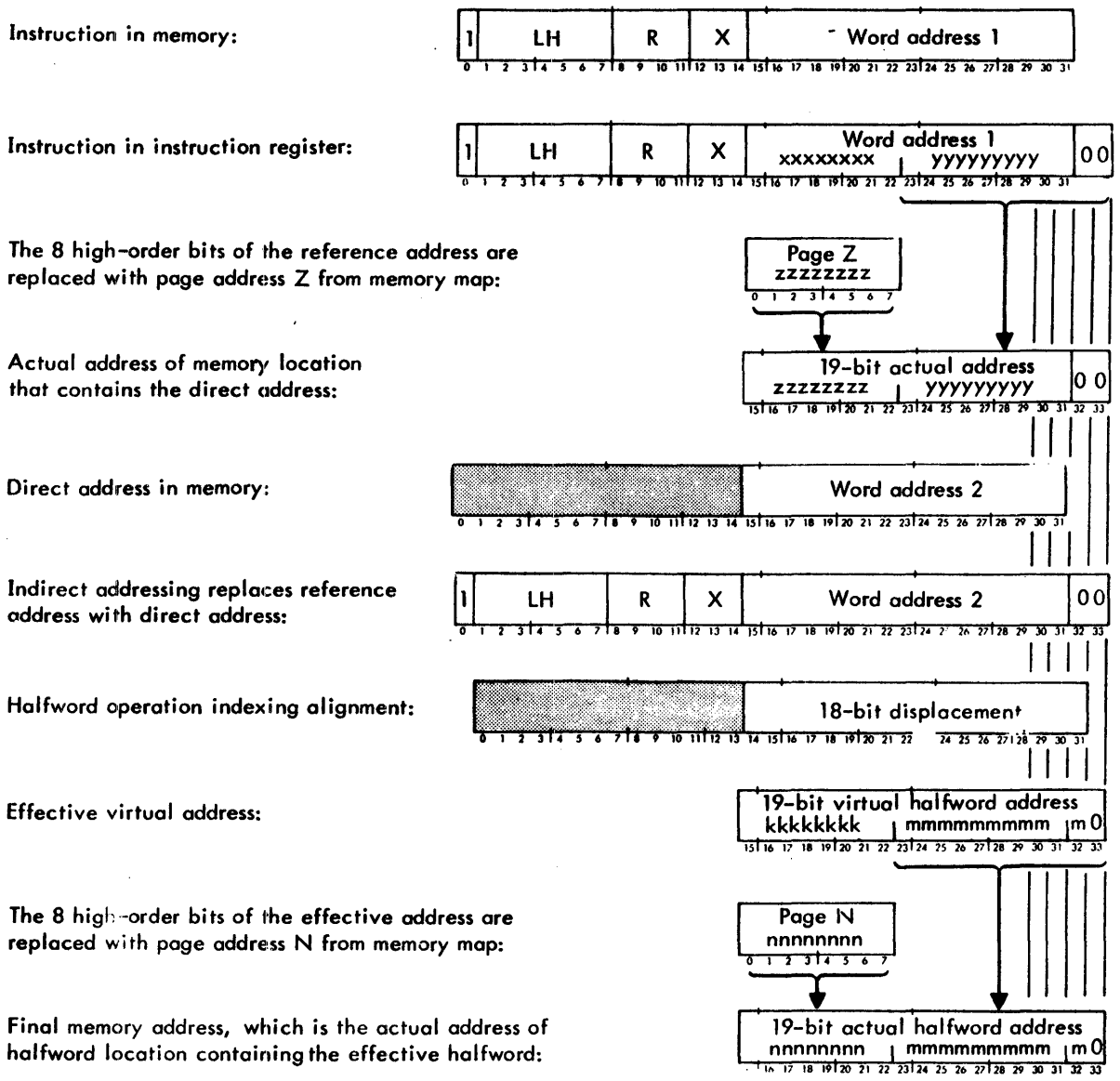


Figure 5. Generation of Actual Memory Addresses

The write-key (a 2-bit field in the arithmetic and control unit) works in conjunction with the lock storage to determine whether or not the program (whether slave or master) can write into a specific page of core memory locations. The keys and locks control access for writing, according to the following rules:

A lock value of 00 means that the corresponding memory page is "unlocked"; write access to that page is permitted independent of the key value.

A key value of 00 is a "skeleton" key that will open any lock; thus, write access to any memory page is permitted independent of its lock value.

A lock value other than 00 for a memory page permits write access to that page only if the key value is identical to the lock value.

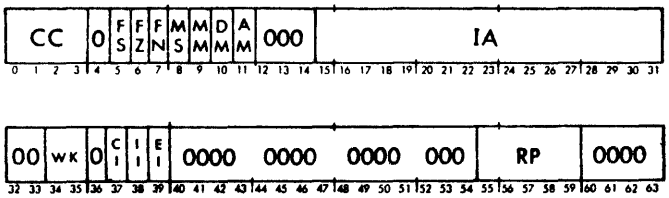
Thus, a program can write into a given memory page if the lock value is 00, if the key value is 00, or if the key value matches the lock value.

Note that the memory access protection feature is provided with the map option and operates on virtual addresses, whereas the memory write protection feature is a separate option that operates on actual memory addresses. Thus, if the access protection feature is invoked (that is, the CPU is in the slave mode and is using the memory map), the access protection codes are examined at the time the virtual address is converted into an actual address. Then, the locks and keys are examined to determine whether or not the program (master or slave) is allowed to alter the contents of the core memory location corresponding to the final actual address. If an instruction attempts to write into a write-protected memory page, the computer aborts

the instruction, and traps to location X'40', which is the "nonallowed operation" trap (see Trap System).

PROGRAM STATUS DOUBLEWORD

The critical control conditions of the SIGMA 7 CPU can be defined within 64 bits of information. These 64 bits are collectively referred to as the current program status doubleword (PSD). The current PSD can be considered as a 64-bit internal CPU register, although it actually exists as a collection of separate registers and flip-flops. When stored in memory, the PSD is always in 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 on the particular instruction just executed. After an instruction is executed, the instructions BRANCH ON CONDITIONS SET (BCS) and BRANCH ON CONDITIONS RESET (BCR) 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, CC3 to the third bit, and CC4 to the fourth bit. Any program (slave or master mode) can change the current value of the condition code by executing either the instruction LOAD CONDITIONS AND FLOATING CONTROL IMMEDIATE (LCFI) or the instruction LOAD CONDITIONS AND FLOATING CONTROL (LCF); any program can store the current condition code by executing STORE CONDITIONS AND FLOATING CONTROL (STCF). These instructions are described in Chapter 3, "Load/Store Instructions".
FS	Floating significance mode control
FZ	Floating zero mode control
FN	Floating normalize mode control The three floating-point mode bits (FS, FZ, and FN) control the operation of the computer with respect to floating-point significance checking, the generation of zero results, and the normalization of the results of floating-point additions and subtractions, respectively. (The floating-point mode controls are described in Chapter 3, "Floating-point Instructions".) Any program (slave or master) can change the state of the current floating-point mode controls by executing either the instruction LCFI or the instruction LCF; any program can store the current state of the current floating-point mode controls by executing the instruction STCF.
MS	Master/slave mode control. The computer is in the master mode when this bit is a 0; it is in the slave mode when this bit is a 1. The master/slave mode control cannot directly be changed by a slave program; however, a master mode program can change the control by executing either the instruction LOAD PROGRAM STATUS DOUBLEWORD (LPSD) or the instruction EXCHANGE PROGRAM STATUS DOUBLEWORD (XPSD). These two privileged instructions are described in Chapter 3, "Control Instructions".
MM	Memory map control. The optional memory map (if implemented) is in effect when this bit is a 1; it is not in effect when this bit is 0. The memory map control cannot be changed by a slave program. A master mode program can change the memory map control by executing either the instruction LPSD or the instruction XPSD.
DM	Decimal mask. The decimal arithmetic trap (see "Trap System") is in effect when this bit is a 1; the trap is not in effect when this bit is a 0. The conditions that can cause a decimal arithmetic trap are described in Chapter 3, "Decimal Instructions". The decimal trap mask cannot be changed by a slave program; a master mode program can change the mask by executing either the instruction LPSD or the instruction XPSD.
AM	Arithmetic mask. The fixed-point arithmetic overflow trap is in effect when this bit is a 1; the trap is not in effect when this bit is a 0. The instructions that can cause fixed-point overflow are described in the section "Trap System". The arithmetic trap mask cannot be changed by a slave program; a master mode program can change the mask by executing either the instruction LPSD or the instruction XPSD.
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 2-bit key used in conjunction with the optional memory protection feature. A slave program cannot change the current write key; a master mode program can change the write key by executing either the instruction LPSD or the instruction XPSD.

Designation	Function
CI	Counter interrupt group inhibit.
II	Input/output interrupt group inhibit.
EI	External interrupt group inhibit.

The three inhibit bits (CI, II, and EI) determine whether an interrupt can occur. The functions of the interrupt inhibits are described in the section "Interrupt System". A slave program cannot change the state of the interrupt inhibits; a master mode program can change the interrupt inhibits by executing LPSD, XPSD, or the instruction WRITE DIRECT (WD). The WD instruction is described in Chapter 3, "Control Instructions".

RP — Register pointer. This 5-bit field selects one of the 32 possible blocks of general-purpose registers as the current register block. A slave program cannot change the register pointer; a master mode program can change the register pointer by executing LPSD, XPSD, or the instruction LOAD REGISTER POINTER (LRP). The LRP instruction is described in Chapter 3, "Control Instructions".

INTERRUPT SYSTEM

The SIGMA 7 priority interrupt system is an improved version of the system used successfully in XDS 900/9300 series computers. Up to 237 external and internal interrupt levels are normally available, each with a unique location (see Table 2) assigned in core memory, each with a unique priority, and (except for the Power on and Power off interrupt levels) each capable of being selectively armed and/or enabled by the CPU. Also, any interrupt level can be "triggered" by the CPU (supplied with 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 the testing of special systems programs before the special systems equipment is actually attached to the computer, and 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 may be processed before the deferred interrupt.

SIGMA 7 interrupt levels are arranged in groups that are connected in a predetermined priority chain by groups of levels. The priority of each level within a group is fixed; the first level has the highest priority and the last level has the lowest. The user has the option of ordering a machine with a priority chain starting with the override group and connecting all remaining groups in any sequence. This allows the user to establish external interrupts above, between, or below the counter and input/output groups of internal interrupts. Figure 6 illustrates this with a configuration that a typical user might establish; where (after the override group) the counter group of internal interrupts is given the second-highest priority, followed by the first group of external interrupts, then the input/output group of internal interrupts, and finally all succeeding groups of external interrupts.

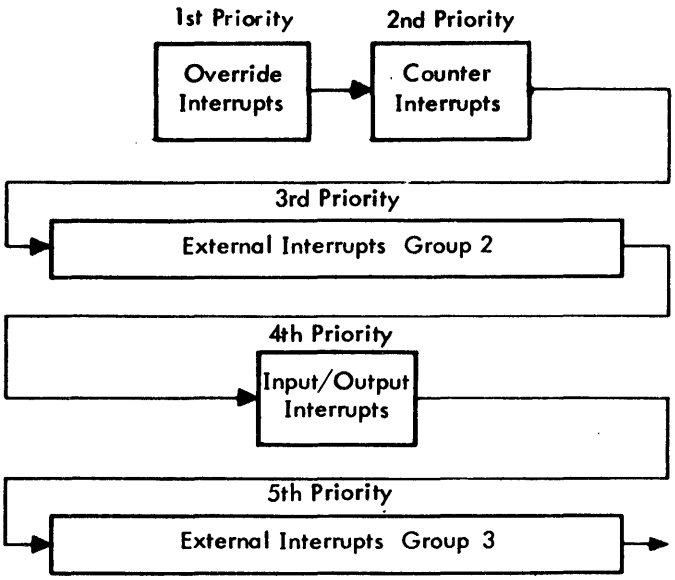


Figure 6. Typical Interrupt Priority Chain

INTERNAL INTERRUPTS

The three groups of internal interrupts include standard interrupts that are normally supplied with a SIGMA 7 system, as well as the optional power fail-safe and the additional counter interrupts.

Override Group (Locations X'50' to X'56')

This group of seven interrupt levels always has the highest priority in a SIGMA 7 system. The optional power fail-safe feature includes the Power on and Power off interrupt levels. A system can contain 2 or 4 count-pulse interrupt levels that 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 five manually switchable frequencies – the commercial line frequency, 500 Hz, 2 kHz, 8 kHz, and a user-supplied external signal – that may be different for each counter. (All counter frequencies are synchronous except for the line frequency and the signal supplied by the user.) Each of the count-pulse interrupt locations must contain one of the modify and test instructions (MTB, MTH, or MTW). Counter 4 uses the mapped location if map is currently invoked in the PSD. The results of any other instruction are unpredictable when the instruction is executed as the result of a count-pulse interrupt level advancing to the active state. When the modification (of the effective byte, halfword, or word) causes a zero result, the appropriate counter-equals-zero interrupt (see "Counter-Equals-Zero Group") is triggered. The override group also includes a memory parity interrupt level that is triggered whenever a memory parity error is reported to the CPU.

Table 2. SIGMA 7 Interrupt Locations

Location Dec.	Location Hex.	WRITE DIRECT Register bit[†]	Function	Availability	PSD Inhibit	WRITE DIRECT Group code[††]
80	50	none	Power on[†††]	optional (as a set)	none	none
81	51		Power off[†††]			
82	52	16	Counter 1 count pulse	optional (as a set)		X'0'
83	53	17	Counter 2 count pulse			
84	54	18	Counter 3 count pulse	standard		
85	55	19	Counter 4 count pulse			
86	56	20	Memory Parity			
87	57		Reserved for future use			
88	58	22	Counter 1 zero	optional (as a set)	CI	
89	59	23	Counter 2 zero			
90	5A	24	Counter 3 zero	standard		
91	5B	25	Counter 4 zero			
92	5C	26	Input/Output	standard	II	
93	5D	27	Control Panel			
94	5E		Reserved for future use			
95	5F		Reserved for future use			
96	60	16	External Group 2	optional	EI	X'2'
⋮	⋮	⋮				
111	6F	31				
112	70	16	External Group 3			X'3'
⋮	⋮	⋮				
127	7F	31				
⋮	⋮	⋮	⋮			⋮
288	120	16	External Group 14			X'E'
⋮	⋮	⋮				
303	12F	31				
304	130	16	External Group 15			X'F'
⋮	⋮	⋮				
319	13F	31				

[†]When the privileged instruction WRITE DIRECT is used in the interrupt control mode to operate on interrupt levels, the interrupt levels are selected by specific bit positions in register R. The numbers in this column indicate the bit position in register R that corresponds to the various interrupt levels.

[††]The numbers in this column indicate the group codes (for use with WRITE DIRECT) of the various interrupt levels.

[†††]These interrupts can not be disarmed, disabled, nor inhibited.

Counter-Equals-Zero Group (Locations X'58' to X'5B)

Each interrupt level in the counter-equals-zero group (called a counter-equals-zero interrupt) is associated with a count-pulse interrupt in the 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 is triggered. The counter-equals-zero interrupts can be inhibited or permitted as a group. If bit position 37 (CI) of the current program status doubleword contains a 0, the counter-equals-zero interrupts are allowed to interrupt the program being executed. However, if the CI bit is a 1, the counter-equals-zero interrupts are not allowed to interrupt the program.

Input/Output Group (Locations X'5C' and X'5D')

This interrupt group includes two standard interrupts: the I/O interrupt and the control panel interrupt. The I/O interrupt

level accepts interrupt signals from the standard I/O system. The I/O interrupt location is assumed to contain an EXCHANGE PROGRAM STATUS DOUBLEWORD (XPSD) instruction that transfers program control to a routine for servicing all I/O interrupts. The I/O routine then contains an ACKNOWLEDGE I/O INTERRUPT (AIO) instruction that identifies the source and reason for the interrupt.

The control panel interrupt level is connected to the INTERRUPT buttons on the processor control panel and the free-standing console. The control panel interrupt level can thus be triggered by the computer operator, allowing him to initiate a specific routine.

The interrupts in the input/output group can be inhibited or permitted by means of bit position 38 (II) of the program status doubleword. If II is a 0, the interrupts in the I/O group are allowed to interrupt the program being executed. However, if the II bit is a 1, the interrupts are inhibited from interrupting the program.

EXTERNAL INTERRUPTS

A SIGMA 7 system can contain up to 14 groups of optional interrupt levels, with 16 levels in each group. As shown in Figure 6, the groups can be connected in any priority sequence.

All external interrupts can be inhibited or permitted by means of bit position 39 (EI) of the program status doubleword. If EI is a 0, external interrupts are allowed to interrupt the program; however, if EI is a 1, all external interrupts are inhibited from interrupting the program.

STATES OF AN INTERRUPT LEVEL

A SIGMA 7 interrupt level is mechanized by means of three flip-flops. Two of the flip-flops are used to define any of four mutually exclusive states: disarmed, armed, waiting, and active. The third flip-flop is used as a level-enable. The various states and the conditions causing them to change state (see Figure 7) are described in the following paragraphs.

Disarmed

When an interrupt level is in the disarmed state, no signal to that interrupt level is admitted; that is, no record is retained of the existence of the signal, nor is any program interrupt caused by it at any time.

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.

Waiting

When an interrupt level in the armed state receives an interrrupt signal, it advances to the waiting state, and remains in the waiting state until it is allowed to advance to the active state. If the level-enable flip-flop is off, the interrupt level can undergo all state changes except that of moving from the waiting to the active state. Furthermore, if this flip-flop is off, the interrupt level is completely removed from the chain that determines the priority of access to the CPU. Thus, an interrupt level in the waiting state with its level-enable in the off condition does not prevent an enabled, waiting interrupt of lower priority from moving to the active state.

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 level-enable flip-flop must be set to 1).
2. The CPU must be at an interruptible point in the execution of a program.
3. The group inhibit (CI, II, or EI, if applicable) must be a 0 .
4. No higher-priority interrupt level is in the active state or is in the waiting state and totally enabled (i.e., enabled and not inhibited).

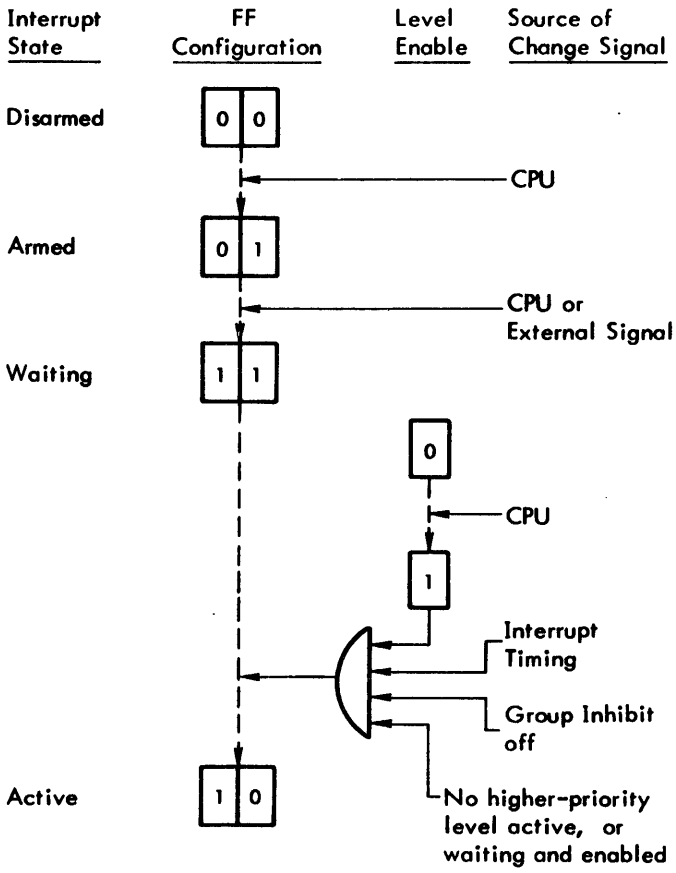


Figure 7. Interrupt Level Operation

Active

When an interrupt meets all of the conditions necessary to permit it to move from the waiting state to the active state, it is permitted to do so by being acknowledged by the computer, which then executes the contents of the assigned interrupt location as the next instruction. The instruction address portion of the program status doubleword remains unchanged until the instruction in the interrupt location is executed.

The instruction in the interrupt location must be one of the following: XPSD, MTB, MTH, or MTW. If the execution of any other instruction in an interrupt location attempted as the result of an interrupt level advancing to the active state, the results of the instruction are unpredictable.

The use of the privileged instruction XPSD in an interrupt location permits an interrupt-servicing routine to save the entire current machine environment and establish a new 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, which results in the assignment of a new block of 16 registers to the routine.

An interrupt level remains in the active state until it is cleared (removed from the active state) by the execution of the LPSD instruction or the WD instruction. An interrupt-servicing routine can itself be interrupted whenever a higher-priority interrupt level meets all of the conditions for becoming active; and then continued after the higher-priority interrupt is cleared. However, an interrupt-servicing routine cannot be interrupted by a lower-priority interrupt as long as it remains in the active state. Normally, the interrupt servicing routine clears its interrupt 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.

CONTROL OF THE INTERRUPT SYSTEM

The SIGMA 7 system has two points of interrupt control. One point of interrupt control is at the individual interrupt level. The WD instruction can be used to individually arm, disarm, enable, disable, or trigger any interrupt level except for the power fail-safe interrupts (which are always armed, always enabled, and cannot be triggered).

The second point of interrupt control is achieved by means of the interrupt inhibits (CI, II, and EI) in the program status doubleword. If an interrupt inhibit is set to 1, all interrupt levels in the corresponding group are effectively disabled; i.e., no interrupt in the group may advance from the waiting state to the active state and the group is removed from the interrupt recognition priority chain. Thus, a waiting, enabled interrupt level (in a group that is not inhibited) is not prevented from interrupting the program by a higher-priority, waiting, enabled interrupt level in a group that is inhibited. However, if an interrupt group is inhibited while a level in that group is in the active state, no lower-priority interrupt level may advance to the active state.

TIME OF INTERRUPT OCCURRENCES

The SIGMA 7 CPU permits an interrupt to occur during the following time intervals (related to the execution cycle of an instruction) providing the control panel COMPUTE switch is in the RUN position and no "halt" condition exists:

1. Between instructions: An interrupt is permitted between the completion of any instruction and the initiation of the next instruction.
2. Between the initiation of an instruction and memory or register modification: For some instructions, an interrupt is permitted after an instruction has been in process and up to the point in time when a memory location or a general register is modified. If an interrupt occurs during this time interval, the instruction is aborted, the instruction address portion of the program status doubleword remains pointing to the interrupted instruction, and the instruction in the interrupt location is executed. After the interrupt-servicing routine has been processed, program control is returned to the interrupted instruction, and the interrupted instruction is then reinitialized. Most instructions have such a short execution time that they are not abortable by an interrupt; thus, an interrupt normally occurs only before or after an instruction execution.
3. Between instruction iterations: 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 intermediate 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 the point in 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 doubleword 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 is a situation where 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 for the single-instruction).

If any of the following instructions is executed in any interrupt location, then that interrupt automatically becomes a single-instruction interrupt.

Instruction Name	Mnemonic
Modify and Test Byte	MTB
Modify and Test Halfword	MTH
Modify and Test Word	MTW

The modify and test instruction modifies the effective byte, halfword, or word (as described in the section "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 or not the memory map is currently being used. Counter 4 uses the mapped location if map is currently invoked in the PSD. The execution of a modify and test instruction in an interrupt location, including mapped and unmapped counter 4, is independent of the memory access protection codes and the write-protection locks; thus, a memory protection violation trap cannot occur (a nonexistent memory address will cause an unpredictable operation). Also, the fixed-point overflow 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, in addition to the following: If the resultant value in the effective location is zero, the corresponding counter-equals-zero interrupt is triggered.

TRAP SYSTEM

When a condition that is to result in an interrupt is sensed, a signal is sent to an interrupt level. If that level is "armed" it advances to the waiting state. When all of the conditions for its acknowledgment have been achieved, the interrupt level eventually advances to the active state, where it finally causes the computer to take an instruction from a specific location in memory. The computer may execute many instructions between the time that the interrupt requesting condition is sensed and the time that the actual interrupt acknowledgment occurs. However, detecting any of the conditions listed in Table 3 results in a trap (the immediate execution of the instruction in a unique location in memory).

When a trap condition occurs, the CPU sets the trap state. Depending on the type of trap, the instruction currently being executed by the CPU may or may not be carried to completion. In any event, the instruction is terminated with a trap sequence. In this sequence, the instruction address (IA) portion of the program status doubleword (PSD), which has already been incremented by 1, is decremented by 1 and then the instruction in the location associated with the trap is executed. An interrupt acknowledgment cannot occur until the execution of the instruction in the trap location is completed. The instruction in the trap location must be an XPSD instruction; if the execution of any other instruction in a trap location is attempted as the result of a trap activation, the results of the instruction are unpredictable. The detailed operation of XPSD is described in Chapter 3, "Control Instructions".

The XPSD instruction in a trap location is accessed without using the memory map, regardless of whether or not the memory map is in effect when the trap condition occurs. Also, no memory protection violation or privileged instruction violation can occur as a result of either accessing or executing an XPSD instruction in a trap location. Table 3 summarizes the description of the trap system.

NONALLOWED OPERATION TRAP

The occurrence of one of the nonallowed operations always causes the computer to abort the instruction being executed (at the time that the nonallowed operation is detected) and to immediately execute the instruction in trap location X'40'.

Nonexistent Instruction

Any instruction that is neither standard nor optional on SIGMA 7 is defined as nonexistent (this includes immediate addressing instructions that are indirectly addressed). If execution of a nonexistent instruction is attempted, the computer traps to location X'40' at the time the instruction is decoded. The operation of the XPSD instruction in trap location X'40' (with respect to the condition code and instruction address portions of the PSD) is as follows:

1. Store the current PSD. The condition code stored is that which existed at the end of the instruction executed immediately prior to the nonexistent instruction.
2. Load the new PSD. The current PSD is replaced by the contents of the doubleword location following the doubleword location in which the current PSD was stored.
3. Modify the new PSD:
   a. Set CC1 to 1 (CC2, CC3, and CC4 remain set at the values loaded from memory).
   b. If bit position 9 of XPSD contains a 1, the instruction address loaded from memory is incremented by 8. If bit position 9 of XPSD contains a 0, the instruction address remains at 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 by memory on the basis of the actual address presented to it. If the CPU is currently using the memory map, the virtual address will already have been modified by the memory map to generate an actual (but nonexistent) address. The operation of XPSD in trap location X'40' is as follows:

1. Store the current PSD.
2. Load the new PSD.
3. Modify the new PSD:
   a. Set CC2 to 1 (CC1, CC3, and CC4 remain set at the values loaded from memory).
   b. If bit position 9 of XPSD contains a 1, the instruction address loaded from memory is incremented by 4. If bit position 9 of XPSD contains a 0, the instruction address remains at the value loaded from memory.

Table 3. Summary of SIGMA 7 Trap System

Location Dec.	Location Hex.	Function	PSD Mask Bit	Time of Occurrence	Special Action During XPSD
64	40	Nonallowed operation	none		
		1. Nonexistent instruction		Instruction decoding	Set CC1 after new CC is loaded from memory. If bit 9 of XPSD is 1, add 8 to the new instruction address value loaded from memory.
		2. Nonexistent memory address		Prior to memory access	Set CC2 after new CC is loaded from memory. If bit 9 of XPSD is 1, add 4 to the new instruction address value loaded from memory.
		3. Privileged instruction in slave mode		Instruction decoding	Set CC3 after new CC is loaded from memory. If bit 9 of XPSD is 1, add 2 to the new instruction address value loaded from memory.
		4. Memory protection		Prior to memory access	Set CC4 after new CC is loaded from memory. If bit 9 of XPSD is 1, add 1 to the new instruction address value loaded from memory.
65	41	Unimplemented instruction	none	Instruction decoding	none
66	42	Push-down stack limit reached	TW, TS†	At the time of stack limit detection	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, registers, CC1, CC3, CC4 unchanged.	none
68	44	Floating-point fault			
		1. Characteristic overflow	none	At time of fault detection; the condition code is set to indicate the reason for the trap	none
		2. Divide by zero	none		
		3. Significance check	FS, FZ, FN		
69	45	Decimal arithmetic fault	DM	At time of fault detection; the condition code is set to indicate the reason for the trap	none
70	46	Watchdog timer runout	none	At time of runout	none
72	48	CALL 1	none	Instruction decoding	The R field of the CALL instruction is ORed into new CC settings loaded from memory. If bit 9 of XPSD is 1, the R field of the CALL instruction is added to the new instruction address value loaded from memory.
73	49	CALL 2	none	Instruction decoding	
74	4A	CALL 3	none	Instruction decoding	
75	4B	CALL 4	none	Instruction decoding	

†The push-down stack limit trap is masked within the stack pointer doubleword for each push-down stack (see page 66).

Privileged Instruction in Slave Mode

An attempt to execute a privileged instruction while the CPU is in the slave mode causes a trap to location X'40' at the time of instruction decoding. The operation of XPSD in trap location X'40' is as follows:

1. Store the current PSD.
2. Load the new PSD.
3. Modify the new PSD.
   a. Set CC3 to 1 (CC1, CC2, and CC4 remain at the values loaded from memory).
   b. If bit position 9 of XPSD contains a 1, the instruction address loaded from memory is incremented by 2. If bit position 9 of XPSD contains a 0, the instruction address remains at the value loaded from memory.

The operation codes, 0C, 0D, 2C, 2D, and their indirectly addressed forms, 8C, 8D, AC, AD, are both nonexistent and privileged. If one of these operation codes is used while the CPU is in the slave state, both CC1 and CC3 will be set to 1's after the new PSD has been loaded, and if bit position 9 of XPSD contains a 1, the instruction address loaded from memory is incremented by 10.

Memory Protection Violation

A memory protection violation can occur either because of a memory map access control bit violation (by a slave program using the memory map) or because of a memory write lock violation (by either a slave or a master mode program). When either memory protection violation occurs, the CPU aborts execution of the current instruction (without changing protected memory) and traps to location X'40'. The operation of the XPSD in trap location X'40' is as follows:

1. Store the current PSD,
2. Load the current PSD.
3. Modify the new PSD:
   a. Set CC4 to 1 (CC1, CC2, and CC3 remain at the values loaded from memory.
   b. If bit position 9 of XPSD contains a 1, the instruction address loaded from memory is incremented by 1. If bit position 9 of XPSD contains a 0, the instruction address remains at the value loaded from memory.

An attempt to access a memory location that is both protected and nonexistent causes both CC2 and CC4 to be set to 1's after the new PSD has been loaded, and if bit position 9 of XPSD contains a 1, the instruction address loaded from memory is incremented by 5.

UNIMPLEMENTED INSTRUCTION TRAP

There are two SIGMA 7 optional instruction groups: the decimal option and the floating-point option. The decimal option includes the following instructions:

Instruction Name	Mnemonic	Operation Code
Decimal Load	DL	X'7E'
Decimal Store	DST	X'7F'
Decimal Add	DA	X'79'
Decimal Subtract	DS	X'78'
Decimal Multiply	DM	X'7B'
Decimal Divide	DD	X'7A'
Decimal Compare	DC	X'7D'
Decimal Shift Arithmetic	DSA	X'7C'
Pack Decimal Digits	PACK	X'76'
Unpack Decimal Digits	UNPK	X'77'
Edit Byte String	EBS	X'63'

The floating-point option includes the following instructions:

Floating Add Short	FAS	X'3D'
Floating Add Long	FAL	X'1D'
Floating Subtract Short	FSS	X'3C'
Floating Subtract Long	FSL	X'1C'
Floating Multiply Short	FMS	X'3F'
Floating Multiply Long	FML	X'1F'
Floating Divide Short	FDS	X'3E'
Floating Divide Long	FDL	X'1E'

If an attempt is made to execute an instruction (directly or indirectly addressed) in either of these groups when the required option is not implemented, the computer traps to location X'41'. An indirectly addressed EDIT BYTE STRING (EBS) instruction is always treated as a nonexistent instruction rather than as an unimplemented instruction. The move to memory control (MMC) instruction is always considered implemented even if the memory map option or the memory-protection option are not implemented. The operation of the XPSD in trap location X'41' is as follows:

1. Store the current PSD. The condition code stored is that which existed at the end of the instruction immediately prior to the unimplemented instruction.
2. Load the new PSD. The condition code and the instruction address portions of the PSD remain at the values 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 Name	Mnemonic
Push Word	PSW
Pull Word	PLW
Push Multiple	PSM
Pull Multiple	PLM
Modify Stack Pointer	MSP

During the execution of any stack-manipulating instruction (see Push-down Instructions) the stack is either pushed (words added to stack) or pulled (words removed from stack). In either case, the space count and word count fields of the stack pointer doubleword are tested prior to moving any words. If execution of the instruction would cause the space count to become less than 0 or greater than $2^{15}-1$, the instruction is aborted with memory and registers unchanged; then, if bit 32 (TS) of the stack pointer doubleword is 0, the CPU traps to location X'42'. If execution of the instruction would cause the word count to become less than 0 or greater than $2^{15}-1$, the instruction is aborted with memory and registers unchanged; then, if bit 48 (TW) of the stack pointer doubleword is a 0, the CPU traps to location X'42'. If trapping does occur, the condition code remains at the value it had immediately prior to the instruction that caused the trap. When trapping is inhibited, either CC1 or CC3 is set to 1 (or both CC1 and CC3 are set to 1's) to indicate the reason for aborting the instruction. The stack pointer doubleword, memory, and registers are modified only if the instruction is successfully executed. The execution of XPSD in trap location X'42' is as follows:

1. Store the current PSD. The condition code stored is that which existed immediately prior to the execution of the aborted push-down instruction.

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

FIXED-POINT OVERFLOW TRAP

Fixed-point overflow can occur for any of the following instructions:

Instruction Name	Mnemonic
Load Complement Word	LCW
Load Absolute Word	LAW
Load Complement Doubleword	LCD
Load Absolute Doubleword	LAD
Add Immediate	AI
Add Halfword	AH
Add Word	AW
Add Doubleword	AD
Subtract Halfword	SH
Subtract Word	SW
Subtract Doubleword	SD
Divide Halfword	DH
Divide Word	DW
Add Word to Memory	AWM
Modify and Test Halfword	MTH
Modify and Test Word	MTW

Except for the instructions DIVIDE HALFWORD (DH) and DIVIDE WORD (DW), the 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 PSD) is a 1, the CPU 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 registers and CC2 is set to 1; but 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 CPU traps to location X'43' instead of executing the next instruction in sequence.

1. Store the current PSD. If the instruction causing the trap was an instruction other than DW or DH, the stored condition code[†] is interpreted as follows:

CC1[††]	CC2	CC3	CC4	Meaning
-[†]	1	0	0	result after overflow is zero
-	1	0	1	result after overflow is negative
-	1	1	0	result after overflow is positive
0	-	-	-	no carry from bit position 0
1	-	-	-	carry from bit position 0

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

CC1	CC2	CC3	CC4	Meaning
-	1	-	-	overflow

2. Load the new PSD. The condition code and instruction address portions of the PSD 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 actually loaded into the general registers; thus, the floating-point operation that causes an arithmetic fault is not carried to completion (in the sense that the original contents of the general registers remain unchanged). Instead, the computer 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 result in a trap condition only if the floating-point mode controls (FS, FZ, and FN) in the program status doubleword are set to the appropriate state.

[†] A hyphen (-) indicates that the condition code bit is not affected by the condition given under the "Meaning" heading.

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

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

1. Store the current PSD. 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	divide by zero
0	1	0	1	characteristic overflow, negative result
0	1	1	0	characteristic overflow, positive result

If none of the above conditions occurs, but characteristic underflow occurs with the floating zero (FZ) mode bit set to 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 occurs, 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 2 postnormalizing shifts, negative result
1	0	1	0	more than 2 postnormalizing shifts, positive result

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

DECIMAL ARITHMETIC FAULT TRAP

When either of two decimal fault conditions occur (see Decimal Instructions), the normal sequencing of instruction execution 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 PSD) 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 bit is a 1, the computer traps to location X'45' at the time of fault detection.

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

1. Store the current PSD. 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 PSD. The condition code and instruction address portions of the PSD remain at the values loaded from memory.

WATCHDOG TIMER RUNOUT TRAP

The instruction watchdog timer insures that the CPU must periodically reach interruptible points of operation in the execution of instructions. An interruptible point is a time during the execution of a program when an interrupt request (if present) would be acknowledged. Interruptible points occur at the end of every instruction and during the execution of some instructions (such as the byte string group). The watchdog timer measures elapsed time from the last interruptible point. If the maximum allowable time has been reached before the next time that an interrupt could be recognized, the current instruction is aborted and the watchdog timer runout trap is activated. Except for a nonexistent address used with READ DIRECT (RD) or WRITE DIRECT (WD) instructions, programs trapped by the watchdog timer cannot (in general) be continued. Execution of XPSD in trap location X'46' is as follows:

1. Store the current PSD. The stored condition code is, in general, meaningless.
2. Load the new PSD. The instruction address portion of the PSD remain at the values loaded from memory; however, the resulting condition code is, generally, meaningless.

CALL INSTRUCTION TRAPS

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

1. Store the current PSD. The stored condition code is that which existed at the end of the instruction immediately prior to the call instruction.
2. Load the new PSD.
3. Modify the new PSD.

   a. The R field of the call instruction is logically ORed with the condition code value loaded from memory, and the result is loaded into the condition code.

   b. If bit 9 of XPSD contains a 1, the R field of the call instruction is added to the instruction address loaded from memory.

      If bit 9 of XPSD contains a 0, the instruction address remains at the value loaded from memory.

3. INSTRUCTION REPERTOIRE

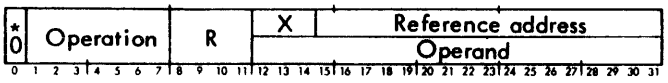
This section describes all SIGMA 7 instructions, grouped into the following functional classes:

SIGMA 7 instructions are described in the following format:

MNEMONIC ① INSTRUCTION NAME ②

(Addressing type ③, Optional ④, Privileged ⑤, Interrupt Action ⑥)

⑦



Description ⑧

Affected ⑨ Trap ⑩

Symbolic notation ⑪

Condition Code Settings ⑫

Trap Action ⑬

Example ⑭

1. MNEMONIC is the code used by the SIGMA 7 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 core 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 core 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 core 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 core memory or in the current block of general registers. The addressed doubleword is automatically located within doubleword storage boundaries.
   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, in which case the computer 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 computer treats the instruction 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'. Indexing does not apply to this type of instruction.
4. If the instruction is not in the standard SIGMA 7 instruction set, it is labeled "optional". If execution of an optional instruction is attempted on a computer in which the instruction is not implemented, the computer unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'41', which is the "unimplemented instruction trap".
5. If the instruction is not executable while the computer is in the slave mode, it is labeled "privileged". If execution of a privileged instruction is attempted while the computer is in the slave mode, the computer unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40'.
6. 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". Otherwise, the instruction is either completed or the instruction is aborted and then restarted after the interrupt is cleared. 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 In the case of aborted instructions, all affected registers are restored to the values they contained immediately before the aborted instruction was begun.

7. Instruction format:

   a. Indirect addressing – If bit position 0 of the instruction format contains an asterisk (*), the instruction can utilize indirect addressing; however, if bit position 0 of the instruction format contains a 0, the instruction is of the immediate addressing 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.

   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 can specify 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 reference address field (bit positions 15-31) of the instruction format is used as the initial address value for an instruction operand. For instructions of the immediate addressing type, the effective address of the instruction is not used to access an operand; instead, the effective address itself is used as an operand. In these cases, the function of the effective address is represented in the lower half of the reference address field in the instruction format diagram.

   f. Value field – In some fixed-point arithmetic instructions, bit positions 12-31 of the instruction format contain the word "value". This 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 word "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. Ignored fields – In the instruction format diagrams, any area that is shaded represents a field or bit position that is ignored by the computer (i. e., the content of the shaded field or bit has no effect on instruction execution) but should be coded with 0's so as to preclude conflict with possible modifications.

      In any format diagram of a general register or memory word modified by an instruction, a shaded area represents a field whose content is indeterminate after execution of the instruction.

8. The description of the instruction defines the operations performed by the computer 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.

9. 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 doubleword is considered to be affected only if a branch condition can occur as a result of the instruction execution, since the instruction address is updated (incremented by 1) as part of every instruction execution.

10. All trap conditions that may be invoked by the execution of the instruction are listed after the word "Trap". SIGMA 7 trap locations are summarized in the section "Trap System".

11. The symbolic notation presents the instruction operation as a series of generalized symbolic statements. The symbolic terms used in the notation are defined in Table 4.

12. 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 by the reason given for the condition code bit(s) containing a 0 or 1. 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
-	0	-	-	the logical product (AND) of the two operands is zero
-	1	-	-	the logical product of the two operands is nonzero

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 then the appropriate trap location is activated.

13. Actions taken by the computer 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 computer. In order to avoid unnecessary repetition, the two trap conditions that apply to all instructions (i.e., nonallowed operations and watchdog timer runout) are not described for each instruction.

14. 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.

Table 4. Glossary of Symbolic Terms

Term	Meaning
( )	Contents of
AM	Fixed-point arithmetic trap mask – bit 11 of the program status doubleword. If this bit is a 1, the computer traps to location X'43' after executing an instruction that causes fixed-point overflow; if this bit is a 0, the computer does not trap to location X'43'.
I	Instruction register – the internal CPU register used to hold instructions obtained from memory while they are being decoded.
R	General register address value – the 4-bit contents of bit positions 8-11 (the R field) of an instruction word, also expressed symbolically as $(I)_{8-11}$. In the instruction descriptions, register R is the general register (of the current register block) that corresponds to the R field address value.
Ru1	Odd register address value – register Ru1 is the general register pointed to by the value obtained by logically ORing 0001 into the address value for register R. Thus, if the R field of an instruction contains an even value, Ru1 = R + 1 and if the R field contains an odd value, Ru1 = R.
X	Index register address value – the 3-bit contents of bit positions 12-14 (the X field) of an instruction word. If X = 0 for an instruction, no indexing is performed. If $X \neq 0$ for an instruction, indexing is performed (after indirect addressing if indirect addressing is called for) with general register X in the current register block.
RA	Reference address – the contents of bit positions 15-31 of an instruction word. This 17-bit field is capable of directly addressing any general register in the current register block (by using a value in the range 0-15) or any word in core memory in the address range 16 through 131,071. This address value is the initial address value for any subsequent address computations, memory mapping, or both computation and mapping.
EVA	Effective virtual address – the virtual address value obtained as a result of indirect addressing and/or indexing. This address value is independent of the program's actual location in core memory, and is the final address value before memory mapping is performed.
EBL	Effective byte location – the byte location pointed to by the effective virtual address of an instruction for a byte operation.
EB	Effective byte – the 8-bit contents of the effective byte location, or (EBL).
EHL	Effective halfword location – the halfword location pointed to by the effective virtual address of an instruction for a halfword operation.
EH	Effective halfword – the 16-bit contents of the effective halfword location, or (EHL).
EWL	Effective word location – the word location pointed to by the effective virtual address of an instruction for a word operation.
EW	Effective word – the 32-bit contents of the effective word location, or (EWL).
EDL	Effective doubleword location – the doubleword location pointed to by the effective virtual address of an instruction for a doubleword operation. 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 0. Thus, an odd-numbered word address (referring to the middle of a doubleword) designates the same doubleword as an even-numbered word address, when used for a doubleword operation.
ED	Effective doubleword – the 64-bit contents of the effective doubleword location, or (EDL).
CC	Condition code – a 4-bit value (whose bit positions are labeled CC1, CC2, CC3, and CC4) that is established as part of the execution of most SIGMA 7 instructions.
FN	Floating normalize mode control – bit 7 of the program status doubleword. If this bit is a 0,

Table 4. Glossary of Symbolic Terms (cont.)

Term	Meaning	Term	Meaning
FN (cont.)	the results of floating-point additions and subtractions are to be normalized; if this bit is a 1, the results are not normalized.	X'n' (cont.)	(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'.
FS	Floating significance mode control – bit 5 of the program status doubleword. If this bit is a 1, the computer traps to location X'44' when more than two hexadecimal places of postnormalization shifting are required for a floating-point addition or subtraction; if this bit is 0, no significance checking is performed.	n	AND (logical product, where 0 n 0 = 0, 0 n 1 = 0, 1 n 0 = 0, and 1 n 1 = 1).
		u	OR (logical inclusive OR, where 0 u 0 = 0, 0 u 1 = 1, 1 u 0 = 1, and 1 u 1 = 1).
FZ	Floating zero mode control – bit 6 of the program status doubleword. If this bit is a 1, the computer traps to location X'44' when either characteristic underflow or a zero result occurs for a floating-point multiplication or division; if this bit is a 0, characteristics underflow and zero results are treated as normal conditions.	ⓤ	EOR (logical exclusive OR, where 0ⓤ0 = 0, 0ⓤ1 = 1, 1ⓤ0 = 1, and 1ⓤ1 = 0).
		SE	Sign extension – some SIGMA 7 instructions operate on two operands of different lengths. The two operands are made equal in length by extending the sign of the shorter operand by the required number of bit positions. For positive operands, the result of sign extension is high-order 0's prefixed to the operand; for negative operands, high-order 1's are prefixed to the operand. This sign extension process is performed after the operand is accessed from memory and before the operation called for by the instruction code is performed.
IA	Instruction address – the 17-bit value that defines the virtual address of an instruction immediately prior to the time that the instruction is executed.		
X'n'	Hexadecimal qualifier – a hexadecimal value (n) is an unsigned string of hexadecimal digits		

LOAD/STORE INSTRUCTIONS

The following load/store instructions are implemented in SIGMA 7 computers:

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 Selective	LS
Load Multiple	LM
Load Conditions and Floating Control Immediate	LCFI
Load Conditions and Floating Control	LCF
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

SIGMA 7 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 general registers of the current register block. Load instructions do not affect core memory storage; 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:

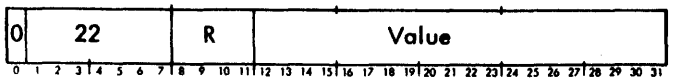
Condition code settings:

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

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

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	zero
-	-	0	1	negative
-	-	1	0	positive

If LI 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 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.

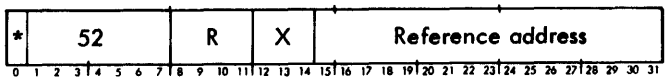
Affected: (R), CC3, CC4

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

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.

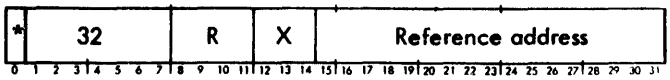
Affected: (R), CC3, CC4

$EH_{SE} \longrightarrow R$

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.

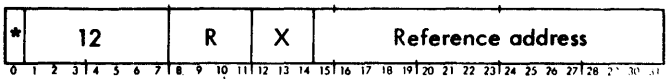
Affected: (R), CC3, CC4

$EW \longrightarrow R$

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 Rul 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), (Rul), CC3, CC4

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

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'
(Rul)	=	xxxxxxxx	X'89ABCDEF'
CC	=	xxxx	xx10

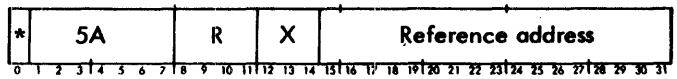
Example 2, odd R field value:

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

Example 3, odd R field value:

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

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

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.)

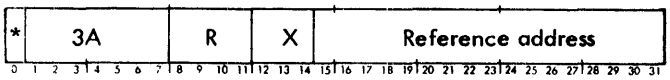
Affected: (R),CC3,CC4

$|EH_{SE}| \longrightarrow R$

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.

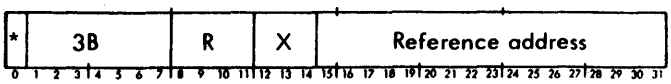
$-EW \longrightarrow R$

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 computer traps to location X'43' after execution of LOAD COMPLEMENT WORD; otherwise, the computer 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

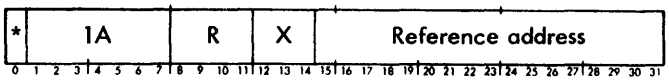
$|EW| \longrightarrow R$

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 computer traps to location X'43' after execution of LOAD ABSOLUTE WORD; otherwise, the computer 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 computer traps to location X'43' after execution of LOAD COMPLEMENT DOUBLEWORD; otherwise, the computer executes the next instruction in sequence.

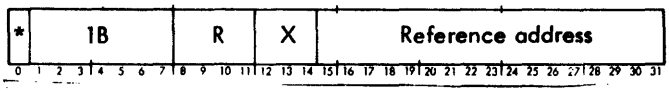
Example 1, even R field value:

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

Example 2, odd R field value:

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 computer traps to location X'43' after execution of LOAD ABSOLUTE DOUBLEWORD; otherwise, the computer 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

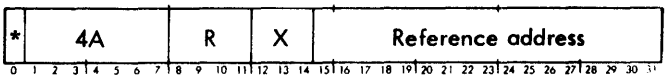
Example 2, even R field value:

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

Example 3, odd R field value:

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

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

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

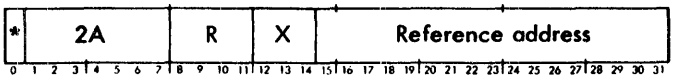
Example 1, even R field value:

		Before execution	After execution
EW	=	X'01234567'	X'01234567'
(R+1)	=	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).

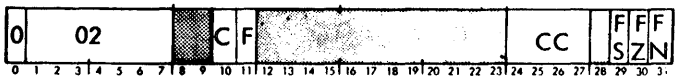
The number of words to be loaded into the general registers is determined by the value of the condition code immediately before the execution of LM. (The desired value of the condition code can be set with LCF or LCFI.) An initial value of 0000 for the condition code causes 16 consecutive words to be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

Affected: (R) to (R+CC-1)
(EWL) → R, (EWL+1) → R+1, . . . , (EWL+CC-1) → R+CC-1

If the instruction starts loading words from an accessible region of memory and then crosses into an inaccessible memory region, either the memory protection trap or the nonexistent memory address trap can occur. In either case, the trap is activated with the condition code unchanged from the value it contained before the execution of LM. The effective address of the instruction permits the trap routine to compute how many registers have been loaded. Since it is permissible to use indirect addressing or indexing through a general register, or even to execute an instruction located in a general register, a trapped LM instruction may have already overwritten the index, direct address, or the LM instruction itself, thus destroying any possibility of continuing the program successfully. If such programming must be done, it is advisable that the register containing the direct address, index displacement, or instruction be the last register loaded by the LM instruction.

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 core 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 29 through 31 of the instruction word into the floating significance (FS), floating zero (FZ), and floating normalize (FN) mode control bits, respectively (in the program status doubleword); however, if bit 11 is 0, the FS, FZ and FN control bits are not affected. The functions of the floating-point control bits are described in the section "Floating-point Instructions".

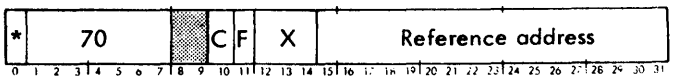
Affected: CC, FS, FZ, FN
If $(I)_{10} = 1$, $(I)_{24-27} \longrightarrow$ CC
If $(I)_{10} = 0$, CC is not affected
If $(I)_{11} = 1$, $(I)_{29-31} \longrightarrow$ FS, FZ, FN
If $(I)_{11} = 0$, FS, FZ, and FN not affected

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 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 5 through 7 of the effective byte into the floating significance (FS), floating zero (FZ), and floating normalize (FN) mode control bits, respectively; however, if bit 11 is 0, the FS, FZ and FN control bits are not affected. The

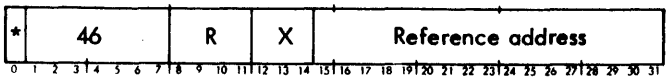
functions of the floating-point mode control bits are described in the section "Floating-point Instructions".

Affected: CC, FS, FZ, FN
If $(I)_{10} = 1$, $EB_{0-3} \longrightarrow CC$
If $(I)_{10} = 0$, CC not affected
If $(I)_{11} = 1$, $EB_{5-7} \longrightarrow$ FS, FZ, FN
If $(I)_{11} = 0$, FS, FZ, FN not affected

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

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

XW EXCHANGE WORD
(Word index alignment)



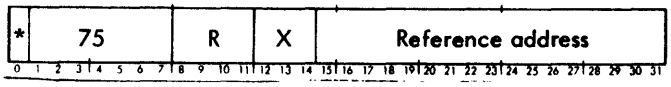
EXCHANGE WORD exchanges the contents of register R with the contents of the effective word location.

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

Condition code settings:

1	2	3	4	Result in R
-	-	0	0	zero
-	-	0	1	negative
-	-	1	0	positive

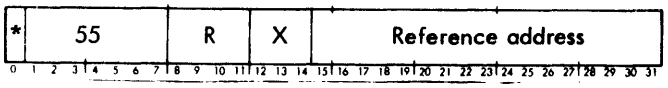
STB STORE BYTE
(Byte index alignment)



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:

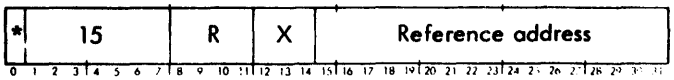
1	2	3	4	Information in R
-	0	-	-	$(R)_{0-16}$ = all 0's or all 1's
-	1	-	-	$(R)_{0-16} \neq$ all 0's or all 1's

STW STORE WORD
(Word index alignment)

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

Affected: (EWL)
(R) ⟶ 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) $\longrightarrow EDL_{32-63}$

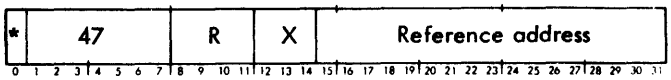
Example 1, even R field value:

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

Example 2, odd R field value:

(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) ∪ EW ⟶ EWL

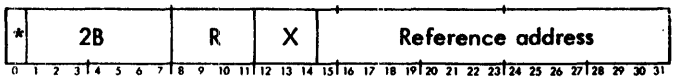
Example 1, even R field value:

	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'	X'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

If the instruction starts storing words into an accessible region of the memory and then crosses into an inaccessible memory region, either the memory protection trap or the nonexistent memory address trap can occur. In either case, the trap is activated with the condition code unchanged from the value it contained before the execution of STM. The effective address of the instruction permits the trap routine to compute how many words of memory have been changed. Since it is permissible to use indirect addressing through one of the affected locations, or even to execute an instruction located in one of the affected locations, a trapped STM instruction may have already overwritten the direct address, or the STM instruction itself, thus destroying any possibility of continuing the program successfully. If such programming must be done, it is advisable that the direct address, or the STM instruction, occupy the last location in which the contents of a register are to be stored by the STM instruction.

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 in core memory. In this case the results will be unpredictable if any of the source registers are also used as destination registers.

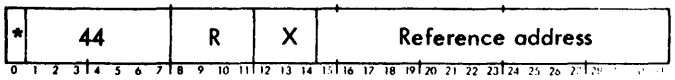
STCF STORE CONDITIONS AND FLOATING CONTROL
(Byte index alignment)

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

Affected: (EBL)
$(PSD)_{0-7}$ → EBL

ANALYZE/INTERPRET INSTRUCTIONS

ANLZ ANALYZE
(Word index alignment)



The ANALYZE instruction treats the effective word as a SIGMA 7 instruction and calculates the effective virtual address that would be generated by the instruction if the instruction were to be executed. ANALYZE produces an answer to the question, "What effective virtual address would be used by the instruction located at N if it were executed now?" The ANALYZE instruction determines the addressing type of the "analyzed" instruction, calculates its effective virtual address (if the instruction is not an immediate-operand instruction), and loads the effective virtual address into register R as a displacement value (the condition code settings for the ANALYZE instruction indicate the addressing type of the analyzed instruction).

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 occur, 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:

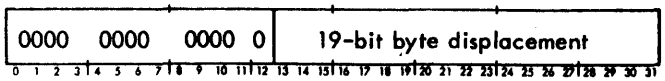
1. The contents of the location pointed to by the effective virtual address of the ANLZ instruction is obtained. This effective word is the instruction to be analyzed. From a memory-protection viewpoint, the instruction (to be analyzed) is treated as an operand of the ANLZ instruction; that is, the analyzed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

2a. 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.

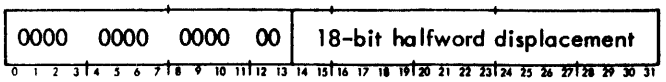
2b. If the operation code portion of the effective word specifies a reference-addressing instruction type, the condition code is set to indicate the addressing type of the analyzed instruction and the effective address of the analyzed instruction is computed (using all of the normal address computation 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. 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:

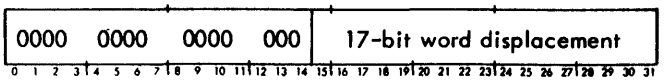
Byte Addressing:



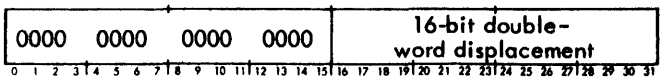
Halfword Addressing:



Word Addressing:



Doubleword Addressing:



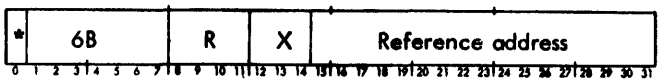
Operation codes and mnemonics for the SIGMA 7 instruction set are shown in Table 5. Circled numbers in the table indicate the condition code value (decimal) available to the next instruction after ANALYZE when a direct-addressing operation code in the corresponding addressing type is analyzed.

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 4-15 of the effective word into bit positions 20-31 of register R (and loads 0's into the remainder of register R), and then 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). 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).

Table 5. ANALYZE Table for SIGMA 7 Operation Codes

X	X'00'+n	20+	40+	60+
00	—	AI	TTBS	CBS
01	—	CI	TBS ①	MBS
02	LCFI ⑨	LI	—	—
03	—	MI	—	EBS
04	CAL1	SF	ANLZ	BDR
05	CAL2	S	CS	BIR
06	CAL3	—	XW	AWM
07	CAL4	—	STS	EXU
08	PLW	CVS	EOR	BCR
09	PSW	CVA ⑧	OR	BCS
0A	PLM	LM	LS	BAL
0B	PSM	STM	AND	INT
0C	—	—	SIO	RD
0D	—	—	TIO	WD
0E	LPSD ⑫	WAIT	TDV	AIO
0F	XPSD	LRP	HIO	MMC
10	AD	AW	AH	LCF
11	CD	CW	CH	CB
12	LD	LW	LH	LB
13	MSP	MTW	MTH	MTB
14	—	—	—	STCF
15	STD	STW	STH	STB ⓪
16	—	DW	DH ④	PACK
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

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 following fixed-point arithmetic instructions are included as a standard feature of the SIGMA 7 computer:

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

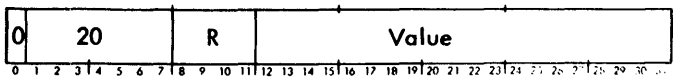
The fixed-point arithmetic instruction set performs binary addition, subtraction, multiplication, and division with integer operands that may be data, addresses, index values, or counts. One operand may be either in the instruction word itself or may be in one or two of the current general registers; the second operand may be either in core 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

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

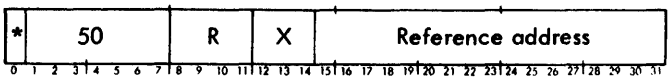
Condition code settings:

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 AI 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 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 computer traps to location X'43' after loading the sum into register R; otherwise, the computer 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

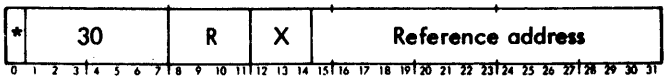
$(R) + EH_{SE} \longrightarrow R$

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 computer traps to location X'43' after loading the sum into register R; otherwise, the computer 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
(R) + EW ⟶ R

Trap: Fixed-point overflow

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 computer traps to location X'43' after loading the sum into register R; otherwise, the computer 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 result in register R is unpredictable.

Affected: (R), (Ru1), CC
(R, Ru1) + ED ⟶ R, Ru1

Trap: Fixed-point overflow

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 computer traps to location X'43' after loading the sum into registers R and Ru1; otherwise, the computer executes the next instruction in sequence.

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
$-EH_{SE}$ + (R) ⟶ R

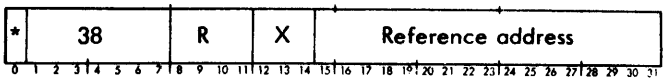
Trap: Fixed-point overflow

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 computer traps to location X'43' after loading the sum into register R; otherwise, the computer 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
-EW + (R) ⟶ R

Trap: Fixed-point overflow

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 computer traps to location X'43' after loading the sum into register R; otherwise, the computer 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. R must be an even value; if R is an odd value, the result in register R is unpredictable.

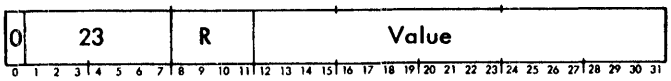
Affected: (R),(Ru1),CC Trap: Fixed-point overflow
-ED + (R, Ru1) ⟶ R, Ru1

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 computer traps to location X'43' after the result is loaded into registers R and Ru1; otherwise, the computer executes the next instruction in sequence.

MI MULTIPLY IMMEDIATE
(Immediate operand)



The value field (bit positions 12-31 of the instructions 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
(Ru1) x $(I)_{12-31SE}$ ⟶ 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 computer 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 computer executes the next instruction in sequence.

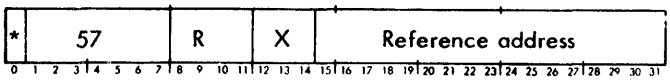
Example 1, even R field value:

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

Example 2, odd R field value:

$(I)_{12-31}$	= 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
$(R)_{16-31} \times EH \longrightarrow Ru1$

Condition code settings:

1	2	3	4	Result in Ru1
-	-	0	0	zero
-	-	0	1	negative
-	-	1	0	positive

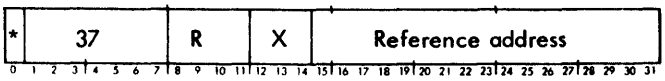
Example 1, even R field value:

		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:

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).

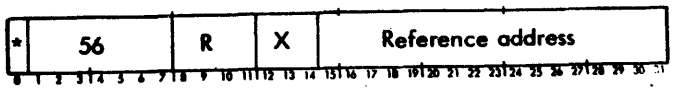
If R is an odd value, the result in register R is the 32 low-order bits of the product. Thus, in order to generate a 64-bit product, the R field of the instruction must be even and the multiplicand must be in register R+1. The condition code settings are based on the 64-bit product formed during instruction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

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	-	-	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.

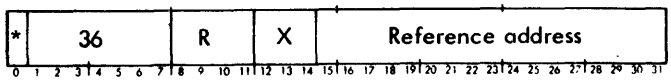
Affected: (R), CC2, CC3, CC4 Trap: Fixed-point overflow
(R) ÷ EH ⟶ R

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 computer 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:

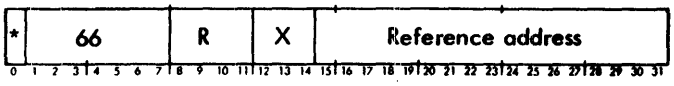
1	2	3	4	Result in Ru1
-	0	0	0	zero quotient, no overflow
-	0	0	1	negative quotient, no overflow

Condition code settings:

1	2	3	4	Result in Ru1
-	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 computer traps to location X'43' with the original contents of register R, register Ru1, CC1, CC3, and CC4 unchanged; otherwise, the computer 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.

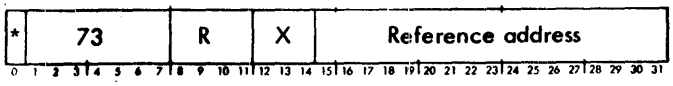
Affected: (EWL), CC Trap: Fixed-point overflow
EW + (R) ⟶ EWL

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 the fixed-point arithmetic trap mask (AM) is a 1, the computer traps to location X'43' after the result is stored in the effective word location; otherwise, the computer 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 $(I)_{8-11} = 0$;
(EBL) and CC if $(I)_{8-11} \neq 0$

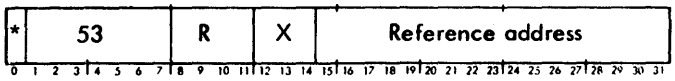
If $(I)_{8-11} \neq 0$, EB + $(I)_{8-11SE}$ ⟶ EBL and set CC
If $(I)_{8-11} = 0$, test byte and set CC

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 location, the condition code is not affected (see Chapter 2, "Single-Instruction Interrupts").

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.

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 $(I)_{8-11} \neq 0$

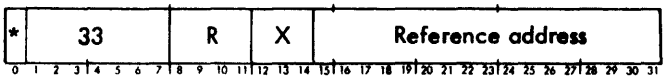
If $(I)_{8-11} = 0$, test halfword and set CC
If $(I)_{8-11} \neq 0$, EH + $(I)_{8-11SE}$ ⟶ EHL and set CC

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 computer traps to location X'43' after the result is stored in the effective halfword location; otherwise, the computer executes the next instruction in sequence. However, if MTH is executed in an interrupt location, the condition code is not affected and no fixed-point overflow trap can occur (see "Single-Instruction Interrupts").

MTW MODIFY AND TEST WORD
(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 the 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 computer traps to location X'43' after the result is stored in the effective word location; otherwise, the computer executes the next instruction in sequence. However, if MTW is executed in an interrupt location, the condition code is not affected and no fixed-point overflow trap can occur (see "Single-Instruction Interrupts").

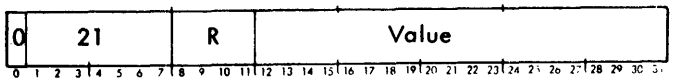
COMPARISON INSTRUCTIONS

The following comparison instructions are available to SIGMA 7 computers:

Instruction Name	Mnemonic
Compare Immediate	CI
Compare Byte	CB
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 SIGMA 7 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

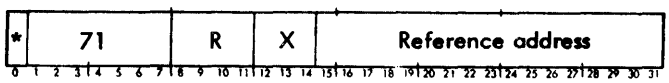
$(R) : (I)_{12-31SE}$

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 computer 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

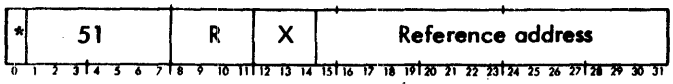
treated as positive integer magnitudes) and sets the condition code according to the results of the comparison.

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)



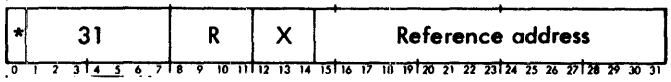
COMPARE HALFWORD extends the sign of the effective halfword 16 bit positions to the left, then compares the resultant 32-bit word with the contents of register R (with both words treated as signed, fixed-point quantities) and sets the condition code according to the results of the comparison.

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
-	0	-	-	no 1-bits compare, $(R) \cap EH_{SE} = 0$
-	1	-	-	one or more 1-bits compare, $(R) \cap EH_{SE} \neq 0$

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, $(R) \cap EW = 0$
-	1	-	-	one or more 1-bits compare, $(R) \cap EW \neq 0$

CD COMPARE DOUBLEWORD
(Doubleword index alignment)

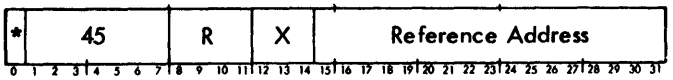
COMPARE DOUBLEWORD compares the effective doubleword with the contents of registers R and Ru1 (with both doublewords treated as signed, fixed-point quantities) and sets the condition code according to the results of the comparison. If the R field of CD is an odd value, CD forms a 64-bit register operand (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
(Word index alignment)



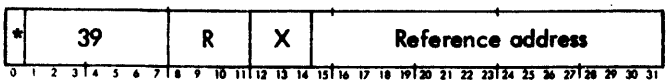
COMPARE SELECTIVE compares the contents of register R with the effective word in only those bit positions selected by a 1 in corresponding bit positions of register Ru1 (mask). The contents of register R and the effective word are ignored in those bit positions designated by a 0 in corresponding bit 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)



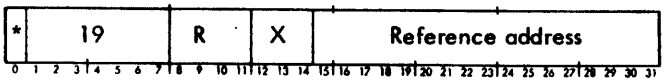
COMPARE WITH LIMITS IN REGISTERS simultaneously compares the effective word with the contents of register R and with the contents of register Ru1 (with all three words treated as signed fixed-point quantities), and sets the condition code according to the results of the comparisons.

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)

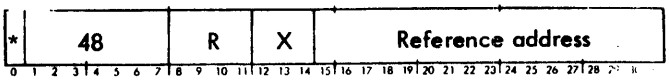
OR WORD logically ORs the effective word into register R. If corresponding bits of register R and the effective word are both 0, a 0 remains in register R; otherwise, a 1 is placed in the corresponding bit position of register R. The effective word is not affected.

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)



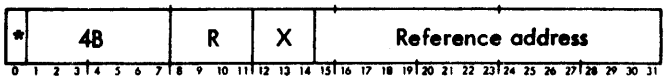
EXCLUSIVE OR WORD logically exclusive ORs the effective word into register R. If corresponding bits of register R and the effective word are different, a 1 is placed in the corresponding bit position of register R; if the contents of the corresponding bit positions are alike, a 0 is placed in the corresponding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

Affected: (R), CC3, CC4
(R) ⓤ EW ⟶ R, where 0 ⓤ 0 = 0, 0 ⓤ 1 = 1,
1 ⓤ 0 = 1, 1 ⓤ 1 = 0

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) ∩ EW ⟶ R, where $0 \cap 0 = 0$, $0 \cap 1 = 0$,
$1 \cap 0 = 0$, $1 \cap 1 = 1$

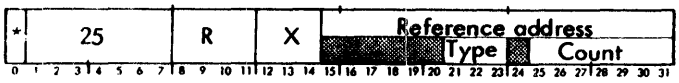
Condition code settings:

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

SHIFT INSTRUCTIONS

The instruction format for logical, circular, and arithmetic 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 21-31 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.

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	Undefined
1	1	1	Undefined

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.

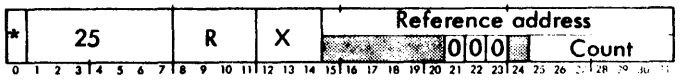
Overflow occurs (on left shifts only) whenever the value of the sign bit (bit position 0 of register R) changes. At the completion of logical left, circular left, and arithmetic left shifts, the condition code is set as follows:

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
-	0	-	-	no overflow on left shift
-	1	-	-	overflow on left shift

At the completion of logical right, circular right, and arithmetic right shifts, the condition code is set as follows:

1	2	3	4
0	0	-	-

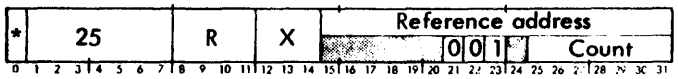
Logical Shift, Single Register



If the shift count, C, is positive, the contents of register R are shifted left C places, 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 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

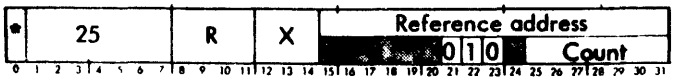


If the shift count, C, is positive, the contents of registers R and Ru1 are shifted left C places, with 0's copied into vacated bit positions on the right. Bits shifted past bit position 0 of register Ru1 are copied into bit position 31 of register R. (Bits shifted past R_0 are lost.) If C is negative,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

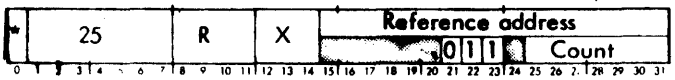
Circular Shift, Single Register



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

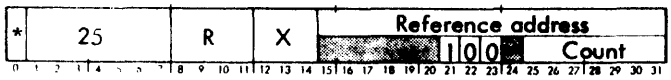
Circular Shift, Double Register



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

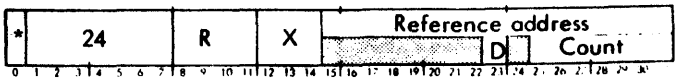
If the shift count, C, is positive, the contents of registers R and Ru1 are shifted left C places, with 0's copied into vacated bit positions on the right. Bits shifted past bit position 0 of register Ru1 are copied into bit position 31 of register R. (Bits shifted past R_0 are lost.) If C is negative,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

FLOATING-POINT SHIFT

Floating-point numbers are defined on page 47. The format for the floating-point shift instruction is:

SF SHIFT FLOATING
(Word index alignment)



If indirect addressing or indexing is called for in the instruction word, the effective virtual address is computed as for the instruction SHIFT (see page 44) except that bit position 23 of the effective virtual address determines the type of shift. If bit 23 is a 0, the contents of register R are treated as a short-format floating-point number; if bit 23 is a 1, the contents of registers R and Ru1 are treated as a long-format floating-point number.

The shift count, C, in bit positions 25 through 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.)

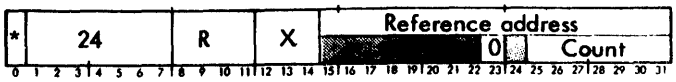
3. If the resultant fraction field is all 0's, the entire floating-point number is set to all 0's (true zero), regardless of the sign and the characteristic of the original number.

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

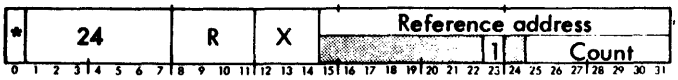
Floating Shift, Single Register



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 following two conversion instructions are provided by the SIGMA 7 computer:

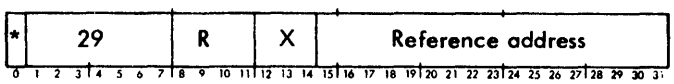
Instruction Name	Mnemonic
Convert by Addition	CVA
Convert by Subtraction	CVS

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 CPU 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 have been performed.

If an interrupt or memory 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 restarted (at the beginning of the instruction sequence) after the interrupt or 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 ⟶ A, 0 ⟶ n

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

If $(Ru1)_n = 0$, then n + 1 ⟶ n

If n < 32, repeat; otherwise, (A) ⟶ 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) ⟶ A, 0 ⟶ B, 0 ⟶ n

If (EWL + n) ≤ (A) then A - (EWL + n) ⟶ A,
1 ⟶ B_n, n + 1 ⟶ n

If (EWL + n) > (A) then n + 1 ⟶ n

If n < 32, repeat; otherwise, (A) ⟶ R, (B) ⟶ 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

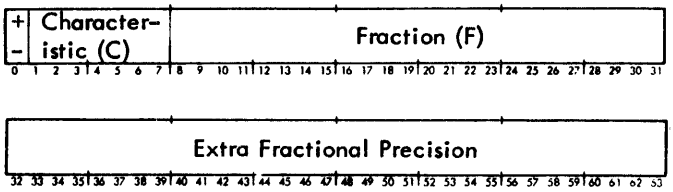
The following floating-point arithmetic instructions are available as optional SIGMA 7 instructions:

Instruction Name	Mnemonic
Floating Add Short	FAS
Floating Add Long	FAL
Floating Subtract Short	FSS
Floating Subtract Long	FSL
Floating Multiply Short	FMS
Floating Multiply Long	FML
Floating Divide Short	FDS
Floating Divide Long	FDL

FLOATING-POINT NUMBERS

SIGMA 7 accommodates two number formats 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 doubleword memory location or an even-odd pair of general registers.

A SIGMA 7 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 $0 \leq C \leq 127$

†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.

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 6 contains examples of floating-point numbers.

Modes of Operation

SIGMA 7 contains three mode control bits that are used to qualify floating-point operations. These mode control bits are identified as FS (floating significance), FZ (floating zero), and FN (floating normalize), and are contained in bit positions 5, 6, and 7, respectively, of the program status doubleword (PSD_{5-7}).

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

Instruction Name	Mnemonic	Page
Load Conditions and Floating Control	LCF	32
Load Conditions and Floating Control Immediate	LCFI	32
Load Program Status Doubleword	LPSD	72
Exchange Program Status Doubleword	XPSD	72

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

Instruction Name	Mnemonic	Page
Store Conditions and Floating Control	STCF	34
Exchange Program Status Doubleword	XPSD	72

Table 6. 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.				

UNIMPLEMENTED FLOATING-POINT INSTRUCTIONS

If the optional floating-point instruction set is not implemented in the computer and execution of a floating-point arithmetic instruction is attempted, the computer unconditionally aborts execution of the instruction (at the time of operation code decoding). The computer then traps to location X'41', with the contents of the condition code and all general registers unchanged. Location X'41' is the "unimplemented instruction" trap location.

FLOATING-POINT ADD AND SUBTRACT

The 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:

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 occur, 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 results 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 computer 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 signifiance 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 computer 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 computer executes the next instruction in sequence. (Exception: if characteristic underflow occurs with FS = 0, FZ determines the resultant action.)
- FS = 1 The computer 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 CPU 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 zero (FZ) mode control bit alone determines the operation of floating-point multiplication and division (if characteristic overflow does not occur and division by zero is not attempted) as follows:

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 computer 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 computer always traps to location X'44' with the general registers unchanged and the condition code set to 0100. If characteristic overflow occurs, the computer 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 LOATING-POINT INSTRUCTIONS

The condition code settings for floating-point instructions are summarized in Table 7. The following provisions apply to all floating-point instructions:

1. Underflow 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 and truncation are performed on absolute magnitudes. If the fraction is negative, then the two's complement is formed after shifting or truncation.

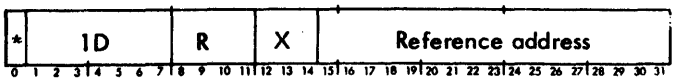
FAS FLOATING ADD SHORT
(Word index alignment, optional)

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. If no floating-point arithmetic fault occurs, the sum is loaded into register R as a short-format floating-point number.

Affected: (R), CC
(R) + EW ⟶ R

Traps: Unimplemented instruction, floating-point arithmetic fault

FAL FLOATING ADD LONG
(Doubleword index alignment, optional)



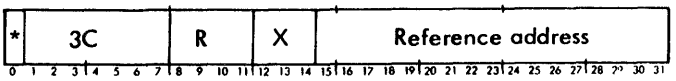
The effective doubleword and the contents of registers R and Ru1 are loaded into a set of internal registers.

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 not appended to the fractions, 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
(R, Ru1) + ED ⟶ R, Ru1

Traps: Unimplemented instruction, floating-point arithmetic fault

FSS FLOATING SUBTRACT SHORT
(Word index alignment, optional)



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
(R) - EW ⟶ R

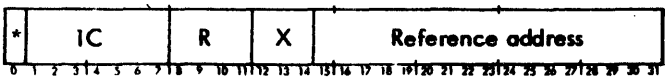
Traps: Unimplemented instruction, floating-point arithmetic fault

Table 7. Condition Code Settings for Floating-point Instructions

Condition Code 1	2	3	4	Meaning if no trap to location X'44' occurs	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	-A + A
③ 1	0	0	1	N < 0 (> 2 postnormalizing shifts)	N < 0 (> 2 postnormalizing shifts)
③ 1	0	1	0	N > 0 (> 2 postnormalizing shifts)	N > 0 (> 2 postnormalizing shifts)
				FS=0, FN=0, and no underflow	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

FSL FLOATING SUBTRACT LONG
(Doubleword index alignment, optional)



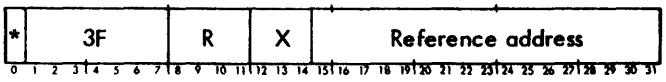
The effective doubleword and the contents of registers R and Ru1 are loaded into a set of internal registers.

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 long-format floating-point number.

Affected: (R), (Ru1), CC
(R, Ru1) − ED ⟶ R, Ru1

Traps: Unimplemented instruction, floating-point arithmetic fault

FMS FLOATING MULTIPLY SHORT
(Word index alignment, optional)



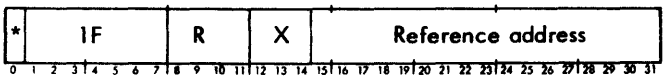
The effective word (multiplier) and the contents of register R (multiplicand) are loaded into a set of internal registers, and both numbers are then prenormalized (if necessary). The product of the fractions contains 12 hexadecimal digits. If no floating-point arithmetic fault occurs, the product is loaded into register R as a properly truncated short-format floating-point number.

The result of floating-multiply is always postnormalized. At most, one place of postnormalizing shift may be required. Truncation takes place after postnormalization.

Affected: (R), CC
(R) x EW ⟶ R

Traps: Unimplemented instruction, floating-point arithmetic fault

FML FLOATING MULTIPLY LONG
(Doubleword index alignment, optional)



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 multiplier and the multiplicand fractions are each 14 hexadecimal digits long, the product fraction is 28 hexadecimal digits long, and R must be an even value for correct results. If no floating-point arithmetic fault occurs, the postnormalized product is truncated to a long-format floating-point number and loaded into registers R and Ru1.

Affected: (R), (Ru1), CC
(R, Ru1) x ED ⟶ R, Ru1

Traps: Unimplemented instruction, floating-point arithmetic fault

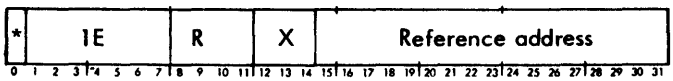
FDS FLOATING DIVIDE SHORT
(Word index alignment, optional)

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). FLOATING DIVIDE SHORT then forms a floating-point quotient with a 6-digit, normalized hexadecimal fraction. 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

Traps: Unimplemented instruction, floating-point arithmetic fault

FDL FLOATING DIVIDE LONG
(Doubleword index alignment, optional)



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 divisor, dividend, and quotient fractions are each 14 hexadecimal digits long, and 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

Traps: Unimplemented instruction, floating-point arithmetic fault

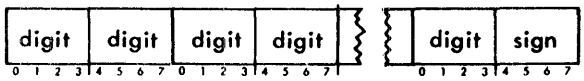
DECIMAL INSTRUCTIONS

The following instructions comprise the optional 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

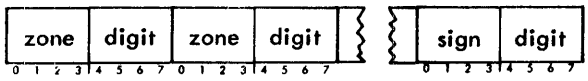
All SIGMA 7 decimal arithmetic instructions operate on packed decimal numbers, each consisting of from 1 to 31 decimal digits (in absolute form) plus a decimal sign. A decimal digit is a 4-bit code in the range 0000 through 1001, where 0000 = 0, 0001 = 1, 0010 = 2, 0011 = 3, 0100 = 4, 0101 = 5, 0110 = 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:



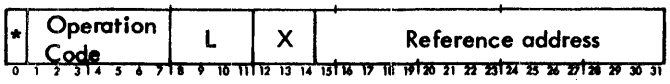
A decimal number can be converted from zoned to packed format by means of the instruction PACK DECIMAL DIGITS. A decimal number can be converted from packed to zoned format by means of the instruction UNPACK DECIMAL DIGITS.

DECIMAL ACCUMULATOR

All decimal arithmetic instructions imply the use of registers 12 through 15 of the current register bank 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 SIGMA 7 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.)

UNIMPLEMENTED DECIMAL INSTRUCTIONS

If the optional decimal arithmetic instructions described in this section are not implemented in a SIGMA 7 computer, the computer unconditionally aborts the execution of the instruction (at the time of operation code decoding), and traps to location X'41', which is the "unimplemented instruction" trap location.

ILLEGAL DIGIT AND SIGN DETECTION

Prior to executing any decimal instruction, the computer checks all decimal operands for the presence of illegal decimal digits or illegal decimal signs. For all decimal arithmetic instructions except DECIMAL MULTIPLY and DECIMAL DIVIDE, 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 effective decimal operand is checked for illegal digits or signs as above. However, the operand in the decimal accumulator is checked to verify that there is at least one legal decimal sign code somewhere in the number. (This type of check is a result of the interruptibility of these instructions, which may leave the decimal accumulator with a partially-completed result containing an internal sign code.) For these two instructions, 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'.

If an illegal digit or sign is detected, the computer 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 doubleword) is a 0, the computer then executes the next instruction in sequence; however, if the decimal arithmetic fault trap mask (PSD_{10}) is a 1, the computer traps to location X'45'. In either case, the contents of the decimal accumulator, the effective decimal operand, CC3, and CC4 remain unchanged.

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 computer 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 (PSD_{10}) is a 1, the computer traps to location X'45'; if the decimal arithmetic fault trap mask is a 0, the computer 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

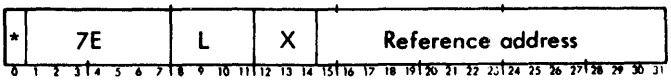
For the purpose of abbreviating the instruction descriptions to follow, the symbolic term "DECA" is used to represent the decimal accumulator, and the symbolic term "EDO" is used to represent the effective decimal operand of the instruction. For the instructions DECIMAL LOAD, DECIMAL ADD, DECIMAL SUBTRACT, 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 4 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 4 low-order bit positions; the remainder of the decimal accumulator contains at least one non-zero decimal digit.
1	0	positive – the decimal accumulator contains a positive decimal sign code in the 4 low-order bit positions; the remainder of the decimal accumulator contains at least one nonzero decimal digit.

DL DECIMAL LOAD
(Byte index alignment, optional)



If no illegal digit or illegal 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.

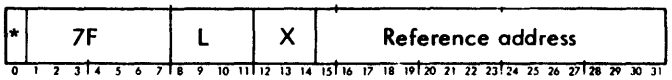
Affected: (DECA), CC
(EBL to EBL+L-1) ⟶ DECA

Traps: Unimplemented instruction, decimal arithmetic

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, optional)



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 nonzero 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

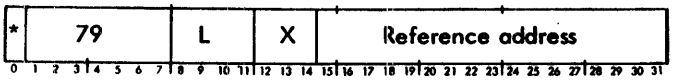
Traps: Unimplemented instruction, decimal arithmetic

(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, optional)



If no illegal digit or sign is detected in the effective decimal operand or in the decimal accumulator, DECIMAL ADD expands the effective decimal operand to 16 bytes (31 digits plus sign) by appending high-order 0's, algebraically adds the expanded decimal number to the contents of the entire decimal accumulator, and then loads the sum 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 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.

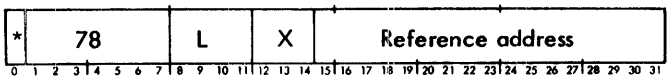
Affected: (DECA), CC
(DECA) + EDO ⟶ DECA

Traps: Unimplemented instruction, decimal arithmetic

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, optional)



If no illegal digit or sign is detected in the effective decimal operand or in the decimal accumulator, DECIMAL SUBTRACT expands the effective decimal operand to 16 bytes (31 digits plus sign) by appending high-order 0's, algebraically subtracts the expanded decimal number from the contents of the entire 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.

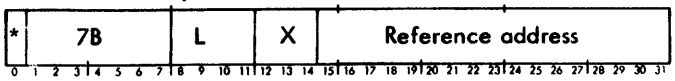
Affected: (DECA), CC
(DECA) - EDO ⟶ DECA

Traps: Unimplemented instruction, decimal arithmetic

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, optional, continue after interrupt)



If no illegal digit or sign is detected in the effective decimal operand and there is at least one decimal sign in the decimal accumulator, DECIMAL MULTIPLY multiplies the effective decimal operand (multiplicand) by the entire contents of the decimal accumulator (multiplier) and then loads the product into the decimal accumulator. If the result in the decimal accumulator is zero, the resulting sign is forced to the positive form.

No overflow can occur; however, an indeterminate result occurs (with an incorrect condition code indication, and with no trap activation) if any of the following conditions are not satisfied before the initial execution of DECIMAL MULTIPLY:

1. The 4 low-order bit positions of the decimal accumulator must contain the sign of the multiplier.
2. The 16 high-order digit positions of the decimal accumulator (i.e., general registers 12 and 13) must contain all 0's.
3. The effective decimal operand must not exceed 15 decimal digits (i.e., the value of L must not exceed 8). The illegal values of L are X'0' and the range X'9' to X'F'. An illegally coded L field is recognized as an illegal digit or sign and the instruction is aborted.

This instruction can be interrupted during the course of its execution, and then be resumed, without producing an erroneous product (provided that the contents of the decimal accumulator are not altered between the interruption and continuation). Actually, the instruction is reexecuted, but since there is no initializing phase, it begins with the same iteration that was started prior to the interrupt.

Affected: (DECA), CC
(DECA) x EDO ⟶ DECA

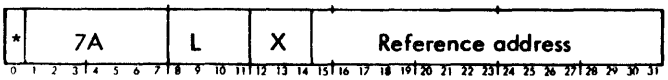
Traps: Unimplemented instruction, decimal arithmetic

Condition code settings:

1	2	3	4	Result in DECA
1	0	-	-	illegal digit or sign detected, instruction aborted

1	2	3	4	Result in DECA	
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, optional, 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 computer loads the quotient (15 decimal digits plus sign) into the 8 low-order bytes of the decimal accumulator (registers 14 and 15), and loads the remainder (also 15 decimal digits plus sign) into the 8 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 can occur 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.
3. The effective decimal operand must not exceed 15 decimal digits (i.e., the value of L must not exceed 8). The illegal values of L are X'0' and the range X'9' to X'F'. An illegally coded L field is recognized as an illegal digit or sign and the instruction is aborted.

This instruction can be interrupted during the course of its execution, and can then be resumed without producing an erroneous result (provided that the contents of the decimal accumulator are not altered between interruption and continuation). Actually, the instruction is reexecuted, but since there is no initializing phase, it begins with the same iteration that was started prior to the interrupt.

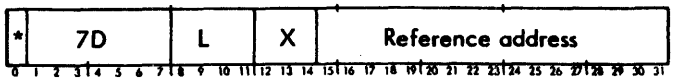
Affected: (DECA), CC
(DECA) ÷ EDO ⟶

Traps: Unimplemented instruction, decimal arithmetic

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, optional)



If there is no illegal digit or illegal 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).

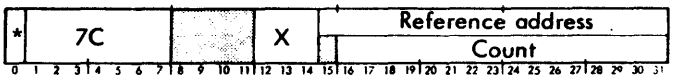
Affected: CC
(DECA) : EDO

Traps: Unimplemented instruction, decimal arithmetic

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, optional)



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 $2^{-15} \leq C \leq 2^{15}-1$, a shift account 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 Traps: Unimplemented instruction, 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	-	-	1 or more nonzero digit(s) shifted out of DECA on left shift	

PACK PACK DECIMAL DIGITS
(Byte index alignment, optional, continue after interrupt)

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 forming 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).

This instruction can be interrupted during the course of its execution, and can then be resumed without producing an erroneous result (provided that the contents of the decimal accumulator are not altered between interruption and continuation). Actually, the instruction is re-executed, but since there is no initializing phase, it begins with the same iteration that was started prior to the interrupt.

Affected: (DECA), CC Traps: Unimplemented instruction, 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	X'F0F1F2F3
		F4F5F6F7	F4F5F6F7
		F8F9F0'	F8F9F0'
(DECA)	=	xxxxxxxx	X'00000000
		xxxxxxxx	00000000
		xxxxxxxx	00000123
		xxxxxxxx	4567890C'
CC	=	xxxx	0010

Example 2, L = 6:

EDO	=	X'000938F7	X'000938F7
		E655B483	E655B483
		02F1B0'	02F1B0'
(DECA)	=	xxxxxxxx	X'00000000
		xxxxxxxx	00000000
		xxxxxxxx	00000987
		xxxxxxxx	6543210D'
CC	=	xxxx	0001

UNPK UNPACK DECIMAL DIGITS
(Byte index alignment, optional, 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 4 low-order bit positions of the decimal accumulator are used to select the sign code for the last digit of the string; a zone of 1111 (X'F') is used for all other digits. 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.

This instruction can be interrupted during the course of its execution, and can then be resumed without producing an erroneous result (provided that the contents of the decimal accumulator are not altered between interruption and continuation). Actually, the instruction is re-executed, but since there is no initializing phase, it begins with the same iteration that was started prior to the interrupt.

Affected: (EBL to EBL + 2L -2), CC1, CC2 Traps: Unimplemented instruction, 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	X'00000000
		00000001	00000001
		23456789	23456789
		0123456D'	0123456D'
EDO	=	xxxxxxxx	X'F0F0F0F1
		xxxxxxxx	F2F3F4F5
		xxxxxxxx	F6F7F8F9
		xxxxxxxx	F0F1F2F3
		xxxxxx	F4F5D6'
CC	=	xxxx	00xx

Example 2, L = 8:

(DECA)	=	X'00000000	X'00000000
		23000000	23000000
		10001234	10001234
		0012345C'	0012345C'
EDO	=	xxxxxxxx	X'F1F0F0F0
		xxxxxxxx	F1F2F3F4
		xxxxxxxx	F0F0F1F2
		xxxxxx	F3F4C5'
CC	=	xxxx	01xx

Example 3, L = 4:

(DECA)	=	X'00001001	X'00001001
		00001002	00001002
		00001003	00001003
		0001004F'	0001004F'
EDO	=	xxxxxxxx	X'F0F0F0F1
		xxxxxxxx	F0F0C4'
CC	=	xxxx	01xx

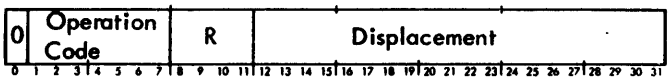
BYTE STRING INSTRUCTIONS

Five instructions provide for the manipulation of strings of consecutive bytes. Four of these instructions are standard with the SIGMA 7 computer, and one additional instruction (EDIT BYTE STRING) is provided with the decimal option. The byte string instructions and their mnemonic codes are as follows:

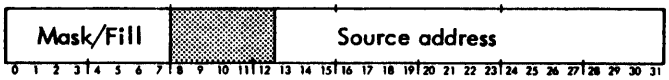
Instruction Name	Mnemonic
Move Byte String	MBS
Compare Byte String	CBS
Translate Byte String	TBS
Translate and Test Byte String	TTBS
Edit Byte String (optional)	EBS

These instructions are in the immediate displacement class, are memory-to-memory operations, and proceed one byte at a time (except for the instruction MOVE BYTE STRING, which proceeds four bytes at a time under certain conditions). These operations are under the control of information that must be loaded into certain general registers before the instruction is executed; hence, they may be interrupted after any individual byte operation. The general format for the information in the instruction word and in the general registers is as follows:

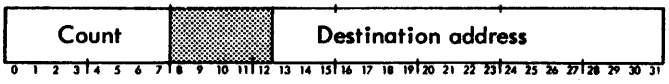
Instruction word:



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 computer 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 bank.
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.

Designation	Function
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 registers R or Ru1 that is not explicitly defined (i.e., in the shaded part of the register diagram for the instruction) should be coded with zeros.

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.

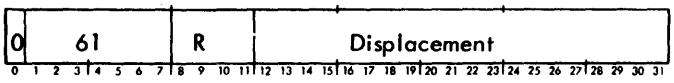
In the descriptions of the byte-string instructions, the following abbreviations and terms are used:

D	Displacement, $(I)_{12-31}$
SA	Source address, $(R)_{13-31}$
ESA	Effective source address, $[(R)_{13-31}+(I)_{12-31}]_{13-31}$

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 nonzero) or is 1 byte 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.

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).

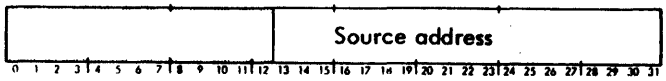
Affected: (DBS), (R), (Ru1)
(SBS) ⟶ DBS

If MBS is indirectly addressed, it is treated as a nonexistent instruction, 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 contents of register R and the destination byte string unchanged.

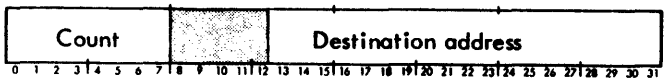
A speed advantage can be gained in the MBS instruction if the source and destination byte strings both begin on the same byte within their respective words. This allows all bytes (except possibly the first few bytes and the last few bytes to be moved in fullword units.

Case I: even, nonzero R field (Ru1=R+1)

Contents of register R:



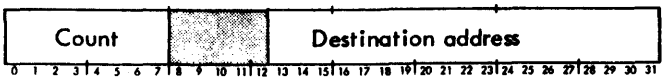
Contents of register R+1:



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)

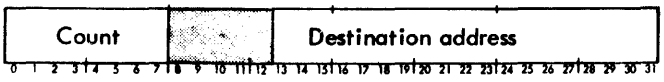
Contents of register R:



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.

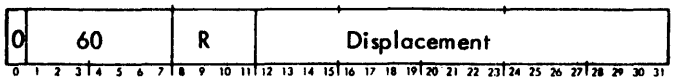
Case III: zero R field (Ru1=1)

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 terminates, 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)

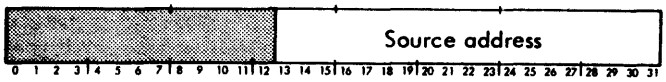
Condition code settings:

1	2	3	4	Result of CBS
-	-	0	0	source byte string equals destination byte string
-	-	0	1	source byte string less than destination byte string
-	-	1	0	source byte string greater than destination byte string

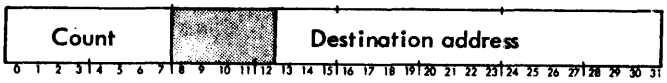
If CBS is indirectly addressed, it is treated as a nonexistent instruction, in which case the computer unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the destination byte string unchanged.

Case I: even, nonzero R field (Ru1=R+1)

Contents of register R



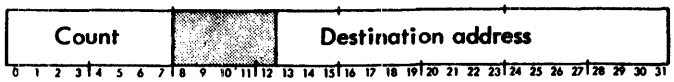
Contents of register R+1



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)

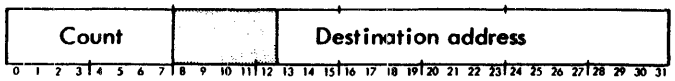
Contents of register R



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.

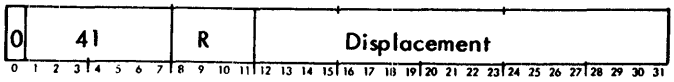
Case III: zero R field (Ru1=1)

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:

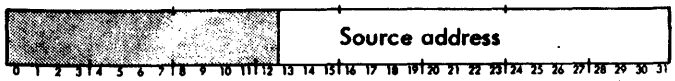
1. The displacement in bit positions 12-31 of the TBS instruction
2. The current contents of bit positions 13-31 of register R (source address)
3. The numeric value of the current destination byte, the 8-bit contents of the byte location pointed to by the current destination address in bit positions 13-31 of register (Ru1)

Affected: (DBS),(Ru1)
translated (DBS) ⟶ DBS

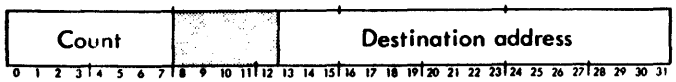
If TBS is indirectly addressed, it is treated as a nonexistent instruction, in which case the computer unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the destination byte string unchanged.

Case I: even, nonzero R field (Ru1=R+1)

Contents of register R



Contents of register R+1



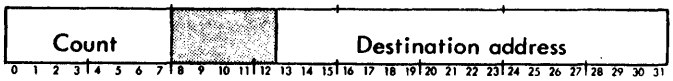
The destination byte string begins with the byte location pointed to by the destination address in register R + 1 and is C bytes in length. The source byte string (translation table) begins with the byte location pointed to by the displacement in TBS plus the source address in register R. When the instruction 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 results of the instruction are unpredictable when an odd-numbered general register is specified by the R field of the instruction word.

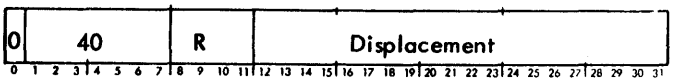
Case III: zero R field (Ru1=1)

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

If translated (SBS) n mask ≠ 0, translated (SBS) n mask ⟶ mask and stop

If translated (SBS) n mask = 0, continue

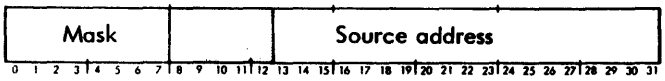
Condition code settings:

1	2	3	4	Result of TTBS
-	-	-	0	translation bytes and the mask do not compare ones anyplace
-	-	-	1	the last translation byte compared with the mask contained at least one 1 corresponding to a 1 in the mask

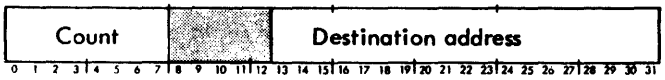
If TTBS is indirectly addressed, it is treated as a nonexistent instruction, in which case the computer unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40' with the contents of register R and the destination byte string unchanged.

Case I: even, nonzero R field (Ru1=R+1)

Contents of register R



Contents of register R+1



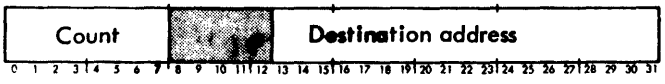
The destination byte string begins with the byte location pointed to by the destination address in register R+1 and is C bytes in length. The source byte string (translation table) begins with the byte location pointed to by the displacement in TTBS plus the source address in register R.

Case II: odd R field

Because of the interruptible nature of TRANSLATE AND TEST BYTE STRING, the results of the instruction are unpredictable when an odd-numbered general register is specified by the R field of the instruction word.

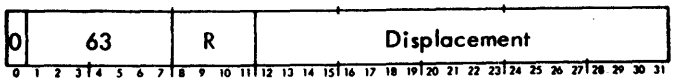
Case III: zero R field (Ru1=1)

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, that register 0 provides all 0's as its contents.)

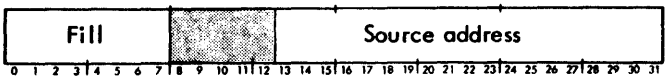
EBS EDIT BYTE STRING
(Immediate displacement, optional, continue after interrupt)



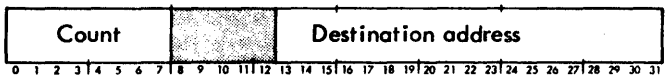
EDIT BYTE STRING converts a decimal information field from packed decimal format to zoned decimal format, under control of the editing pattern in the destination byte string, and replaces the destination byte string with the edited, zoned result. (See page 52 for a description of packed and zoned decimal formats.) EBS proceeds 1 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 positions 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.)

The results of EBS are unpredictable if the R field of EBS is an odd value, or if the R field of EBS is 0.

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 (character code X'40') replaces the byte in the destination byte string
2. the decimal digit is expanded to zoned decimal format (by generating X'Fd', where d is the decimal digit) 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), 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 (X'40') 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 computer executes the next instruction in sequence.

The detailed operation of EDIT BYTE STRING is as given below.

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 on the next page will help clarify the discussion below.

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 computer 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 computer 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 computer 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 (because CC4 = 1 means significance already encountered	None
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 step 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 CC4 = 0	store fill character in pattern byte location
CC1 = 1 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, 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: Unimplemented instruction, decimal arithmetic

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, 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 contents of register R, register Ru1, register 1, the destination byte string, and the condition code unchanged.

If the decimal instruction set is not implemented, the computer unconditionally aborts execution of EBS (at the time of operation code decoding) and traps to location X'41' with the condition code, the contents of register R, register Ru1, register 1, and the destination byte string unchanged.

If an illegal digit or sign is detected in the decimal information field, the computer 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 R+1, 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.

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 b is used to re represent the character blank (X'40').

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 b 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:

* * * * * * . 0 0 b b b

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 , 5 4 3 . 2 1 b C R

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

* * * 5 4 3 . 2 1 b b b

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:

6 1 2 . 5 0 0 # # # 1 2 . 3 4 b # # 0 3 5 END

The new condition code is: 1011

The new contents of register 1 are: X'xxx02013'

PUSH-DOWN INSTRUCTIONS

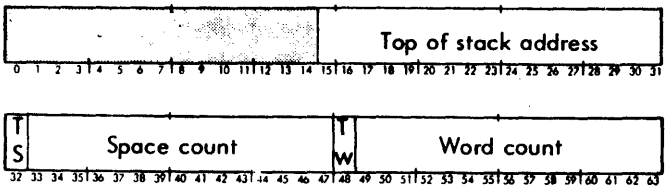
The term "push-down processing" refers to the programming technique (used extensively in recursive routines) of storing the context of a calculation in memory, proceeding with a new set of information, and then activating the previously stored information. Typically, this process involves a reserved 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 SIGMA 7 computer

provides for simplified and efficient programming of push-down processing by means of the following instructions:

Instruction Name	Mnemonic
Push Word	PSW
Pull Word	PLW
Push Multiple	PSM
Pull Multiple	PLM
Modify Stack Pointer	MSP

STACK POINTER DOUBLEWORD

Each of these instructions operates with respect to a memory stack that is defined by a doubleword located at the effective address of the instruction. This doubleword, referred to as a stack pointer doubleword (SPD), has the following structure:



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 previous 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 push-down instructions, the space count and the word count are each tested to determine whether or not 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 computer unconditionally aborts execution of the instruction, with the stack, the stack pointer doubleword, and the contents of general registers unchanged. Ordinarily, the computer 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 or not the computer is to 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 computer traps to location X'42' with the condition code unchanged.
1	Instead of trapping to location X'42', the computer 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 or not the computer is to trap 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 computer traps to location X'42' with the condition code unchanged.
1	Instead of trapping to location X'42', the computer 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 computer 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})

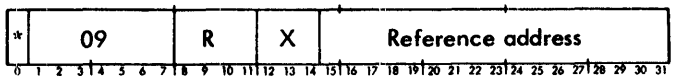
1	2	3	4	Reason for abort
1	-	1	-	impending overflow of word count and underflow of space count on a push operation or impending overflow of space count and underflow of word count on a pull operation. The push-down stack limit trap was inhibited by both the TW and the TS bits

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 computer 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 or not the space and word counts were actually modified by the instruction. In the following descriptions of the push-down instruction, only those condition codes are given that can actually be produced by the instruction, provided the computer 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.

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

$(SPD)_{15-31} + 1 \longrightarrow SPD_{15-31}$

$(R) \longrightarrow (SPD_{15-31})$

$(SPD)_{33-47} - 1 \longrightarrow SPD_{33-47}$

$(SPD)_{49-63} + 1 \longrightarrow SPD_{49-63}$

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	instruction completed
0	0	1	0	word count = $2^{15}-1$, TW = 1	instruction aborted
1	1	0	0	space count = 0, TS = 1	instruction aborted
1	1	0	1	space count = 0, word count = 0, TS = 1	instruction aborted
1	1	1	0	word count = $2^{15}-1$, space count = 0, TW = 1, and TS = 1	instruction aborted

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}).

2. The current top-of-stack address is decremented by 1, to point to the new top-of-stack location.

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

$((SPD)_{15-31}) \longrightarrow R$; $(SPD)_{15-31} - 1 \longrightarrow SPD_{15-31}$

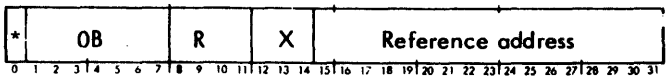
$(SPD)_{33-47} + 1 \longrightarrow SPD_{33-47}$

$(SPD)_{49-63} - 1 \longrightarrow SPD_{49-63}$

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 is assumed to 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 into 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 an ascending sequence, beginning with the location pointed to by the current top-of-stack address (SPD_{15-31}) 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

$(R) \longrightarrow (SPD)_{15-31} + 1 \ldots (R+CC-1) \longrightarrow (SPD)_{15-31}+CC$

$(SPD)_{15-31}+CC \longrightarrow SPD_{15-31}$

$(SPD)_{33-47}-CC \longrightarrow SPD_{33-47}$

$(SPD)_{49-63}+CC \longrightarrow SPD_{49-63}$

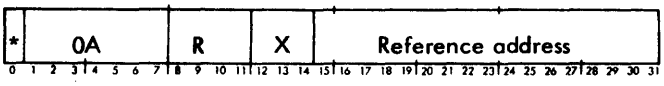
Condition code settings:

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	

If the instruction starts storing words into an accessible region of memory and then crosses into an inaccessible memory region, either the memory protection trap or the nonexistent memory address trap can occur. In either case, the trap is activated with the condition code unchanged from the value it contained before the execution of PSM. The effective address of the instruction permits the trap routine to compute how many words of memory have been changed. Since it is permissible to use indirect addressing through one of the affected locations, or even to execute an instruction located in one of the affected locations; a trapped PSM instruction may have already overwritten the direct address, or the PSM instruction itself, thus destroying any possibility of continuing the program successfully. If such programming must be done, it is advisable that the direct address, or the PSM instruction, occupy the last location in which the contents of a register are to be stored by the PSM instruction.

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 core 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 is assumed to 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 become 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 a descending sequence, beginning with the contents of the location pointed to by the current top-of-stack address (SPD_{15-31}) 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

$((SPD)_{15-31}) \longrightarrow R+CC-1, \ldots,$

$((SPD)_{15-31} - |CC-1|) \longrightarrow R$

$(SPD)_{15-31} - CC \longrightarrow SPD_{15-31}$

$(SPD)_{33-47} + CC \longrightarrow SPD_{33-47}$

$(SPD)_{49-63} - CC \longrightarrow SPD_{49-63}$

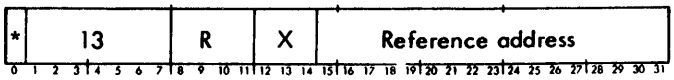
Condition code settings:

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	

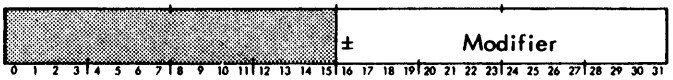
If the instruction starts loading from an existent region of memory and then crosses a memory page boundary into an inaccessible memory region, either the memory protection trap or the nonexistent memory address trap can occur. In either case, the trap is activated with the condition code unchanged from the value it contained before the execution of PLM. The effective address of the instruction permits the trap routine to compute how many registers have been loaded. Since it is permissible to use indexing or indirect addressing through a general register, or even to execute an instruction located in a general register, a trapped PLM instruction may have already overwritten the index, direct address, or the PLM instruction itself, thus destroying any possibility of continuing the program successfully. If such programming must be done, it is advisable that the register containing the direct address, index displacement, or instruction be the last register loaded by the PLM instruction.

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 core 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 is assumed to 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 computer 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 $(SPD)_{15-31}$, 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

$(SPD)_{15-31} + (R)_{16-31SE} \longrightarrow SPD_{15-31}$

$(SPD)_{33-47} - (R)_{16-31} \longrightarrow SPD_{33-47}$

$(SPD)_{49-63} + (R)_{16-31} \longrightarrow SPD_{49-63}$

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	

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 count and word count
-	-	-	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 $\leq 2^{15}-1$
1	-	-	-	space count - modifier < 0, and TS = 1 or space count - modifier > $2^{15}-1$ TS = 1

EXECUTE/BRANCH INSTRUCTIONS

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.

Prior to the time that an instruction is accessed from memory for execution, bit positions 15-31 of the program status doubleword contain the virtual address of the instruction, referred to as the instruction address. At this time, the computer 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 doubleword 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 computer 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. Thus, during execution of a branch instruction, the updated instruction address is decremented, unchanged, or replaced, as determined by the following critera:

1. Trap condition. A nonallowed operation trap condition can occur during execution of a branch instruction, but only if an attempt is made to access either a nonexistent memory address or an address that is not available to the slave program for instruction access. The trap condition occurs in the following situations:

   a. The branch instruction is indirectly addressed, but the address of the location containing the direct address is either nonexistent or unavailable to the slave program for read access.

   b. 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 unavailable to the slave program for instruction access.

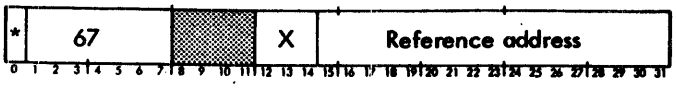
   c. The effective address of any branch instruction (conditional or unconditional) is nonexistent.

   If any of the above situations occur, the computer aborts execution of the branch instruction, decrements the updated instruction address by 1, and traps to location X'40'. In this case, the instruction address value (IA) stored by the XPSD instruction in location X'40' is the address of the aborted branch instruction.

2. No branch condition. If the branch instruction is conditional, the condition for the branch is not satisfied, and no trap condition occurs, the updated instruction address remains unchanged. Then, instruction execution proceeds with the instruction pointed to by the updated instruction address.

3. Branch condition. If the branch instruction is unconditional (or if the branch instruction is conditional and the condition for the branch is satisfied) and no trap condition occurs, the updated instruction address is replaced by the effective virtual address of the branch instruction. Then, instruction execution proceeds with the instruction pointed to by the effective virtual address of the branch instruction.

EXU EXECUTE
(Word index alignment)



EXECUTE causes the computer 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 computer 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 computer processes the interrupt-servicing routine for the active interrupt level and then returns program control to the EXU instruction (or the intial 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.

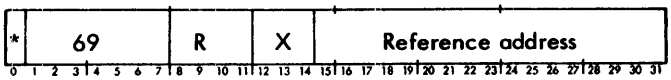
If a trap condition occurs between the beginning of an EXU instruction (or chain of EXU instructions) and the completion of the subject instruction, the computer traps to the appropriate trap location. The instruction address stored by the XPSD instruction in the trap location is the address of the EXU 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.

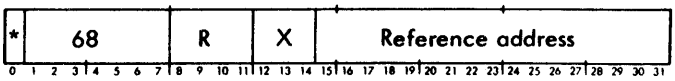
Affected: (IA) if CC n R $\neq$ 0

If CC n $(I)_{8-11} \neq 0$, $EVA_{15-31} \longrightarrow$ IA

If CC n $(I)_{8-11} = 0$, 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.

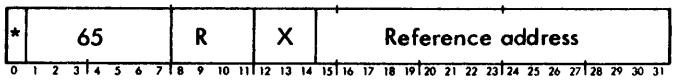
Affected: (IA) if CC n R = 0

If CC n $(I)_{8-11} = 0$, $EVA_{15-31} \longrightarrow$ IA

IF CC n $(I)_{8-11} \neq 0$, 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 (EVA) 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

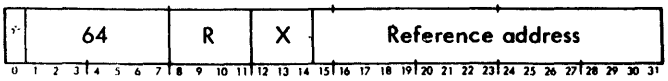
If $(R)_0 = 1$, $EVA_{15-31} \longrightarrow$ IA

If $(R)_0 = 0$, IA not affected

If the effective address of BIR is unavailable to the slave program for instruction access and the branch condition is satisfied, or if the effective address of BIR is nonexistent,

the computer 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 computer traps because of instruction access protection, register R will contain the value that existed just before the BIR instruction.

BDR BRANCH ON DECREMENTING REGISTER
(Word index alignment)



BRANCH ON DECREMENTING REGISTER computes the effective virtual address (EVA) 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)

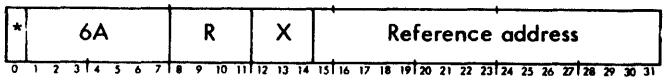
(R) - 1 ⟶ R

If $(R)_0 = 0$ and $(R)_{1-31} \neq 0$, EVA_{15-31} ⟶ IA

if $(R)_0 = 1$ or (R) = 0, IA not affected

If the effective address of BDR is unavailable to the slave program for instruction access and the branch condition is satisfied, or if the effective address of BDR is nonexistent, the computer 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 computer traps because of instruction access protection, register R will contain the value that existed just before the BDR instruction.

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.

Affected: (R), (IA)

IA ⟶ R_{15-31}; 0 ⟶ R_{0-14}; EVA_{15-31} ⟶ IA

If the effective address of BAL is either nonexistent or is unavailable to the slave program for instruction access, the computer aborts execution of the BAL instruction (after loading the updated instruction address into register R) 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 BAL instruction.

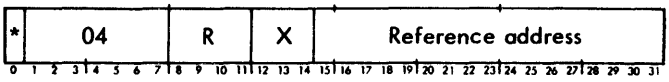
CALL INSTRUCTIONS

Each of the four call instructions causes the computer to trap to a specific location for the next instruction in sequence. The four call instructions, their mnemonics, and the locations to which the computer 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'

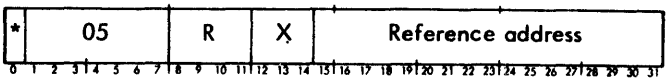
Each of these four trap locations must contain an EXCHANGE PROGRAM STATUS DOUBLEWORD (XPSD) instruction. Execution of XPSD in the trap location for a call instruction is described on page 74. If the XPSD instruction is coded with bit position 9 set to 1, the next instruction (executed after the XPSD) 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 computer to branch to a specific routine; thus, the four call instructions can be used to enter any of as many as 64 unique routines.

CAL1 CALL 1
(Word index alignment)



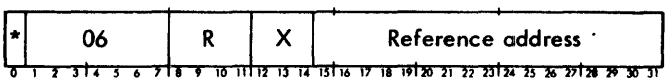
CALL 1 causes the computer to trap to location X'48'.

CAL2 CALL 2
(Word index alignment)



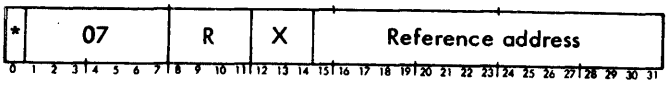
CALL 2 causes the computer to trap to location X'49'.

CAL3 CALL 3
(Word index alignment)



CALL 3 causes the computer to trap to location X'4A'.

CAL4 CALL 4
(Word index alignment)



CALL 4 causes the computer to trap to location X'4B'.

CONTROL INSTRUCTIONS

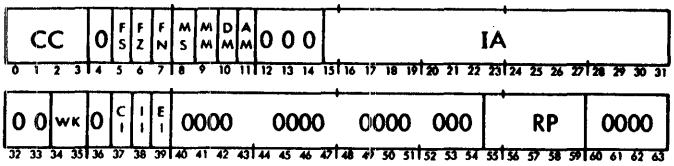
The following privileged instructions are used to control the basic operating conditions of the SIGMA 7 computer:

Instruction Name	Mnemonic
Load Program Status Doubleword	LPSD
Exchange Program Status Doubleword	XPSD
Load Register Pointer	LRP
Move to Memory Control	MMC
Wait	WAIT
Read Direct	RD
Write Direct	WD

If execution of any control instruction is attempted while the computer is in the slave mode (i.e., while bit 8 of the current program status doubleword is a 1), the computer unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40'.

PROGRAM STATUS DOUBLEWORD

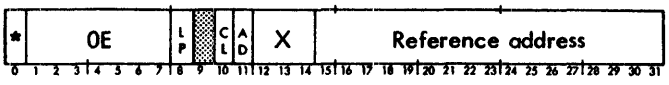
The SIGMA 7 program status doubleword has the following structure when stored in memory:



Bit Position	Designation	Function
0-3	CC	Condition code
5	FS	Floating significance mask
6	FZ	Floating zero mask
7	FN	Floating normalize mask
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
34, 35	WK	Write key
37	CI	Counter interrupt group inhibit
38	II	I/O interrupt group inhibit
39	EI	External interrupt inhibit
55-59	RP	Register pointer

The detailed functions of the various portions of the SIGMA 7 program status doubleword are described on page 15.

LPSD LOAD PROGRAM STATUS DOUBLEWORD
(Doubleword index alignment, privileged)



LOAD PROGRAM STATUS DOUBLEWORD replaces bits 0 through 39 of the current program status doubleword with bits 0 through 39 of the effective doubleword. The following conditional operations are also performed:

1. If bit position 8 (LP) of LPSD contains a 1, bits 55 through 59 of the current program status doubleword (register pointer) are replaced by bits 55 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 of LPSD (AD) is a 1, or is disarmed if bit 11 of LPSD is 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. (Interrupt levels are described in detail on page 18.)

Those portions of the effective doubleword that correspond to undefined fields in the program status doubleword are ignored.

Affected: (PSD), interrupt system if $(I)_{10} = 1$

$ED_{0-3} \longrightarrow CC$; $ED_{5-7} \longrightarrow FS, FZ, FN$

$ED_{8} \longrightarrow MS$; $ED_{9} \longrightarrow MM$

$ED_{10} \longrightarrow DM$; $ED_{11} \longrightarrow AM$

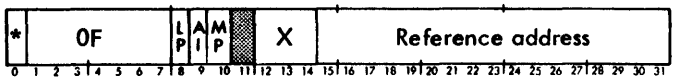
$ED_{15-31} \longrightarrow IA$; $ED_{34-35} \longrightarrow WK$

$ED_{37-39} \longrightarrow CI, II, EI$; If $(I)_8 = 1$, $ED_{55-59} \longrightarrow RP$

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 DOUBLEWORD
(Doubleword index alignment, privileged)



EXCHANGE PROGRAM STATUS DOUBLEWORD stores the entire program status doubleword and then replaces the current program status doubleword with a new program status doubleword.

Use of the memory map in interpreting the XPSD instruction address depends on the combined settings of bit 9 of the current PSD and bit 10 of the XPSD instruction, and on whether or not the XPSD is executed in an interrupt or trap location as the result of an interrupt or trap:

1. If the XPSD instruction is executed in an interrupt or trap location, the map is used to interpret the indirect reference address and the effective address if, and only if, a 1 is contained in bit positions 9 (MM) of the current PSD and 10 (MP) of XPSD.

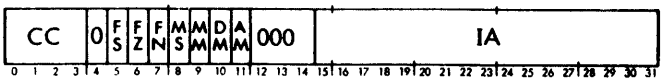
2. The same logic applies with one exception when the instruction is not executed in an interrupt or trap location. The exception is that if the program is in the mapping mode ($PSD_9 = 1$), the map is used to interpret the indirect reference address regardless of the state of $XPSD_{10}$.

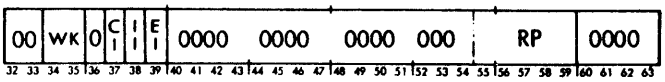
These conditions are summarized in the truth table shown below. General information on memory addressing is contained in Chapter 2 under "Memory Control Storage", "Memory Reference Addresses", and "Memory Address Control".

$XPSD_{10}$	PSD_9	XPSD Address Type	Map?	
1	1	Ind. Ref. Addr.	yes	
		Effect. Addr.	yes	
	0	Ind. Ref. Addr.	no	
		Effect. Addr.	no	
0	1	Ind. Ref. Addr.	no	yes†
		Effect. Addr.	no	
	0	Ind. Ref. Addr.	no	
		Effect. Addr.	no	

†"Yes" only if XPSD not executed in an interrupt or trap location.

The current program status doubleword is stored in the doubleword location pointed to by the effective address of XPSD in the following form:





The current program status doubleword is replaced by a new program status doubleword as follows:

1. The effective address of XPSD is incremented by 2, so that it points to the next doubleword location. The address thus generated is subject to the same mapping consideration as the original effective address (i.e., mapping is performed with the new address if bit 10 of XPSD is a 1 and bit 9 of the current program status doubleword is also a 1; otherwise, mapping is not performed). The contents of the next doubleword location are referred to as the second effective doubleword, or ED2.

2. Bits 0 through 35 of the current program status doubleword are unconditionally replaced by bits 0 through 35 of the second effective doubleword. The affected portions of the program status doubleword are:

Bit Position	Designation	Function
0-3	CC	Condition code
5-7	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
34-35	WK	Write key

3. A logical inclusive OR is performed between bits 37 through 39 of the current program status doubleword 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 doubleword 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 doubleword are set to 1's. See page 19 for a detailed discussion of the interrupt inhibits.

4. If bit position 8 (LP) of XPSD contains a 1, bits 55-59 of the current program status doubleword (register pointer) are replaced by bits 55 through 59 of the second effective doubleword; if bit 8 of XPSD is a 0, the current register pointer value remains unchanged.

The following additional operations are performed on the new program status doubleword 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 doubleword (CC1) is set to 1. Then, if bit 9 (AI) of XPSD is a 1, bit positions 15-31 of the new program status doubleword (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 doubleword (CC2) is set to 1. Then, if bit 9 of XPSD is a 1, the instruction address portion of the new program status doubleword 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 computer is in the slave mode, bit position 2 of the new program status doubleword (CC3) is set to 1. Then, if bit position 9 of XPSD is 1, the instruction address portion of the new program status doubleword 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 doubleword (CC4) is set to 1. Then, if bit 9 of XPSD is a 1, the instruction address portion of the new program status doubleword 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', X'0D', X'2C', and X'2D'. If any one of these operation codes is used as an instruction while the computer is in the slave 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 doubleword 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 doubleword 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 PSD.

   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 PSD.

If bit position 9 of XPSD contains a 0, the instruction address portion of the new PSD 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 trap or a call instruction 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.

Affected: (EDL), (PSD)

If $(I)_{10} = 1$, effective address is virtual

If $(I)_{10} = 0$, effective address is actual

PSD ⟶ EDL

$ED2_{0-3}$ ⟶ CC; $ED2_{5-7}$ ⟶ FS, FZ, FN

$ED2_{8}$ ⟶ MS; $ED2_{9}$ ⟶ MM

$ED2_{10}$ ⟶ DM; $ED2_{11}$ ⟶ AM

$ED2_{15-31}$ ⟶ IA; $ED2_{34-35}$ ⟶ WK

$ED2_{37-39}$ u CI, II, EI ⟶ CI, II, EI

If $(I)_8 = 1$, $ED2_{55-59}$ ⟶ RP

If $(I)_8 = 0$, RP not affected

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

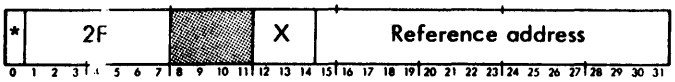
If privileged instruction violation, 1 ⟶ CC3 then, if $(I)_9 = 1$, IA + 2 ⟶ IA

If memory protection violation, 1 ⟶ CC4 then, if $(I)_9 = 1$, IA + 1 ⟶ IA

If call instruction, CC u $CALL_{8-11}$ ⟶ CC then, if $(I)_9 = 1$, IA + $CALL_{8-11}$ ⟶ IA

If $(I)_9 = 0$, IA not affected

LRP LOAD REGISTER POINTER
(Word index alignment, privileged)

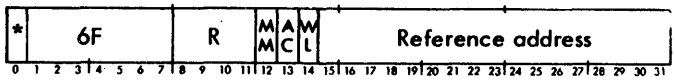


LOAD REGISTER POINTER loads bits 23 through 27 of the effective word into the register pointer (RP) portion of the current program status doubleword. Bit positions 0 through 22 and 28 through 31 of the effective word are ignored, and no other portion of the program status doubleword is affected. If the register pointer is loaded with a value that points to a nonexistent block of general registers, the computer subsequently generates either all 1's or all 0's as the contents of the nonexistent block of general registers, whenever an instruction designates a general register by means of the R field or the reference address field.

Affected: RP

EW_{23-27} ⟶ RP

MMC MOVE TO MEMORY CONTROL
(Word index alignment, privileged, continue after interrupt)



MOVE TO MEMORY CONTROL loads a string of one or more words into one of the three blocks of memory control registers (memory control registers are described on page 12). Bit positions 12-14 of MMC are not used as an index register address; instead, they are used to specify which block of memory control registers is to be loaded, as follows:

Bit position 12	13	14	Function
1	0	0	Load memory map block addresses
0	1	0	Load access protection
0	0	1	Load memory write protection locks

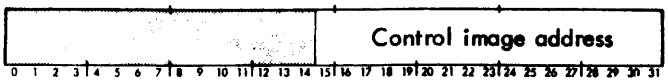
If bit positions 12-14 of MMC contain either all 0's or more than a single 1, the instruction produces an undefined result. Also, if an attempt is made to load unimplemented memory control storage, the contents of the general registers specified by the MMC instruction are undefined at the completion of the instruction, and the implemented memory control storage (if any) is not affected.

Bit positions 15-31 (reference address field) of MMC are ignored insofar as the operation of the instruction is concerned, and the results of the instruction are the same whether or not MMC is indirectly addressed.

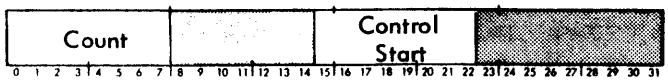
The R field of MMC designates an even-odd pair of general registers (R and Ru1) that are used to control the loading of

the specified bank of memory control registers. Registers R and Ru1 are assumed to contain the following information:

Register R:



Register Ru1:



Bit positions 15 through 31 of register R contain the virtual address of the first word of the control image to be loaded into the specified block of memory control registers. Bit positions 0 through 7 of register Ru1 contain a count of the number of words to be loaded. If bits 0-7 of register Ru1 are initially all 0's, a word count of 256 is implied.)

Bit positions 15 through 22 of register Ru1 point to the beginning of the memory region controlled by the registers to be loaded. The significance of this field is different for the 3 modes of MMC.

The R field of the MMC instruction must be an even value for proper operation of the instruction; if the R field of MMC is an odd value, the operation of the instruction is undefined.

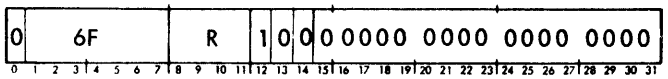
If MCC is indirectly addressed and the indirect reference address is nonexistent, the nonallowed operation trap (location X'40') is activated. The effective virtual address of the MMC instruction however, is not used as a memory reference (thus does not affect the normal operation of the instruction).

Affected: (R),(Ru1), memory control storage

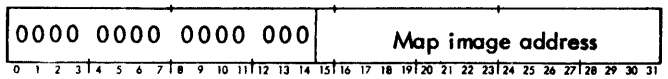
LOADING THE MEMORY MAP

The following diagrams represent the configuration of MMC, register R, and register Ru1 that are required to load the memory map:

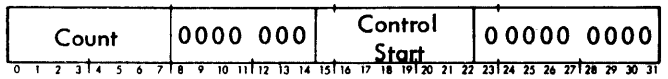
The instruction format is:



The contents of register R are:



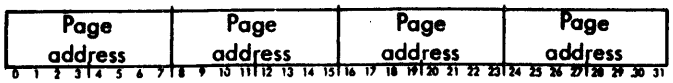
The contents of register Ru1 are:



Memory Map Control Image

The initial address value in bit positions 15-31 of register R is the virtual address of the first word of the memory map control image. The word length of the control image to be loaded is specified by the initial count in bit positions 0-7 of register Ru1. A word count of 64 is sufficient to load the entire block of 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 causes the first registers loaded to be overwritten.

Each word of the memory map control image is assumed to be in the following format:



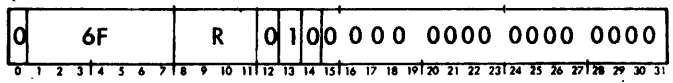
Memory Map Loading Process

Bit positions 15-22 of register Ru1 initially points to the first 512-word page of virtual addresses that is to be controlled by the map image being loaded. MMC moves the map image into the memory map control registers one word at a time, thus loading the page address for four consecutive memory map registers with each image word. As each word is loaded into the memory map, the virtual address of the image area is incremented by 1, the word count is decremented by 1, and the value in bit positions 15-22 of register Ru1 is incremented by 4; this process continues until the word count is reduced to 0. When the loading process is completed, bit positions 15-31 of register R contain a value equal to the sum of the initial map image address plus the initial word count. Also, bit positions 0-7 of register Ru1 contain all 0's, and bit positions 15-22 of register Ru1 contain a value equal to the sum of the initial contents plus 4 times the initial word count.

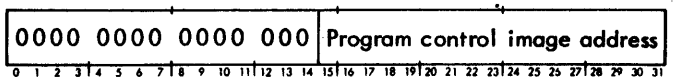
LOADING THE ACCESS PROTECTION CONTROLS

The following diagrams represent the configurations of MMC, register R, and register Ru1 that are required to load the access protection controls:

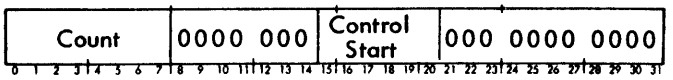
The instruction format is:



The contents of register R are:



The contents of register Ru1 are:



Access Protection Control Image

The initial address value in register R is the virtual address of the first word of the access control image, and the word length of the first control image is specified by the initial count in register Ru1. A word count of 16 is sufficient to load the entire block of 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. Each word of the access control image is assumed to be in the following format:

Access Control Loading Process

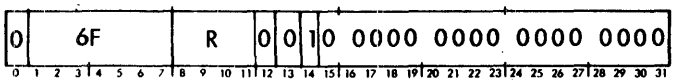
Bit positions 15-20 of register Ru1 initially point to the first 512-word page of virtual addresses that is to be controlled

by the access control image. MMC moves the access control image into the access control registers one word at a time, thus loading the controls for 16 consecutive 512-word pages with each image word. As each word is loaded, the virtual address of the control image is incremented by 1, the word count is decremented by 1, and the value in bit positions 15-20 of register Ru1 is incremented by 4; this process continues until the word count is reduced to 0. When the loading process is completed, register R contains a value equal to the sum of the initial control image address plus the initial word count. Also, the final word count is 0, and bit positions 15-20 of register Ru1 contain a value equal to the sum of the initial contents plus 4 times the initial word count.

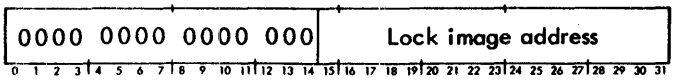
LOADING THE MEMORY WRITE PROTECTION LOCKS

The following diagrams represent the configurations of MMC, register R, and register Ru1 that are required to load the memory write protection locks:

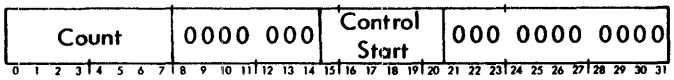
The instruction format is:



The contents of register R are:

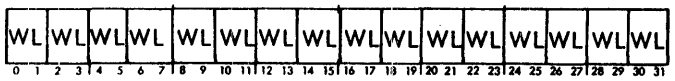


The contents of register Ru1 are:



Memory Lock Control Image

The initial address value in register R is the virtual address of the first word of the memory lock control image, and word length of the image is specified by the initial count in register Ru1. A word count of 16 is sufficient to load the entire block of memory locks. The memory lock registers are treated as a circular set, with the register for memory addresses 0 through X'1FF' immediately following the register for memory addresses X'1FE00' through X'1FFFF'; thus, a word count greater than 16 causes the first registers loaded to be overwritten. Each word of the lock image is assumed to be in the following format:



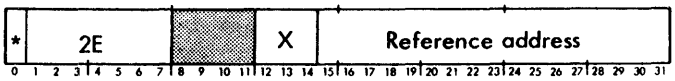
Memory Lock Loading Process

Bit positions 15-20 of register Ru1 initially point to the first 512-word page of actual core memory addresses that is to be controlled by the memory lock image. MMC moves the lock image into the lock registers 1 word at a time, thus loading the locks for 16 consecutive 512-word pages with each image word. As each word is loaded, the virtual address of the lock image is incremented by 1, the word count is decremented by 1, and the value in bit positions 15-20 of register Ru1 is incremented by 4; this process continues until the word count is reduced to 0. When the loading process is completed, register R contains a value equal to the sum of the initial lock image address plus the initial word count. Also, the final word count is 0, and bit positions 15-20 of register Ru1 contain a value equal to the sum of the initial contents plus 4 times the initial word count.

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 doubleword 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.

WAIT WAIT
(Word index alignment, privileged)

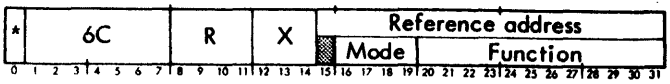


WAIT causes the CPU to cease all operations until an interrupt activation occurs, or until the computer operator manually moves the COMPUTE switch (on the processor control panel or on the free-standing console) from the RUN position to IDLE and then back to RUN. The instruction address portion of the PSD is updated before the computer begins waiting; therefore, while the CPU is waiting, the INSTRUCTION ADDRESS indicators contain the virtual address of the next location in ascending sequence after WAIT and the contents of the next location are displayed in the DISPLAY indicators (on the processor control panel and on the free-standing 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 CPU is waiting, the computer 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 COMPUTE switch is moved from RUN to IDLE and back to RUN while the CPU is waiting, instruction execution proceeds with the next instruction in sequence after the WAIT instruction.

If WAIT is indirectly addressed and the indirect reference address is nonexistent, the nonallowed operation trap (location X'40') is activated. 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 CPU is capable of directly communicating with other elements of the SIGMA 7 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, 2 condition code lines, and various control lines, that are connected to various CPU circuits and to special systems equipment.

READ DIRECT causes the CPU to present bits 16 through 31 of the effective virtual address to other elements of the SIGMA 7 system on the RD/WD address lines. Bits 16-31 of the effective virtual address identify a specific element of the SIGMA 7 system that is expected to return information (2 condition code bits plus a maximum of 32 data bits) to the CPU. The significance and number of data bits returned to the CPU 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. The condition code is 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 computer control
0	0	0	1	Unassigned
0	0	1	0	XDS testers
0	0	1	1	Assigned to various groups of standard XDS products
:	:	:	:	
1	1	1	0	
1	1	1	1	Special systems control (for customer use with specially designed equipment)

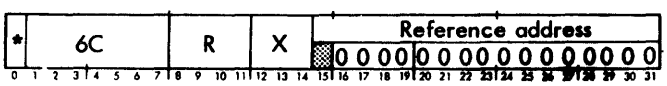
If bits 16-19 of the effective virtual address are nonzero (mode 1 through mode F), CC1 and CC2 are set to zero and CC3 and CC4 are set according to the state of the two condition code lines from the external device.

READ DIRECT
INTERNAL COMPUTER CONTROL (MODE 0)

In this mode, the condition code is unconditionally set according to the states of the four SENSE switches on the processor control panel. If a particular SENSE switch is set, the corresponding bit of the condition code is set to 1; if a SENSE switch is reset, the corresponding bit of the condition code is set to 0 (see "SENSE" in chapter 5).

Read SENSE Switches

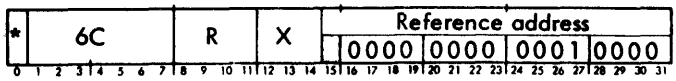
The following configuration of RD can be used to read the control panel SENSE switches:



In this case, only the condition code is affected.

Read and Reset MEMORY FAULT Indicators

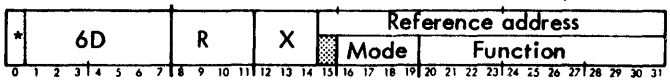
Each core memory module is associated with a MEMORY FAULT indicator that is turned on whenever a memory parity or overtemperature condition occurs. The following configuration of RD is used to record and reset the MEMORY FAULT indicators.



If the R field of RD is nonzero, bit positions 0-23 of register R are reset to all 0's, bit positions 24-31 are set according to the current states of the MEMORY FAULT indicators, and all MEMORY FAULT indicators are reset. If a bit position in register R is set to 1, a memory fault has been detected in the corresponding core memory module. If the R field of RD is 0, the MEMORY FAULT indicators and the contents of register 0 remain unchanged (although the condition code is still set to the value of the SENSE switches). The MEMORY FAULT indicators are also reset by means of the SYS RESET/CLEAR switch on the processor control panel (or on the free-standing console).

Affected: (R), CC, MEMORY FAULT Indicators

WD WRITE DIRECT
(Word index alignment, privileged)



WRITE DIRECT causes the CPU to present bits 16 through 31 of the effective virtual address to other elements of the SIGMA 7 system on the RD/WD address lines (see READ DIRECT). Bits 16-31 of the effective virtual address identify a specific element of the SIGMA 7 system that is to receive control information from the CPU. 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 condition code is set by the addressed element, regardless of the value of the R field.

Bits 16-19 of the effective virtual address determine the mode of the WD instruction, as follows:

Bit Position 16	17	18	19	Mode
0	0	0	0	Internal computer control
0	0	0	1	Interrupt control
0	0	1	0	XDS testers
0	0	1	1	Assigned to various groups of standard XDS products
:	:	:	:	
1	1	1	0	
1	1	1	1	Special systems control (for customer use with specially designed equipment)

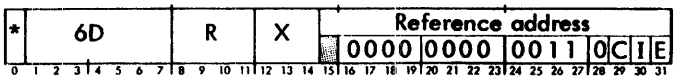
If bits 16-19 of the effective virtual address are nonzero (mode 1 through mode F), CC1 and CC2 are set to zero and CC3 and CC4 are set according to the state of the two condition code lines from the external device.

WRITE DIRECT
INTERNAL COMPUTER CONTROL (MODE 0)

In this mode, the condition code is unconditionally set according to the states of the four SENSE switches on the processor control panel. If a particular SENSE switch is set, the corresponding bit of the condition code is set to 1; if a SENSE switch is reset, the corresponding bit of the condition code is reset to 0 (see "SENSE" in Chapter 5).

Set Interrupt Inhibits

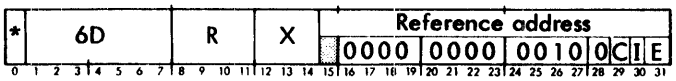
The following configuration of WD can be used to set the interrupt inhibits (bit positions 37-39 of the PSD).



A logical inclusive OR is performed between bits 29-31 of the effective virtual address and bits 37-39 of the PSD. If any (or all) of bits 29-31 of the effective virtual address are 1's, the corresponding inhibit bits in the PSD are set to 1's; the current state of an inhibit bit is not affected if the corresponding bit position of the effective virtual address contains a 0.

Reset Interrupt Inhibits

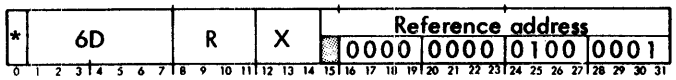
The following configuration of WD can be used to reset the interrupt inhibits:



If any (or all) of bits 29-31 of the effective virtual address are 1's the corresponding inhibit bits in the PSD 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.

Set ALARM Indicator

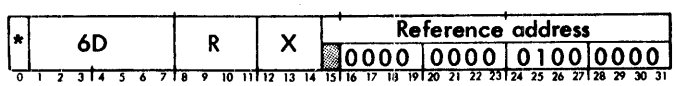
The following configuration of WD is used to set the ALARM indicator on the maintenance section of the processor control panel:



If the COMPUTE switch on the processor control panel is in the RUN position 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 computer speaker. The signal may be interrupted by moving the COMPUTE switch to the IDLE position, by moving the AUDIO switch to the OFF position, or by resetting the ALARM indicator.

Reset ALARM Indicator

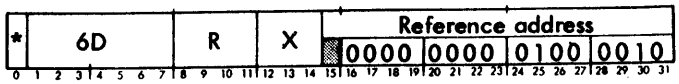
The following configuration of WD is used to reset the ALARM indicator:



The ALARM indicator is also reset by means of either the CPU RESET/CLEAR switch or the SYS RESET/CLEAR switch on the processor control panel (or on the freestanding console).

Toggle Program-Controlled-Frequency Flip-flop

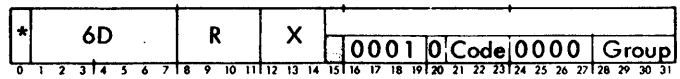
The following configuration of WD is used to "toggle" the CPU program-controlled-frequency (PCF) flip-flop:



The output of the PCF flip-flop is transmitted to the computer speaker through the AUDIO switch on the maintenance section of the processor 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 toggling (setting and resetting) the PCF flip-flop at the appropriate rate. Execution of the above configuration of WD also resets the ALARM indicator.

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 CPU interrupt system:



Bits 28 through 31 of the effective address specify the identification number (see Table 2) 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, excluding Power on/Power off, within the specified group (see Table 2). Bit position 16 of register R contains the selection bit for the highest-priority (lowest-numbered) interrupt level within the group, and bit position 31 of register R contains the selection bit for the lowest-priority (highest-numbered) interrupt level within the group. Each interrupt level in the designated group is operated on according to the function code specified by bits 21 through 23 of the effective address of WD. The codes and their associated functions are as follows:

Code	Function
000	Undefined
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.

†These codes clear the current interrupt, i.e., remove from the active or waiting state all levels selected by a 1 (see Figure 7).

Code	Function
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 the waiting state.

INPUT/OUTPUT INSTRUCTIONS

"Standard" SIGMA 7 I/O refers to the normal I/O system consisting of input/output processors, device controllers, and devices. This system handles normal communications with standard peripherals such as printers, discs, tapes, and so forth. When dealing with standard I/O operations, the CPU uses the following five instructions:

Instruction Name	Mnemonic
Start Input/Output	SIO
Halt Input/Output	HIO
Test Input/Output	TIO
Test Device	TDV
Acknowledge Input/Output Interrupt	AIO

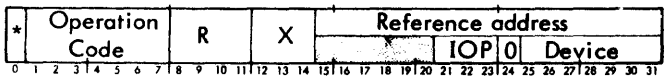
If execution of any input/output instruction is attempted while the computer is in the slave mode (i.e., while bit 8 of the current program status doubleword is a 1), the computer unconditionally aborts execution of the instruction (at the time of operation code decoding) and traps to location X'40'.

I/O ADDRESSES

The device to be operated on by an I/O instruction is selected by the effective virtual address of the I/O instruction itself. Indirect addressing and/or indexing are performed, as for other word-addressing instructions, to compute the effective virtual address of the I/O instruction. However, the effective address is not used as a memory reference (i.e., not subject to memory mapping). For the SIO, HIO, TIO, and TDV instructions, the 11 low-order bits of the effective virtual address constitute an I/O address. For the AIO instruction, the device causing the interrupt returns its 11-bit I/O address as part of the response to the AIO instruction.

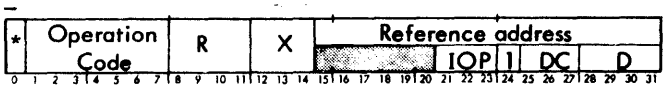
An I/O address occupies bit positions 21 through 31 of the effective virtual address, with bits 21, 22, and 23 of the I/O address specifying one of eight possible IOPs that can be controlled by a CPU. The remainder of the I/O address is factored into one of two forms, depending on bit 24, as follows:

Case I: Single-unit device controllers (bit 24 is 0)



Bits 25 through 31 of the I/O address (DC/Device) constitute a single code specifying a particular combination of device controller and device. Normally these codes refer to device controllers that drive only a single device, such as card readers, card punches, line printers, etc.

Case II: Multiunit device controllers (bit 24 is 1)



Bit positions 25 through 31 of the I/O address contain a 3-bit device controller code (DC) in bit positions 25-27 and a 4-bit device code (Device) in bit positions 28-31. This form of I/O address is used for device controllers (such as magnetic tape and rapid access data file controllers) that control information exchange with only one device at a time (out of a set of as many as 16 devices).

I/O UNIT ADDRESS ASSIGNMENT

Device controller numbers are normally assigned to a multiplexor IOP in numerical sequence, beginning with zero and continuing through the highest number recognized by the IOP (i.e., X'7', X'F', X'17', or X'1F'). In the case of multiunit device controllers, the device controller number must be in the range X'0' through X'7' because the I/O address field structure allows for a 3-bit multiunit device controller number. In the case of single-unit device controllers, any of the available numbers in the range X'0' through X'1F' may be assigned to the device controller, providing that the same number has not already been assigned to a multiunit device controller. For example, if device controller number X'0' is assigned to a magnetic tape unit controller, the number X'0' cannot also be used for a card reader (although the coding of the I/O address field would be different in bit position 24). The I/O address codes used by standard XDS software are

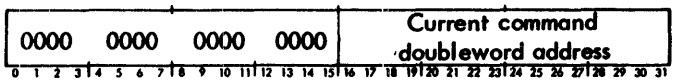
I/O address	Peripheral device designation
X'080'	IOP 0, device controller 0, magnetic tape unit 0
X'081'	IOP 0, device controller 0, magnetic tape unit 1
⋮	⋮
X'087'	IOP 0, device controller 0, magnetic tape unit 7
X'001'	IOP 0, device controller 1, keyboard/printer
X'002'	IOP 0, device controller 2, line printer
X'003'	IOP 0, device controller 3, card reader
X'004'	IOP 0, device controller 4, card punch
X'005'	IOP 0, device controller 5, paper tape reader/punch

I/O STATUS RESPONSE

All I/O instructions result in the setting of condition code CC1 and CC2 to denote the nature of the I/O response. The R field of the I/O instruction specifies one of the general registers that is to accept additional I/O response information during the execution of an I/O instruction. In some situations, the programmer may want two sets of response information loaded into the general registers, while in other situations he may want only one set, or even no information loaded into a general register. This control is achieved by coding the R field of the I/O instruction. One set of response information is loaded into register R and another set may be loaded into register Ru1. If the R field is an even, nonzero number, registers R and R + 1 are each loaded with response information. If the R field specifies

an odd-numbered general register, then only register R is loaded with response information. However, if the R field is 0, R and Ru1 are not loaded with response information. Also, if R ≠ 0 and CC1 is set to 1 as a result of the operation, no status information is returned to R and Ru1. The I/O response information loaded into the general register for SIO, HIO, TIO, and TDV instructions is in the following format:

Word into register R



Word into register Ru1

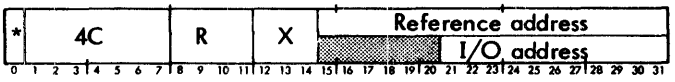
Current Command Doubleword Address. After the addressed device has received an order, this field contains the 16 high-order bits of the core memory address for the command doubleword (see page 88) currently being processed for the addressed device.

Status. The meaning of this field depends on the particular I/O instruction being executed and upon the selected I/O device (see Table 8).

Byte Count. After the addressed device has received an order, this field contains a count of the number of bytes yet to be transmitted to or from memory by the operation called for by the order.

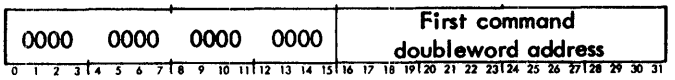
The format of I/O response information loaded into register R for the instruction AIO is described on page 86.

SIO START INPUT/OUTPUT
(Word index alignment, privileged)



START INPUT/OUTPUT is used to initiate an input or output operation with the device selected by the I/O address (bits 21-31 of the effective virtual address of the instruction).

SIO utilizes data in general register 0, which is assumed to have the following content when SIO is executed.



General register 0 is temporarily dedicated during the execution of an SIO instruction to specify the starting doubleword address for the IOP command list. The doubleword address in register 0 is the 16 high-order bits of a memory address; thus, the address in register 0 always specifies an even-numbered word location. (The IOP command list is described in "IOP Command Doublewords", Chapter 4.)

If I/O address recognition exists in the I/O system, and the device controller and device are in the "ready" condition and no interrupt condition is pending, the SIO is accepted and the device is started (i.e., advanced to the "busy" condition). If the SIO is accepted, the first command doubleword address is loaded into the IOP command address counter associated with the device controller specified by the I/O address of the SIO instruction. Then, if the device is in the "automatic" mode, it requests an order from the IOP. The IOP loads the first command doubleword of the I/O command list into its appropriate registers and transmits the order to the device.

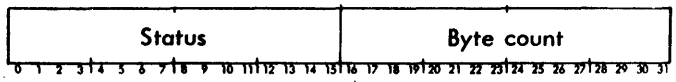
The CPU condition code provides an indication of whether the I/O address specified by the SIO instruction was or was not recognized by the I/O system and whether the SIO instruction was or was not accepted by the device (i.e., whether the device did or did not advance to the "busy" condition).

The condition code settings for SIO are:

1	2	3	4	Result
0	0	-	-	I/O address recognized and SIO accepted
0	1	-	-	I/O address recognized but SIO not accepted
1	0	-	-	IOP address recognized but device controller either is attached to a "busy" selector IOP that cannot return status at this time or, for specific device controllers, is currently "busy" with another device. No status information is returned to general registers.
1	1	-	-	I/O address not recognized and SIO not accepted; no status information is returned to general registers.

STATUS INFORMATION FOR SIO

In the event that the SIO instruction was not accepted (i.e., CC1 = 0 and CC2 = 1), the status information returned as a part of the I/O response provides indications of why the SIO instruction was not accepted. If the SIO instruction has been coded with an R field value of 0, or if CC1 (as a result of the execution of this instruction) is a 1, only the condition code settings are available. If the R field value is odd, register R contains the following information:



Bit Position	Function
0	Device interrupt pending: if this bit is 1, the addressed device has requested an interrupt and the interrupt has not been acknowledged by an AIO instruction. Device interrupts can be achieved by coding of the flag portion of the I/O command doubleword. Device interrupts can also be achieved by using M modifiers in the basic order to the device (M bits in the Order portion of the command doubleword). In either case, the device will not accept a new SIO instruction until the interrupt-pending condition is cleared (i.e., the condition code settings for the SIO instruction will indicate "SIO not accepted" if the interrupt-pending condition is present in the addressed device.

Table 8. Status 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	6	7	8	9	10	11	12	13	14	15		
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	interrupt pending	unique to the device and the device controller
-	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	device ready	
-	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	device not operational	
-	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	device unavailable	
-	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	device busy	
-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	device manual	
-	-	-	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	
-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	unassigned	
-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	incorrect length	same as for SIO, HIO, and TIO
-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	transmission data error	
-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	transmission memory error	
-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	memory address error	
-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	IOP memory error	
-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	IOP control error	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	IOP halt	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	Selector IOP busy	

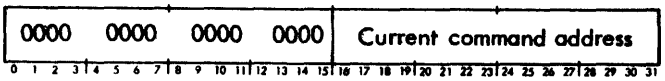
Position and State in Register R

Device Status Byte								Operational Status Byte								Significance for AIO
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
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
-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	unassigned
-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	

Bit Position	Function
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 an IOP 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 operation), 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.
4	Device unusual end occurred during last operation: if this bit is 1, the reason for the indication may be a normal end (such as an end of file) or a fault condition. For a fault condition, the device has halted at other than its normal stopping point. In either case, the device will not automatically request further action from its device controller. The specific details of this indication are a function of the particular device.
5, 6	Device controller condition: 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. In either case, 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.
7	Unassigned
8	Incorrect length: if this bit is 1, an incorrect length condition has been detected during the previous operation. Incorrect length is caused by a channel end (or end of record) condition occurring before the device controller has received a "count done" signal from the IOP, 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. Normally, a count done signal is sent to the device controller by the IOP to indicate that the byte count associated with the current operation has been reduced to zero. The IOP is capable of suppressing an error condition on incorrect length, since there are many situations in which incorrect length is a legitimate situation and not a true error condition. Incorrect length is suppressed as an error by coding the SIL flag (a 1 in bit 38) of the IOP command doubleword (see page 90). At the end of the execution of an I/O command list, this status bit is 1 if an incorrect length condition occurred anywhere in the command list, regardless of the coding of the SIL flag.
9	Transmission data error: this bit is set to 1 if the IOP or device controller has detected a parity error or data overrun in the transmitted information. At the end of an execution of an I/O command list, this status bit is 1 if a transmission data error occurred anywhere in the command list.
10	Transmission memory error: this bit is set to 1 if a memory parity error has occurred during a data input/output operation. A parity error is detected on any output operation and on partial-word input operations. At the end of an execution of an I/O command list, this status bit is 1 if a transmission memory error occurred anywhere in the command list. A device halt does not occur unless the HTE flag in the IOP command doubleword is set to 1 (see page 90).
11	Memory address error: a nonexistent memory address has been encountered on either data or commands. Core memory locations 0 through 15

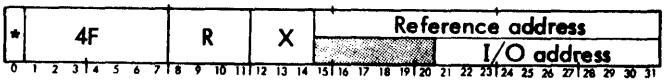
Bit Position	Function
11 (cont.)	are not considered nonexistent because the IOP can work with these addresses as normal memory addresses.
12	IOP memory error: if a memory parity error has occurred while the IOP was fetching a command, this bit is set to 1.
13	IOP control error: this bit is set to 1 if the IOP has encountered two successive TRANSFER IN CHANNEL commands.
14	IOP halt: this bit is set to 1 if the IOP has issued a halt order to the addressed I/O device because of an error condition.
15	Selector IOP busy: this bit is set to 1 if a selector IOP is addressed by the I/O instruction and the selector IOP is currently in use by some I/O device. The selector IOP is considered to be in use from the time that a device accepts an SIO instruction until the operation is completed.
16-31	Byte count: a count of the number of bytes yet to be transmitted to or from memory in the operation called for by the current command doubleword.

If the R field value of the SIO instruction is even and not 0, the condition code and register R+1 contain the information described above and register R contains the following information:



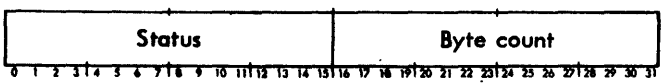
Bit Position	Function
16-31	Current command doubleword address: the 16 high-order bits of the core memory address from which the command doubleword for the I/O operation currently being processed by the addressed device controller was fetched.

HIO HALT INPUT/OUTPUT
(Word index alignment, privileged)

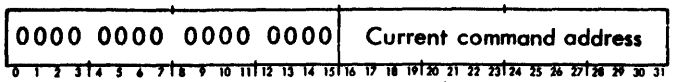


HALT INPUT/OUTPUT causes the addressed device to immediately halt its current operation (perhaps improperly, in the case of magnetic tape units, when the device is forced to stop at other than interrecord gap). If the device is in an interrupt-pending condition, the condition is cleared.

If the R field of the HIO instruction is 0 or if no I/O address recognition exists, no general registers are affected, but the condition code is set. If the R field is an odd value, the condition code is set and the following information is loaded into register R.



The status information returned for HIO has the same interpretation as that returned for the instruction SIO (see page 81), and shows the I/O status at the time of the halt. The count information shows the number of bytes remaining to be transmitted at the time of the halt. If the R field of HIO is an even value and not 0, the condition code is set, register R+1 is loaded as shown above, and register R contains the following information:



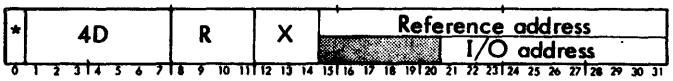
The current command doubleword address has the same interpretation as that for the instruction SIO.

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

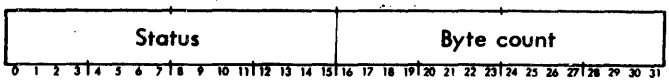
Condition code settings:

1	2	3	4	Result of HIO
0	0	-	-	I/O address recognized and device controller is not "busy".
0	1	-	-	I/O address recognized but device controller was "busy" at the time of the halt.
1	1	-	-	I/O address not recognized.

TIO TEST INPUT/OUTPUT
(Word index alignment, privileged)



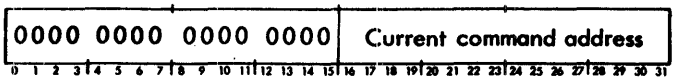
TEST INPUT/OUTPUT is used to make an inquiry on the status of data transmission. The operation of the selected IOP, device controller, and device are not affected, and no operations are initiated or terminated by this instruction. The responses to TIO provide the program with the information necessary to determine the current status of the device, device controller, and IOP, the number of bytes remaining to be transmitted to or from 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, or if CC1 (as a result of the execution of this instruction) is a 1, 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:



The status information has the same interpretation as the status information returned for the instruction SIO (see page 81), and shows the I/O status at the time of sampling.

The count information shows the number of bytes remaining to be transmitted at the time of sampling. If the R field of the TIO instruction is an even value and not 0, the

condition code is set, register R + 1 is loaded as shown above, and register R is loaded as follows:



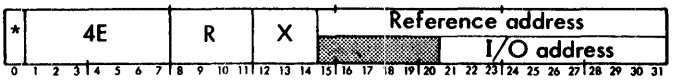
The current command doubleword address has the same interpretation as for the instruction SIO.

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

Condition code settings:

1	2	3	4	Result of TIO
0	0	-	-	I/O address recognized and acceptable SIO is currently possible.
0	1	-	-	I/O address recognized but acceptable SIO is not currently possible.
1	0	-	-	IOP address recognized but device controller either is attached to a "busy" selector IOP that cannot return status at this time or, for specific device controllers, is currently "busy" with another device. No status information is returned to general registers.
1	1	-	-	I/O address not recognized; no status information is returned to general registers.

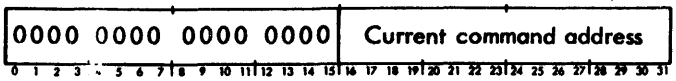
TDV TEST DEVICE
(Word index alignment, privileged)



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 are 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 to or from memory 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, or if CC1 (as a result of the execution of this instruction) is a 1, 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:

Bit Position	Function
0-7	Unique to the device and device controller.
8-15	Same as for bits 8-15 of the status information for instruction SIO.

The count information shows the number of bytes remaining to be transmitted in the current operation at the time of the TDV instruction. If the value of the R field of TDV is an even value and not 0, the condition code is set, register R + 1 is loaded as shown above, and register R is loaded as follows:



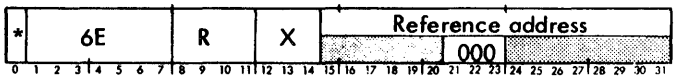
The current command doubleword address has the same interpretation as for the instruction SIO.

Affected: (R), (Ru1), CC1

Condition code settings:

1	2	3	4	Result of TDV
0	0	-	-	I/O address recognized.
0	1	-	-	I/O address recognized and device-dependent condition is present.
1	0	-	-	IOP address recognized but device controller either is attached to a "busy" selector IOP that cannot return status at this time or, for specific device controllers, is currently "busy" with another device. No status information is returned to general registers.
1	1	-	-	I/O address not recognized; no status information is returned to general registers.

AIO ACKNOWLEDGE INPUT/OUTPUT INTERRUPT
(Word index alignment, privileged)



AIO is used to acknowledge an input/output interrupt and to identify what I/O unit is causing the interrupt and why. Bits 21, 22, and 23 of the effective virtual address of the AIO instruction (the IOP portion of the I/O selection code field) specify the type of interrupt being acknowledged. These bits should be coded 000 to specify the standard I/O system interrupt acknowledgement (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) has no other use in the standard I/O interrupt acknowledgement because the identification of the interrupt source is one of the responses of the standard I/O system to the AIO instruction.

Standard I/O system interrupts can be initiated for the following conditions:

Condition	Interrupt prerequisite†	Status bit set
Zero byte count	IZC = 1	10
Channel end	ICE = 1	11

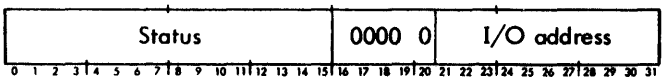
†IZC, ICE, IUE, HTE, and SIL refer to flag bits in the IOP command doublewords (see Chapter 4).

Condition	Interrupt prerequisite[†]	Status bit set
Transmission memory error	IUE = 1, HTE = 1	12
Incorrect length	IUE = 1, HTE = 1 and SIL = 0	8, 12
Memory address error (IOP memory error or IOP control error)	IUE = 1	12
Transmission data error	IUE = 1, HTE = 1	9, 12

When a device interrupt condition occurs, the IOP forwards the request to the CPU interrupt system I/O interrupt level. If this interrupt level is armed, enabled, and not inhibited (see page 20, "Control of the Interrupt System"), the CPU eventually acknowledges the interrupt request and executes the XPSD instruction in core memory location X'5C', which 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 or if no device interrupt request is present, 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:



Bit Position	Function
0-7	Unique to the device and the device controller.
8	Incorrect length: if this bit is 1, an incorrect length condition has been signaled to the IOP by the device controller during the previous operation.
8 (cont.)	Incorrect length is suppressed as an error by coding the SIL flag (a 1 in bit 38) of the command doubleword. At the end of the execution of an I/O command list, this status bit is 1 if an incorrect length condition occurred anywhere in the command list, regardless of the coding of the SIL flag.
9	Transmission data error: this bit is set to 1 if the IOP or device controller has detected a parity error or data overrun in the transmitted information.
10	Zero byte count interrupt: if this bit is 1, the byte count for the operation being performed by the interrupting device has been reduced to 0, and the interrupt at zero byte count (IZC) flag in the command doubleword for the operation was coded with a 1.
11	Channel end interrupt: if this bit is 1, the device controller has signaled channel end to the IOP, and the interrupt at channel end (ICE) flag in the command doubleword for the operation was coded with a 1.
12	IOP unusual end interrupt: if this bit is 1, the IOP has originated the interrupt as a result of a fault or unusual condition reported by the device.
13-20	Unassigned
21-31	I/O address: this field identifies the highest-priority device requesting an interrupt. Bit positions 21-23 identify the IOP. If bit 24 is 0, bits 25-31 constitute a common device controller and device code; if bit 24 is 1, bits 25-27 constitute a device controller code and bits 28-31 identify a device attached to that device controller.

The AIO instruction resets the interrupt request signal from the highest priority I/O device requesting interrupt service (i.e., the device identified above in bits 21-31).

Affected: (R), CC1, CC2

Condition code settings:

1	2	3	4	Result of AIO
0	0	-	-	normal interrupt recognition.
0	1	-	-	unusual interrupt recognition.
1	1	-	-	no interrupt recognition.

[†]IZC, ICE, IUE, HTE, and SIL refer to flag bits in the IOP command doublewords (see Chapter 4).

4. INPUT/OUTPUT OPERATIONS

In a SIGMA 7 system, input/output operations are primarily under control of one or more input/output processors (IOPs). This allows the CPU to concentrate on program execution, free from the time-consuming details of I/O operations. Any I/O events that require CPU intervention are brought to its attention by means of the interrupt system.

In the following discussion, the terminology conventions used are that the CPU executes instructions, the IOP executes commands, and the device controllers and/or I/O devices execute orders. To illustrate, the CPU will execute the START INPUT/OUTPUT (SIO) instruction to initiate an I/O operation. During the course of an I/O operation, the IOP might issue a command called Control, to transmit a byte to a device controller or I/O device that interprets the byte as an order, such as Rewind.

SIGMA 7 IOPs operate independently after they have been started by the central processor. They automatically pick up a chain of one or more commands from core memory and then execute these commands until the chain is completed.

The multiplexor IOP can simultaneously operate up to 32 device controllers. Each device controller is assigned its own channel and chain of I/O commands. The selector IOP can handle any of up to 32 high-speed device controllers at rates up to the full speed of the core memory (one 32-bit word/cycle). A pair of selector IOPs can share a common memory bus if desired.

The flexible SIGMA 7 I/O structure permits both command chaining (making possible multiple-record operations) and data chaining (making possible scatter-read and gather-write operations) without intervening CPU control. Command chaining refers to the execution of a sequence of I/O commands, under control of an IOP, on more than one physical record. Thus, a new command must be issued for each physical record even if the operation to be performed for a record is the same as that performed for the previous record. Data chaining refers to the execution of a sequence of I/O commands, under control of an IOP, that gather (or scatter) information within one physical record from (or to) more than one region of memory. Thus, a new command must be issued for each portion of a physical record when the data associated with that physical record appears (or is to appear) in noncontiguous locations in memory. For example, if information in specific columns of two cards in a file are to be stored in specific regions of memory, the I/O command list might appear as follows:

1. Read card, store columns 1-10, data chain
2. Store columns 11-60, data chain
3. Store columns 61-80, command chain (end of data chain)
4. Read card, store columns 1-40, data chain
5. Store columns 41-80 (end of command chain, end of data chain)

The SIGMA 7 CPU itself plays a minor role in the execution of an I/O operation. The CPU-executed program is responsible for creating and storing the command list (prepared prior to the initiation of any I/O operation) and for supplying the IOP with a pointer to the first command in the I/O command list. Most of the communication between the CPU and the I/O system is carried out through memory.

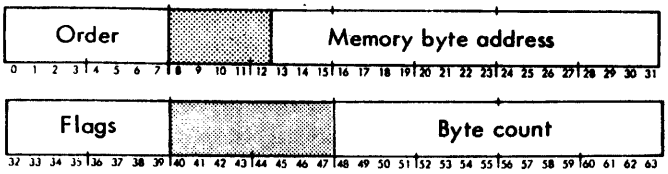
The following is an example of the sequence of events that occurs during an I/O operation:

1. A CPU-executed program writes a sequence of I/O commands in core memory.
2. The CPU executes the instruction START INPUT/OUTPUT and furnishes the IOP with an 11-bit I/O address (designating the device to be started) and a 16-bit first command address (designating the actual core memory doubleword location where the first command for this device is located). At this point, either the device is started (if in the "ready" condition with no device interrupt pending) or an instruction reject occurs. The CPU is informed by condition code settings as to which of the two alternatives has occurred. If the START I/O instruction is accepted, the command counter portion of the IOP register associated with the designated device controller is loaded with the first command address. Assuming that the SIO instruction is accepted, from this time until the full sequence of I/O commands has been executed, the main program of the CPU need play no role in the I/O operation. At any time, however, it may obtain status information on the progress of the I/O operation without interfering with the operation.
3. The device is now in the "busy" condition. When the device determines that it has the highest priority for access to the IOP, it requests service from the IOP with a service call. The IOP obtains the address of the first command doubleword of the I/O sequence (from the command counter asssociated with this device). The IOP then fetches the I/O command doubleword from core memory, loads the doubleword into another register associated with the device, and transmits the first order (extracted from the command doubleword) to the device.
4. Each command counter contains the memory address of the current I/O command in the sequence for its device. When the device requires further servicing, it makes a request to the IOP, which then repeats a process similar to that of step 3.
5. If a data transmission order has been sent to a device, control of the transmission resides in the device. As each character is obtained by the I/O device, the IOP is signaled that data is available. The IOP uses the information stored in its own registers to control the information interchange between the I/O device and the memory, on either a word-by-word or character-by-character basis, depending on the nature of the device.

6. When all information exchanges called for by a single I/O command doubleword have been completed, the IOP uses the command counter to obtain the next command doubleword for execution. This process continues until all such command doublewords associated with the I/O sequence have been executed.

IOP COMMAND DOUBLEWORDS

All IOP command doublewords (except Transfer in Channel and Stop) are assumed to be in the following format:



ORDER

Bit positions 0 through 7 of the command doubleword contain the I/O order for the device controller or device. The I/O orders are shown below. Bits represented by the letter "M" specify orders or special conditions to the device and are unique for each type of device.

Bit positions 0 1 2 3 4 5 6 7	Order
M M M M M M 0 1	Write
M M M M M M 1 0	Read
M M M M M M 1 1	Control
M M M M 0 1 0 0	Sense
M M M M 1 1 0 0	Read Backward

Write. The Write order causes the device controller to initiate an output operation. Bytes are read in an ascending sequence from the memory location specified by the memory byte address field of the command doubleword. The output operation continues until the device signals "channel end", or until the byte count is reduced to 0 and no further data chaining is specified. Channel end occurs when the device has received all information associated with the output operation, has completed all checks, and no longer requires the use of IOP facilities for the operation. Data chaining is described on the following page.

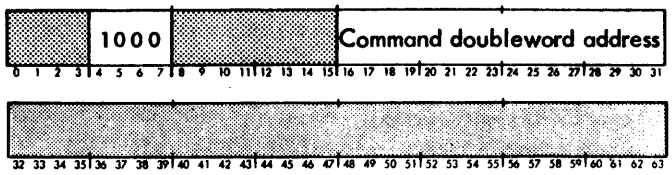
Read. The Read order causes the device controller to initiate an input operation. Bytes are stored in core memory in an ascending sequence, beginning at the location specified by the memory byte address field of the command doubleword. The input operation continues until the device signals channel end, or until the byte count is reduced to 0 and no further data chaining is specified. Channel end occurs when the device has transmitted all information associated with the input operation and no longer requires the use of IOP facilities for the operation.

Control. The Control order is used to initiate special operations by the device. For magnetic tape, it is used to issue orders such as rewind, backspace record, backspace file etc. Most orders can be specified by the M bits of the Control order; however, if additional information is required for a particular operation (e.g., the starting address of a disc-seek), the memory byte address field of the command doubleword specifies the starting address of the bytes that are to be transmitted to the device controller for the additional information. When all bytes necessary for the operation have been transmitted, the device controller signals channel end.

Sense. The Sense order causes the device to transmit one or more bytes of information, describing its current state. The bytes are stored in core memory in an ascending sequence, beginning with the address specified by the memory byte address field of the command doubleword. The number of bytes transmitted is a function of the device and the condition it describes. The Sense order can be used to obtain the current sector address from a disc or drum unit.

Read Backward. The Read Backward order (for devices that can execute it) causes the device to be started in reverse, and bytes to be transmitted to the IOP for storage into core memory in a descending sequence, beginning at the location specified by the memory byte address field of the command doubleword. In all other respects, Read Backward is identical to Read, including reducing the byte count with each byte transmitted.

The Transfer in Channel command doubleword is assumed to be in the following format:



Transfer in Channel. The Transfer in Channel command is executed within the IOP, and it has no direct effect on any of the I/O system elements external to the addressed IOP. The primary purpose of Transfer in Channel is to permit branching within the command list so that the IOP can, for example, repeatedly transmit the same set of information a number of times. When the IOP executes Transfer in Channel, it loads the command counter for the device controller it is currently servicing with the command doubleword address field of the Transfer in Channel command, loads the new command doubleword specified by this address into the IOP registers associated with the device controller, and then executes the new command. (Bit positions 0-3, and 32-63 of the command doubleword for Transfer in Channel are ignored.) Transfer in Channel thus allows a command list to be broken into noncontiguous groups of commands. When used in conjunction with command chaining, Transfer in Channel facilitates the control of devices such as unbuffered card punches or unbuffered line printers. The current flags (see "Flags" below) are not altered during this command; thus, the type of chaining called for in the previous command doubleword is retained until changed by a command doubleword following Transfer in Channel.

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, when twelve rows have been punched, it causes the IOP to skip the command it would be executing next. Thus, a command list for punching two cards might look like the following example.

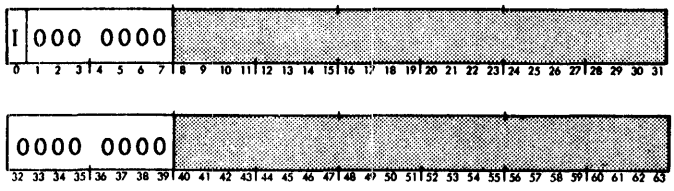
Location	Command
:	:
A	Punch row for card 1, command chain
	Transfer in Channel to A
B	Punch row for card 2, command chain
	Transfer in Channel to B
	Stop
:	:

The Transfer in Channel command also can be used in conjunction with data chaining. As one example, consider a situation often encountered in data acquisition applications, where data is transmitted in extremely long, continuous 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 look like the following example.

Location	Command
:	:
A	Read data, store into buffer 1, data chain
	Store into buffer 2, data chain
	Transfer in Channel to A
:	:

If the IOP encounters two successive Transfer in Channel commands, this is considered an IOP control error, resulting in the IOP setting the IOP control error status bit and issuing an "IOP halt" signal to the device controller. The IOP then halts further servicing of this command list.

The Stop command doubleword is assumed to be in the following format:



Stop. The Stop command causes certain devices to stop, generate a channel end condition, and also request an interrupt at location X'5C' if bit 0 in the Stop command is a 1. An AIO instruction executed after the interrupt is acknowledged results in a 1 in bit position 7 of register R, to indicate the reason for the interrupt. (Bit positions 32-39 of the command doubleword for Stop must be zero; bit positions 8-31 and 40-63 are ignored). The Stop command is primarily used to terminate a command chain for an unbuffered device, as illustrated in the example given for Transfer in Channel.

MEMORY BYTE ADDRESS

For all I/O commands (except Transfer in Channel and Stop), bit positions 13-31 of the command doubleword provide for a 19-bit core memory byte address, designating the memory location for the next byte of data. For the Write, Read, and Control orders, this field (as stored in an IOP register) is incremented by 1 as each byte is transmitted to the I/O operation; for the Read Backward order, the field is decremented by 1 as each byte is transmitted.

FLAGS

For all I/O commands (except Transfer in Channel and Stop) bit positions 32-39 of the command doubleword provide the IOP with eight flags that specify how to handle chaining, error, and interrupt situations. The functions of these flags are:

Bit Position	Function
32 (DC)	Data chain. If this flag is 1, data chaining is called for when the current byte count is reduced to 0. The next command doubleword is fetched and loaded into the IOP register associated with the device controller, but the new order code is not passed out to the device controller; thus, the operation called for by the previous order is continued. (Except for Transfer in Channel, the new command doubleword is used only to supply a new memory address, a new count, and new flags.) If the data chain flag is 0, no further data chaining is called for. Channel end is initiated either by the device running out of information, or by the byte count being reduced to 0. At channel end, the device may accept a new SIO instruction, providing that a device interrupt is not pending as a result of coding the IZC (bit 33), ICE (bit 35), or IUE (bit 37) flags, and no fault condition exists.
33 (IZC)	Interrupt at zero byte count. If this flag is 1, the IOP requests an interrupt at location X'5C' when the byte count of this command doubleword (as stored in the IOP register) is reduced to 0. An AIO instruction executed after the interrupt is acknowledged results in a 1 in bit position 10 of register R, to indicate the reason for the interrupt.
34 (CC)	Command chain. If this flag is 1, command chaining is called for when channel end occurs. The next command doubleword is fetched and loaded into the IOP register associated with the device controller, and the new order code is passed out to the device controller. If the CC flag is 0, no further command chaining is called

Bit Position	Function
	for. If both data chaining and command chaining are called for in the same command doubleword, data chaining occurs if the byte count is reduced to 0 before channel end, and command chaining occurs if the channel end occurs before the byte count is reduced to 0.
35 (ICE)	Interrupt at channel end. If this flag is 1, the IOP requests an interrupt at location X'5C' when channel end occurs for the operation being controlled by this command doubleword. An AIO instruction executed after the interrupt is acknowledged results in a 1 in bit position 11 of the status information, to indicate the reason for the interrupt. If the ICE flag is 0, no interrupt is requested.
36 (HTE)	Halt on transmission error. If this flag is 1, any error condition (transmission data error, transmission memory error, incorrect length error) detected in the device controller or IOP results in halting the I/O operation being controlled by this command doubleword. If the HTE flag is 0, an error condition does not cause the I/O operation to halt, although the error conditions are recorded in the IOP register and returned as part of the status information for the instructions SIO, HIO, and TIO. The HTE flag must be coded identically in every command doubleword associated with the same physical record. This means that when data chaining occurs, the HTE flag in the new IOP command doubleword must be the same as the HTE flag in the previous IOP command doubleword. This restriction applies to data chaining only, and not to command chaining.
37 (IUE)	Interrupt on unusual end. If this flag is 1, the device controller requests an interrupt at location X'5C' when a fault condition or unusual termination is encountered. A fault is a condition requiring the device to halt, irrespective of the coding of the HTE flag. Examples of faults are torn magnetic tape and jammed cards. When unusual termination is signaled to the IOP, further servicing of the commands for that device is suspended. An AIO instruction executed after the interrupt is acknowledged results in a 1 in bit position 12 of register R, to indicate the reason for the interrupt. If the IUE flag is 0, no interrupt is requested.
38 (SIL)	Suppress incorrect length. If this flag is 1, an incorrect length indication is not to be classified as an error by the IOP, although the IOP retains the incorrect length indication and provides an indicator (bit 8 of the status response for SIO, HIO, and TIO) to the program. If the SIL flag is 0, an incorrect length is considered an error and the IOP performs as specified by the HTE and IUE flags. Incorrect length is caused by a channel end condition occurring before the device controller has received a count-done signal from the IOP, or is caused by the device controller receiving a count-done signal before end of record; e.g., count-done before 80 columns have been read from a card. Normally, a count-done signal is sent to the device controller by the IOP to indicate that all data transfer associated with the current operation has been completed. The IOP is capable of suppressing an error condition on incorrect length, since there are many situations in which incorrect length is a legitimate condition and not a true error. The SIL flag must be coded identically in every command doubleword associated with the same physical record. This means that when data chaining occurs, the SIL flag in the new IOP command doubleword must be the same as the SIL flag in the previous IOP command doubleword. This restriction applies to data chaining only, and not to command chaining.
39 (S)	Skip. If this flag is 1, the input operation (Read or Read Backward) controlled by this command doubleword continues normally, except that no information is stored in memory. When used in conjunction with data chaining, the skip operation provides the capability for selective reading of portions of a record. If the S flag is 1 for an output (Write) operation, the IOP does not access memory, but transmits zeros as data instead (i.e., the IOP transmits the number of X'00' bytes specified in the byte count of the command doubleword). This allows a program to punch a blank card (by using the S bit and a Punch Binary order with a byte count of 120) without requiring memory access for data. If the S flag is 0, the I/O operation proceeds normally.

BYTE COUNT

For all commands (except Transfer in Channel and Stop) bit positions 48-63 of the command doubleword provide for a 16-bit count of the number of bytes to be transmitted in the I/O operation; thus, 1 to 65,536 bytes (16,384 words) can be specified for transfer before command chaining or data chaining is required. This field (as stored in an IOP register) is decremented for each byte transmitted in the I/O operation; thus, it always contains a count of the number of bytes to be transmitted to and from memory, and this count is returned as part of the response information for the instructions, SIO, HIO, TIO, and TDV. An initial byte count of 0 is interpreted as 65,536 bytes.

There are two operator control centers for a SIGMA 7 computer. The standard SIGMA 7 has a processor control panel mounted on one of the central processor cabinets. A second, optional control center is available with the free-standing console.

PROCESSOR CONTROL PANEL

The processor control panel (see Figure 8) has two distinct functional sections. The upper section (labeled MAINTENANCE SECTION) is reserved for maintenance controls and indicators, and the lower section contains the controls and indicators for the computer operator. In addition to the SENSE switches, all controls and indicators appearing in the lower section of the processor control panel are functionally duplicated in the free-standing console.

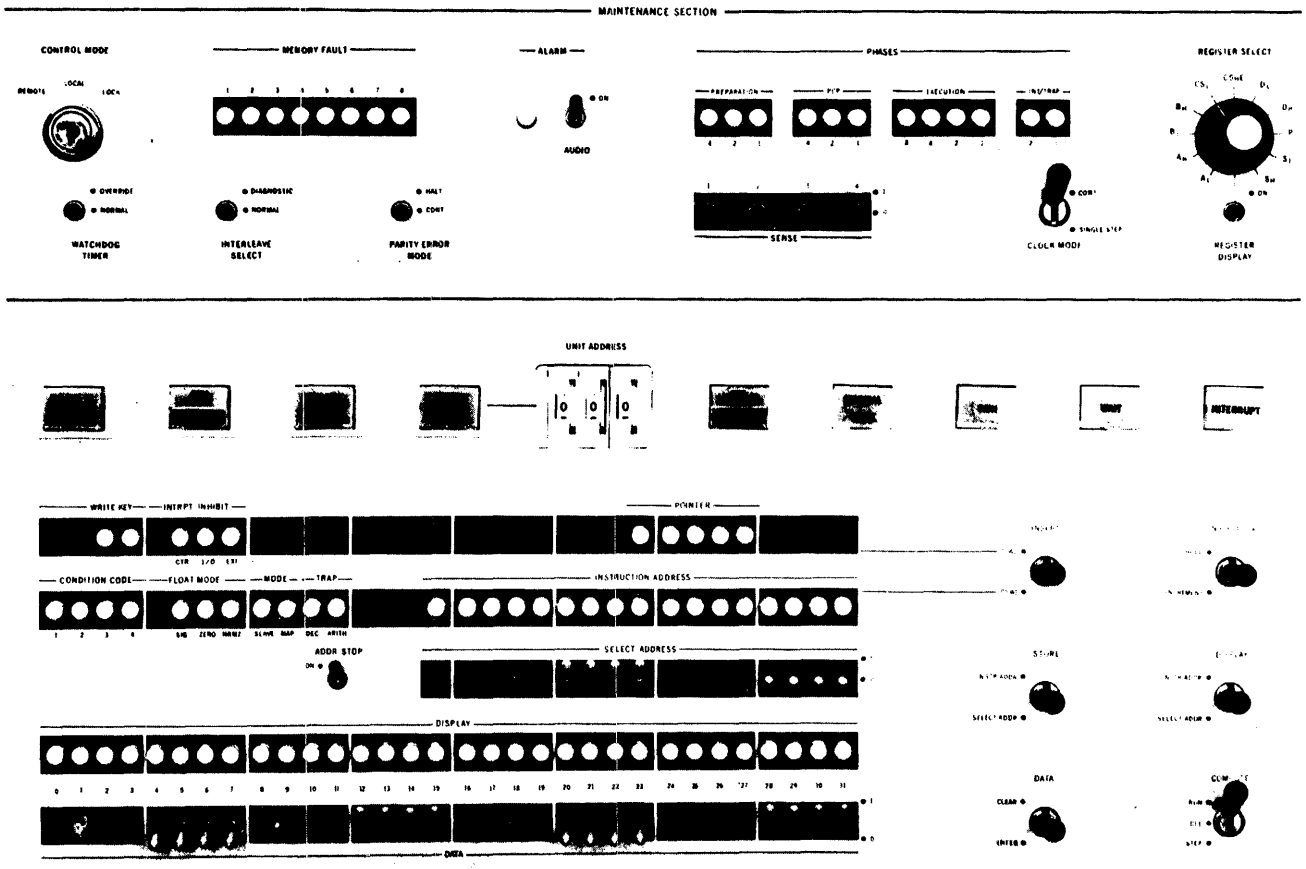
POWER

The POWER switch controls all AC power to the central processor and to all units under its direct control. The POWER switch is unlighted when the AC power is off, and is lighted when AC power is on. The POWER switch is always operative, both on the processor control panel and on the free-standing console.

CPU RESET/CLEAR

The CPU RESET/CLEAR switch is used to initialize the central processor. When this switch is pressed, the following operations are performed:

1. All interrupt levels are reset to the disarmed and disabled state.
2. The ALARM, WRITE KEY, INTRPT INHIBIT, POINTER, CONDITION CODE, FLOAT MODE, MODE, and TRAP indicators are all reset to 0's (turned off).
3. The INSTRUCTION ADDRESS indicators are set to X'25'.
4. The DISPLAY indicators are set to X'02000000', which is a LOAD CONDITIONS AND FLOATING CONTROLS IMMEDIATE (LCFI) with an R field of 0 to produce a "no operation" instruction.



The CPU RESET/CLEAR switch does not affect any operations that may be in process in the standard input/output system.

The CPU RESET/CLEAR switch is also used in conjucntion with the SYS RESET/CLEAR switch to clear core memory (i.e., reset memory to all 0's). The two switches are interlocked so that both must be pressed simultaneously for the memory clear operation to occur. The memory clear operation does not affect any general register – core memory locations 0 through 15 are cleared instead. Also the clear operation does not affect the memory control storage (write locks). Note that pressing the SYS RESET/CLEAR switch affects the I/O system and the MEMORY FAULT indicators.

I/O RESET

The I/O RESET switch is used to initialize the standard input/output system. When the switch is pressed, all peripheral devices under control of the central processor are reset to the "ready" condition, and all status, interrupt, and control indicators in the input/output system are reset. The I/O RESET switch does not affect any operations that may be processed in the central processor.

LOAD

The LOAD switch initializes memory for an input operation that uses the peripheral unit selected by the UNIT ADDRESS switches. The detailed operation of the loading process is described in the section "Loading Operation".

UNIT ADDRESS

The three UNIT ADDRESS switches are used to select the peripheral unit to be used in the loading process. The left switch has eight positions, numbered 0 through 7, designating an input/output processor. The center and right switches each have 16 positions, numbered 0 through F (hexadecimal) that designate a device controller/device under the control of the selected input/output processor.

SYSTEM RESET/CLEAR

The SYS RESET/CLEAR switch is used to reset all controls and indicators in the SIGMA 7 system. Pressing this switch causes the computer to perform all operations described for the CPU RESET/CLEAR switch, perform all operations described for the I/O RESET switch, initialize the memory control logic, and reset the MEMORY FAULT indicator.

The SYS RESET/CLEAR switch is also used in conjunction with the CPU RESET/CLEAR switch to reset core memory to 0's.

NORMAL MODE

The NORMAL MODE indicator is lighted when all the following conditions are satisfied:

1. The WATCHDOG TIMER switch is in the NORMAL position
2. The INTERLEAVE SELECT switch is in the NORMAL position
3. The PARITY ERROR MODE switch is in the CONT (continue) position
4. The CLOCK MODE switch is in the CONT (continuous) position
5. All logic power margins are "normal"

If any of the above conditions is not satisfied, the NORMAL MODE indicator is unlighted.

RUN

The RUN indicator is lighted when the COMPUTE switch is in the RUN position and no halt condition exists.

WAIT

The WAIT indicator is lighted when any of the following halt conditions exist:

1. The computer is executing a WAIT instruction
2. The program is stopped because of the ADDRESS STOP switch
3. The computer is halted because of the PARITY ERROR MODE switch

INTERRUPT

The INTERRUPT switch is used by the operator to activate the control panel interrupt. If the control panel interrupt (level X'5D') is armed when the INTERRUPT switch is pressed, a single pulse is transmitted to the interrupt level, advancing it to the waiting state. The INTERRUPT switch is lighted when the control panel interrupt level is in the waiting state, and remains lighted until the interrupt level advances to the active state (at which time the INTERRUPT switch is turned off). If the control panel interrupt level is disarmed (or already in the active state) when the INTERRUPT switch is pressed, no computer or control panel action occurs. If the control panel interrupt level advances to the waiting state and the level is disabled, the INTERRUPT switch remains lighted until the level is either enabled and allowed to advance to the active state or is returned to the armed or disarmed state. The INTERRUPT switch is always operative, both on the processor control panel and on the free-standing console.

PROGRAM STATUS DOUBLEWORD

Two rows of binary indicators are used to display the current program status doubleword (PSD). For the convenience of use and display, the second portion of the PSD, labeled PSW2, is arranged above the first portion, labeled PSW1. The PSD display consists of the indicators shown in Table 9.

INSERT

The INSERT switch is used to make changes in the program status doubleword. The switch is inactive in the center position and is momentary in the upper (PSW2) and lower (PSW1) positions. When the INSERT switch is moved to the

Table 9. Program Status Doubleword Display

	Indicator	Function	PSD Bit Posiiton	PSD Designation
PSW2	WRITE KEY	Write key	34-35	WK
	INTRPT INHIBIT	Interrupt inhibits	37-39	CI, II, EI
	CTR	Counter interrupt group inhibit	37	CI
	I/O	Input/output interrupt group inhibit	38	II
	EXT	External interrupts inhibit	39	EI
	POINTER	Register block pointer	55-59	RP
PSW1	CONDITION CODE	Condition code	0-3	CC
	FLOAT MODE	Floating-point mode controls	5-7	FS, FZ, FN
	SIG	Significance trap mask	5	FS
	ZERO	Zero trap mask	6	FZ
	NRMZ	Normalize mask	7	FN
	MODE	Machine state/memory map controls	8-9	MS, MM
	SLAVE	Master/slave mode control	8	MS
	MAP	Memory map control	9	MM
	TRAP	Arithmetic trap masks	10, 11	DM, AM
	DEC	Decimal arithmetic fault trap mask	10	DM
	ARITH	Fixed-point arithmetic overflow trap mask	11	AM
	INSTRUCTION ADDRESS	Address of next instruction to be executed	15-31	IA

PSW1 or PSW2 position, the corresponding indicators in the program status doubleword are altered (or unchanged, according to current state of the 32 DATA switches below the DISPLAY indicators).

INSTR ADDR

The INSTR ADDR (instruction address) switch is inactive in the center position; the upper position (HOLD) is latching and the lower position (INCREMENT) is momentary. When the switch is placed in the HOLD position, the normal process of incrementing the instruction address portion of the program status doubleword with each instruction execution in inhibited. If the COMPUTE switch is placed in the RUN position while the INSTR ADDR switch is at HOLD, the instruction in the location pointed to by the value of the INSTRUCTION ADDRESS indicators is executed, repeatedly, with the INSTRUCTION ADDRESS indicators remaining unchanged. If the COMPUTE switch is moved to the STEP position while the INSTR ADDR switch is at HOLD, the instruction is executed once each time the COMPUTE switch is moved to STEP; the INSTRUCTION ADDRESS indicators remain unchanged unless the instruction is LPSD, XPSD, or a branch instruction with the branch condition satisfied.

The following operations are performed each time the INSTR ADDR switch is moved from the center position to the INCREMENT position:

1. The current value of the INSTRUCTION ADDRESS indicators is incremented by 1.
2. Using the new value of the INSTRUCTION ADDRESS indicators, the contents of the location pointed to by the INSTRUCTION ADDRESS is displayed in the DISPLAY indicators.

ADDR STOP

The ADDR STOP (address stop) switch is used (with the COMPUTE switch in the RUN position) to cause the central processor to establish a halt condition and turn on the WAIT indicator whenever the CPU accesses the memory location whose address is equal to the SELECT ADDRESS value.

When the halt condition occurs, the instruction in the location pointed to by the INSTRUCTION ADDRESS indicators appears in the DISPLAY indicators. The displayed instruction is the one that would have been executed next, had the halt condition not occurred. If the halt condition is caused by an instruction access, the value of the INSTRUCTION ADDRESS indicators (at the time of the halt) is equal to the SELECT ADDRESS value. If the halt condition is caused by execution of an instruction with an indirect reference address equal to the SELECT ADDRESS value (i.e., by a direct address fetch), is caused by an instruction operand fetch, or is caused by an unsatisfied conditional branch instruction whose effective address is equal to the SELECT ADDRESS value, the value of the INSTRUCTION ADDRESS indicators (at the time of the halt) is 1 greater than the address of the instruction that referenced the SELECT ADDRESS value.

If an interrupt or trap condition is detected after the ADDRESS STOP halt condition is detected and before the CPU reaches the normal ADDRESS STOP halt phase, the CPU executes the instruction in the appropriate interrupt or trap location and then enters the ADDRESS STOP halt phase. In this case; the value of the INSTRUCTION ADDRESS indicators (at the time of the halt) is equal to the address of the next instruction in logical sequence after the instruction in the interrupt or trap location.

The ADDRESS STOP halt condition is reset when the COMPUTE switch is moved from RUN to IDLE; if the COMPUTE switch is then moved back to RUN (or to STEP), the instruction shown in the DISPLAY indicators is the next instruction executed.

SELECT ADDRESS

The SELECT ADDRESS switches select the address at which a program is to be halted (when used in conjunction with the ADDR STOP switch), select the address of a location to be altered (when used in conjunction with the STORE switch), and select the address of a word to be displayed (when used in conjunction with the DISPLAY switch). Each SELECT ADDRESS switch represents a 1 when it is in the upper position, and represents a 0 in the lower position.

STORE

The STORE switch is used to alter the contents of a general register or a memory location. The switch is inactive in the center position and is momentary in the INSTR ADDR and SELECT ADDR positions. When the switch is moved to the INSTR ADDR position, the current value of the DISPLAY indicators is stored in the location pointed to by the INSTRUCTION ADDRESS indicators; when the switch is moved to the SELECT ADDR position, the current value of the DISPLAY indicators is stored in the location pointed to by the SELECT ADDRESS switches.

DISPLAY

The DISPLAY switch is used to display the contents of a general register or memory location. The switch is inactive in the center position and is momentary in the INSTR ADDR and SELECT ADDR positions. When the switch is moved to the INSTR ADDR or SELECT ADDR position, the word in the location pointed to by the indicators or switches, respectively, is loaded into the instruction register and displayed with the DISPLAY indicators.

The 32 DISPLAY indicators are used to display a computer word, when used together with the INSTR ADDR, STORE, DISPLAY, and DATA switches. The DISPLAY indicators represent the current contents of the internal CPU instruction register.

DATA

The 32 DATA switches beneath the DISPLAY indicators are used to alter the contents of the program status doubleword (when used in conjunction with the INSERT switch) and to alter the value of the DISPLAY indicators (when used in conjunction with the single DATA switch). Each of the 32 DATA switches is inactive in the center position and is latching in both the upper (1) and lower (0) positions. In the center position, a DATA switch represents no change, in the upper or lower position it represents a 1 or 0, respectively.

The single DATA switch is used to change the state of the DISPLAY indicators. The switch is inactive in the center position and is momentary in the CLEAR and ENTER positions. When the switch is moved to the CLEAR position, all the DISPLAY indicators are reset (turned off). When the switch is moved to the ENTER position, the display indicators are not affected in those positions corresponding to DATA switches that are in the center position, but if a DATA switch is in the 1 or 0 position, that value is inserted into the corresponding indicator.

COMPUTE

The COMPUTE switch is used to control the execution of instructions. The center position (IDLE) and the upper position (RUN) are both latching, and the lower position (STEP) is momentary. When the COMPUTE switch is in the IDLE position, all other control panel switches are operative and the ADDRESS STOP halt and the WAIT instruction halt conditions are reset (cleared). If the computer is in a halt condition as a result of a memory parity error, moving the COMPUTE switch to IDLE does not clear the memory parity halt condition. This condition can be cleared only by pressing the SYS RESET/CLEAR switch.

When the COMPUTE switch is moved from IDLE to RUN, the RUN indicator is lighted and the computer begins to execute instructions (at machine speed) as follows

1. The current setting of the DISPLAY indicators is taken as the next instruction to be executed, regardless of the contents of the location pointed to by the current value of the INSTRUCTION ADDRESS indicators.
2. The value of the INSTRUCTION ADDRESS indicators is incremented by 1 unless the instruction in the DISPLAY indicators was LPSD, XPSD, or a branch instruction and the branch should occur (in which case the INSTRUCTION ADDRESS indicators are set to the value established by the LPSD, XPSD, or branch instruction).
3. Instruction execution continues with the instruction in the location pointed to by the new value of the INSTRUCTION ADDRESS indicators.

When the COMPUTE switch is in the RUN position, the only switches that are operative are the POWER switch, the INTERRUPT switch, the ADDR STOP switch, the INSTR ADDR switch (in the HOLD position), and the switches in the maintenance section.

Each time the COMPUTE switch is moved from the IDLE to the STEP position, the following operations occur:

1. The current setting of the DISPLAY indicators is taken as an instruction, and the single instruction is executed.
2. The current value of the INSTRUCTION ADDRESS indicators is incremented by 1 unless the "stepped" instruction was LPSD, XPSD, or branch instruction and the branch should occur (in which case the INSTRUCTION ADDRESS indicators are set to the value established by the LPSD, XPSD, or branch instruction).

3. The instruction in the location pointed to by the new value of the INSTRUCTION ADDRESS indicator is displayed in the DISPLAY indicators.

If an instruction is being stepped (executed by moving the COMPUTE switch from IDLE to STEP), all interrupt levels are temporarily inhibited while the instruction is being executed; however, a trap condition can occur while the instruction is being executed. In this case, the XPSD instruction in the appropriate trap location is executed as if the COMPUTE switch were in the RUN position. Thus, if a trap condition occurs during a stepped instruction, the program status doubleword display automatically reflects the effects of the XPSD instruction and the DISPLAY indicators then contain the first instruction of the trap routine.

CONTROL MODE

The CONTROL MODE switch is a three-position, key-operated locking switch. When the switch is in the REMOTE position, all controls on the free-standing console are operative. In addition, all controls and indicators in the maintenance section of the PCP are operative (except for the SENSE and CLOCK MODE switches) and all indicators in the lower portion of the PCP continue to display the same information as the equivalent indicators on the free-standing console. However, all of the controls in the lower portion of the PCP (except for the POWER switch) are inoperative. The POWER switch is always operative (on both the PCP and the free-standing console); in order for the system to be operative, both switches must indicate that power is on.

When the CONTROL MODE switch is in the LOCAL position, all controls on the PCP are operative. In addition, all indicators on the free-standing console continue to display the same information as the equivalent indicators on the PCP. However, all of the controls on the free-standing console (except for the POWER switch) are inoperative. The COMPUTE switches on both the PCP and the free-standing console must be in their IDLE positions whenever the CONTROL MODE switch is moved either from the REMOTE to the LOCAL position or from the LOCAL to the REMOTE position; otherwise, an undefined operation occurs.

When the CONTROL MODE switch is in the LOCK position, all controls on the free-standing console (except for POWER, INTERRUPT, and SENSE) are inoperative and all controls on the PCP (except for POWER, INTERRUPT, SENSE, and AUDIO) are inoperative. However, all indicators on both the free-standing console and the PCP continue to indicate the various computer states. The AUDIO switch is not affected by the position of the CONTROL MODE switch. In addition, the following switches (both on the PCP and on the free-standing console) are operative when the CONTROL MODE switch is in the LOCK position:

1. The POWER switch remains operative to allow for situations in which power must be removed from the system. System power is present only if both POWER switches indicate that power is on.
2. The INTERRUPT switch remains operative to allow the operator to interrupt the program being executed. If either INTERRUPT switch is pressed, the control panel interrupt level is triggered.
3. The SENSE switches remain operative to allow the operator to provide information to the program being executed. If a RD or WD instruction is executed in the internal control mode while the switch is in the LOCK position, the resulting condition code value is the logical sum (inclusive OR) of the PCP and free-standing console SENSE switches.

Certain switches on the PCP are locked to specific states when the CONTROL MODE switch is in the LOCK position. The affected switches and their locked states are:

Switch	Locked State
COMPUTE	RUN
WATCHDOG TIMER	NORMAL
INTERLEAVE SELECT	NORMAL
PARITY ERROR MODE	CONT
CLOCK MODE	CONT

The COMPUTE switch on the PCP must be in the RUN position whenever the CONTROL MODE switch is moved either from the LOCAL to the LOCK position or from the LOCK to the LOCAL position; otherwise, an undefined operation may occur.

MEMORY FAULT

The MEMORY FAULT indicators each correspond to a specific memory module. Whenever a memory parity error occurs in a memory module, the appropriate indicator is lighted and remains lighted until the indicators are reset. When a memory parity error occurs, an interrupt pulse is also transmitted to the memory parity interrupt level.

The MEMORY FAULT indicators are reset whenever the SYS RESET/CLEAR switch is pressed or whenever the computer executes a READ DIRECT instruction coded to read the MEMORY FAULT indicators. If the reason for a MEMORY FAULT indicator being on is overtemperature, and the condition still exists when the indicators are reset, the indicator is immediately turned on again.

ALARM

The ALARM indicator is used to attract the computer operator's attention, and is turned on and off (under program control) by executing a properly coded WRITE DIRECT instruction. When the ALARM indicator is lighted and the AUDIO switch is ON, a 1000-Hz signal is sent to the computer speaker; when the AUDIO switch is not in the ON position, the speaker is disconnected. (The AUDIO switch does not affect the state of the ALARM indicator.) The ALARM indicator is reset (turned off) whenever either the CPU RESET/CLEAR or the SYS RESET/CLEAR switch is pressed.

AUDIO

The AUDIO switch controls all signals to the computer speaker, whether from the ALARM indicator or from the program-controlled frequency flip-flop.

WATCHDOG TIMER

The WATCHDOG TIMER switch is used to override the instruction watchdog timer. When this switch is at NORMAL, the watchdog timer is operative; when the switch is in the OVERRIDE position, the watchdog timer is inactive.

INTERLEAVE SELECT

The INTERLEAVE SELECT switch is used to override the normal operation of interleaved memory modules. When this switch is in the NORMAL position, memory address interleaving occurs normally; however, when the switch is in the DIAGNOSTIC position, memory addresses are not interleaved between core memory modules.

PARITY ERROR MODE

The PARITY ERROR MODE switch controls the action of the computer when a memory parity error occurs. If the PARITY ERROR MODE switch is in the CONT (continue) position when a parity error occurs, the appropriate MEMORY FAULT indicator is turned on and an interrupt pulse is transmitted to the memory parity interrupt level. If the switch is in the HALT position when a parity error occurs, the appriate MEMORY FAULT indicator is turned on and the computer enters a "halt" state; the memory module in which the parity error occurred is unavailable to any access until the MEMORY FAULT indicators are reset. If the COMPUTE switch is in the RUN position during a halt, the WAIT indicator is lighted; however, the COMPUTE switch cannot be used alone to proceed from a halt caused by a parity error. In order to proceed, the SYS RESET/CLEAR switch must first be pressed.

PHASES

The PHASES indicators, used for maintenance functions, display certain internal operating phases of the computer. The PREPARATION indicators display computer phases during the preparation portion of an instruction cycle. The PCP (processor control panel) indicators display computer phases during processor control panel operations. The EXECUTION indicators display computer phases during the execution portion of an instruction cycle. The INT/TRAP (interrupt/trap) indicators are individually lighted when an interrupt, or trap condition occurs. When the COMPUTE switch is in the IDLE position, all of the PHASES indicators are normally off except for the center PCP indicator (phase 2 is the "idle" phase for processor control panel functions).

REGISTER SELECT

The REGISTER SELECT switch is used to display the contents of selected internal registers. When the REGISTER DISPLAY switch is in the inactive position, the DISPLAY indicators display the current contents of the internal instruction register. When the COMPUTE switch is in the IDLE position, the register selected by the REGISTER SELECT switch may be shown in the DISPLAY indicators by moving the REGISTER DISPLAY switch to the ON position.

SENSE

The four SENSE switches are used, under program control, to set the condition code portion of the program status doubleword. When a READ DIRECT or WRITE DIRECT instruction is executed in the internal control mode, the condition code is set according to the state of the four SENSE switches. If a SENSE switch is in the set (1) position, the corresponding bit of the condition code is set to 1; if a SENSE switch is in the reset (0) position, the corresponding bit of the condition code is reset to 0. The SENSE switches on the PCP are operative only if the CONTROL MODE switch is in either the LOCAL position or the LOCK position.

CLOCK MODE

The CLOCK MODE switch controls the internal computer clock. When the switch is in the CONT (continuous) position, the clock operates at normal speed. However, when the CLOCK MODE is in the inactive (center) position, the clock enters an idle state and can be made to generate one clock pulse each time the switch is moved to the SINGLE STEP position. When the clock is pulsed by the CLOCK MODE switch, the PHASE indicators reflect the computer phase during each pulse of the clock.

LOADING OPERATION

This section describes the procedure for initially loading programs into core memory from certain peripheral units attached to an input/output processor in the SIGMA 7 system. The computer operator may initiate a loading operation from the processor control panel (with the CONTROL MODE switch in the LOCAL position) or from the free-standing console (with the CONTROL MODE switch in the REMOTE position).

The LOAD switch and the UNIT ADDRESS switches are used to prepare a SIGMA 7 computer for a load operation. When the LOAD switch is pressed, the following bootstrap program is stored in core memory locations X'20' through X'29':

Location (Hex.)	Location (Dec.)	Contents (Hexadecimal)	Symbolic form of Instruction	
20	32	00000000		
21	33	00000000		
22	34	020000A8		
23	35	0E000058		
24	36	00000011		
25	37	00000xxx†		
26	38	32000024	LW, 0	36
27	39	CC000025	SIO, 0	*37
28	40	CD000025	TIO, 0	*37
29	41	69C00028	BCS, 12	40

When the LOAD switch is pressed, the selected peripheral device is not activated, and no other indicators or controls are affected; only core memory is altered.

†The x's in location X'25' represent the value of the UNIT ADDRESS switches at the time the LOAD switch is pressed.

LOAD PROCEDURE

To assure correct operation of the loading process, the following sequence should always be used when initiating a load operation:

1. Place the COMPUTE switch in the IDLE position.
2. Press the SYS RESET/CLEAR switch.
3. Set the UNIT ADDRESS switches to the address of the desired peripheral unit.
4. Press the LOAD switch.
5. Place the COMPUTE switch in the RUN position.

After the COMPUTE switch is placed in the RUN position, in step 5, the following actions occur:

1. The first record on the selected peripheral device is read into memory 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 loading are capable of accepting 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'.

LOAD OPERATION DETAILS

The first executed instruction of the bootstrap program (in location X'26') loads general register 0 with the doubleword address of the first I/O command doubleword. The I/O address for the SIO instruction in location X'27' is the 11 low-order bits of location X'25' (which have been set equal to the load unit address as a result of pressing the LOAD switch). During the SIO instruction, general register 0 points to locations X'22' and X'23' as the first I/O command doubleword for the selected device. This command doubleword contains an order that instructs the selected peripheral device to read 88 (X'58') bytes into consecutive memory locations starting at word location X'2A' (byte location X'A8'). At the completion of the read operation, neither data chaining nor command chaining is called for in the I/O command doubleword. Also, the suppress incorrect length flag is set to 1 so that an incorrect length indication will not be considered an error. (This means that no transmission error halt will result if the first record is either less than or greater than 88 bytes. If the record is greater than 88 bytes, only the first 88 bytes will be stored in memory.) After the SIO instruction, the computer executes a TIO instruction with the same effective address 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 back to the TIO instruction as long as either CC1 or CC2 (or both) is set to 1. In normal operation, CC1 is reset to 0 and CC2 remains set to 1 until the device can accept another SIO instruction, at which time the next instruction will be 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 initiate an unusual end interrupt signal (as specified by the appropriate flags in the I/O command doubleword). The unusual end interrupt will be ignored, however, since all interrupt levels have been disarmed by pressing the SYS RESET/CLEAR switch prior to loading. The device will not accept another SIO while the device interrupt is pending and, therefore, the BCS instruction in location X'29' will continue to branch to location X'28'. The correct operator action at this point is to repeat the load procedure. 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; thus causing the BCS instruction to branch.

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	Binary	Most Significant Digits: 0 (0000)	1 (0001)	2 (0010)	3 (0011)	4 (0100)	5 (0101)	6 (0110)	7 (0111)	8 (1000)	9 (1001)	A (1010)	B (1011)	C (1100)	D (1101)	E (1110)	F (1111)
0	0000	NUL	DLE	ds		SP	&	-									0
1	0001	SOH	DC1	ss				/		a	j		\[1]	A	J		1
2	0010	STX	DC2	fs						b	k	s	{[1]	B	K	S	2
3	0011	ETX	DC3	si						c	l	t	}[1]	C	L	T	3
4	0100	EOT	DC4							d	m	u	[[1]	D	M	U	4
5	0101	HT	LF NL			Will not be assigned				e	n	v	][1]	E	N	V	5
6	0110	ACK	SYN							f	o	w		F	O	W	6
7	0111	BEL	ETB							g	p	x		G	P	X	7
8	1000	EOM BS	CAN							h	q	y		H	Q	Y	8
9	1001	ENQ	EM							i	r	z		I	R	Z	9
A	1010	NAK	SUB			¢[2]	!	^[1]	:								
B	1011	VT	ESC			.	$	,	#								
C	1100	FF	FS			<	*	%	@					Will not be assigned			
D	1101	CR	GS			(	)	_	'								
E	1110	SO	RS			+	;	>	=								
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 c 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 (rows)	Binary [1]	Most Significant Digits (col's.): 0 (x000)	1 (x001)	2 (x010)	3 (x011)	4 (x100)	5 (x101)	6 (x110)	7 (x111)
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	! [5]	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
10	1010	LF NL	SUB	*	:	J	Z	j	z
11	1011	VT	ESC	+	;	K	[[5]	k	{
12	1100	FF	FS	,	<	L	\	l	¦
13	1101	CR	GS	-	=	M	] [5]	m	}
14	1110	SO	RS	.	>	N	[4] ^ [5]	n	~ [4]
15	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 ∧.)

5 On the Xerox 7670 Remote Batch Terminal, the symbol

! is | (2-1)
[is ¢ (5-11)
] is ! (5-13)
^ is ¬ (5-14)

and none of the symbols appearing in columns 6-7 are provided. Except for the four symbol differences noted above, therefore, this terminal provides all the characters in the 64-character ANSCII set.

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 Sigma EDIT BYTE STRING (EBS) instruction – not input/output control codes.
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			24 through 2E are unassigned.
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. On model 7670, ‘ not available, and ¢ = ANSCII 5-11.
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	On Model 7670, ¦ not available, and | = ANSCII 2-1.
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	On Model 7670, ! is ǀ.
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	On Model 7670, ~ is not available, and ¬ = ANSCII 5-14.
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 7670 ^ is ¬. 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	On Model 7670, [is ¢.
B5	181	]	12-11-0-5	5-13	right bracket	On Model 7670,] is !.
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

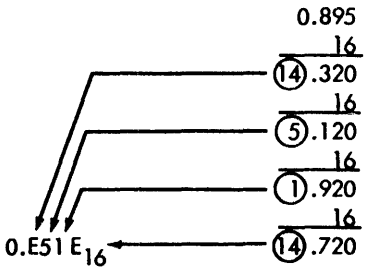
$$CA9\,BF3_{16} = 13\,278\,195_{10}$$

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

HEXADECIMAL - DECIMAL INTEGER CONVERSION TABLE (cont.)

	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. REFERENCE DIAGRAMS

This appendix contains flow diagrams that are intended to illustrate the major operations involved during the execution of instructions by the SIGMA 7 computer. The flow diagrams are not intended to depict actual computer operations and sequences, but the operations and sequences shown are valid representations of the internal computer operations. The symbolic notation used in the flow diagrams is consistent with that used in other portions of this reference manual. The symbolic terms used are:

Term	Meaning
A	An internal CPU register used to hold an operand obtained from the general register that is specified by the R field value in the instruction word.
AC	Access control code – the code used to determine whether or not a slave program operating with the memory map may read from, access instruction from, or write into a specific page of virtual addresses.
ADDR	Address – any virtual address.
B	An internal CPU register used to hold an operand obtained from the general register that is specified by the value produced by performing a logical OR between the R field of the instruction and the value 1.
C	An internal CPU register used to hold an immediate operand obtained from the instruction, or a byte, halfword, or word operand obtained from the memory (or general register) location specified by the effective address of the instruction. For doubleword operations, this register holds the 32 high-order bits of the effective doubleword.
D	An internal CPU register used to hold the 32 low-order bits of the effective doubleword in a doubleword operation.
EB	Effective byte.
EBL	Effective byte location.
ED	Effective doubleword
EDL	Effective doubleword location.
EH	Effective halfword.
EHL	Effective halfword location.
EW	Effective word.
EWL	Effective word location.
I	Instruction register.
IA	Instruction address.
IRA	Indirect reference address.
MA	Memory Address – an actual core memory address.
OP	Operation code – bits 1-7 of an instruction word.
R	General register address value.
TCC	Trap condition code – the code that is used during the EXCHANGE PROGRAM STATUS DOUBLEWORD (XPSD) instruction.
TYPE	Memory access type – the following values are used to indicate the reason for accessing memory: 0 = write 1 = instruction read 2 = operand read
WK	Write key
WL	Write lock
X	Index register designator.

BASIC SIGMA 7 INSTRUCTION EXECUTION CYCLE

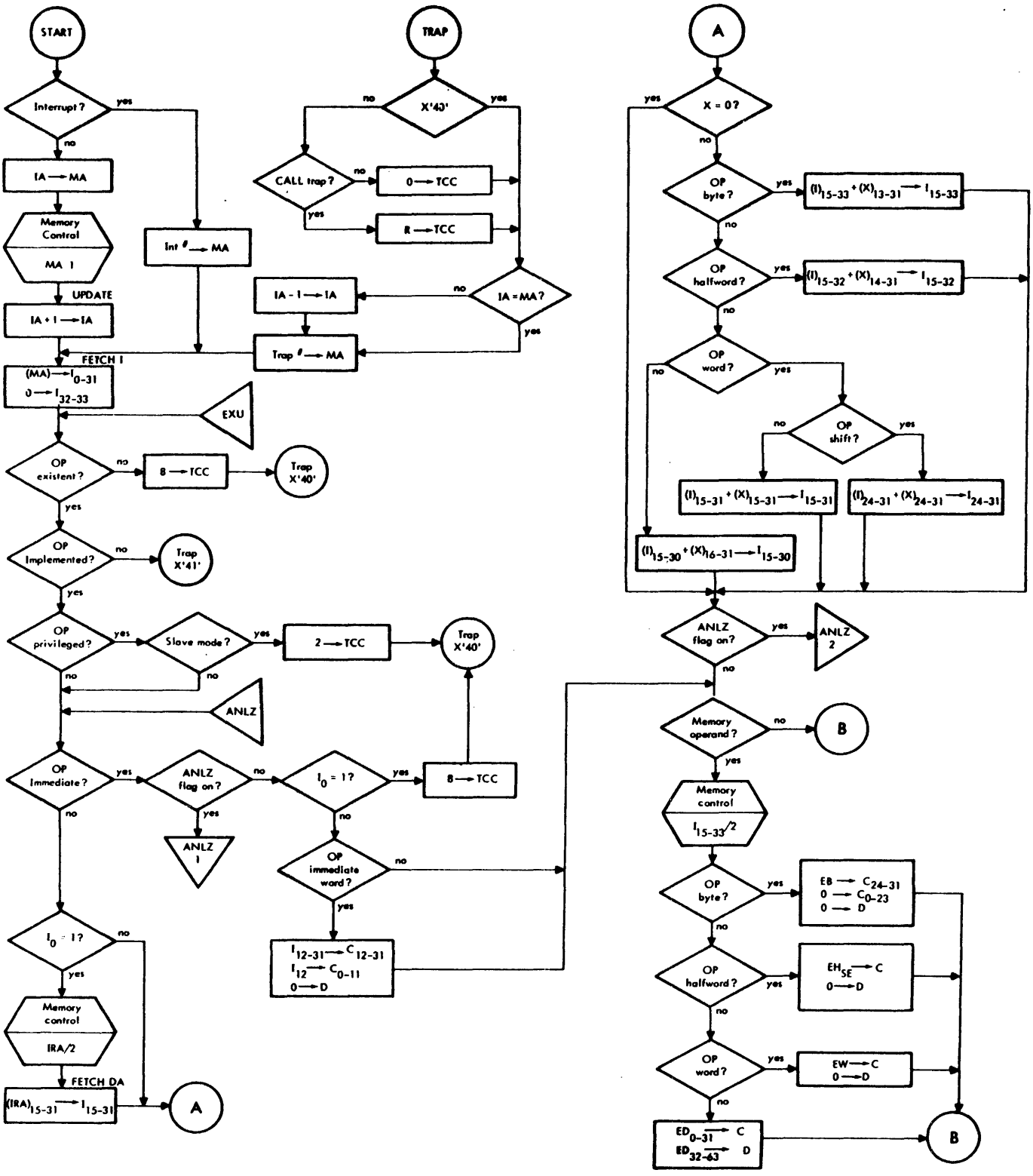
The hexagonal elements in the flow diagram labeled "Memory Control" refer to the memory request process shown at the right of the basic flow diagram. The memory request process is represented as a subroutine with two inputs: an address value (ADDR) and a memory access TYPE, separated by a slash, that correspond to the values shown in the "Memory Control" elements of the basic flow diagram.

The circular entry point labeled "TRAP" is a continuation of the circular exit points labeled "Trap X'n'", where n is the appropriate trap location.

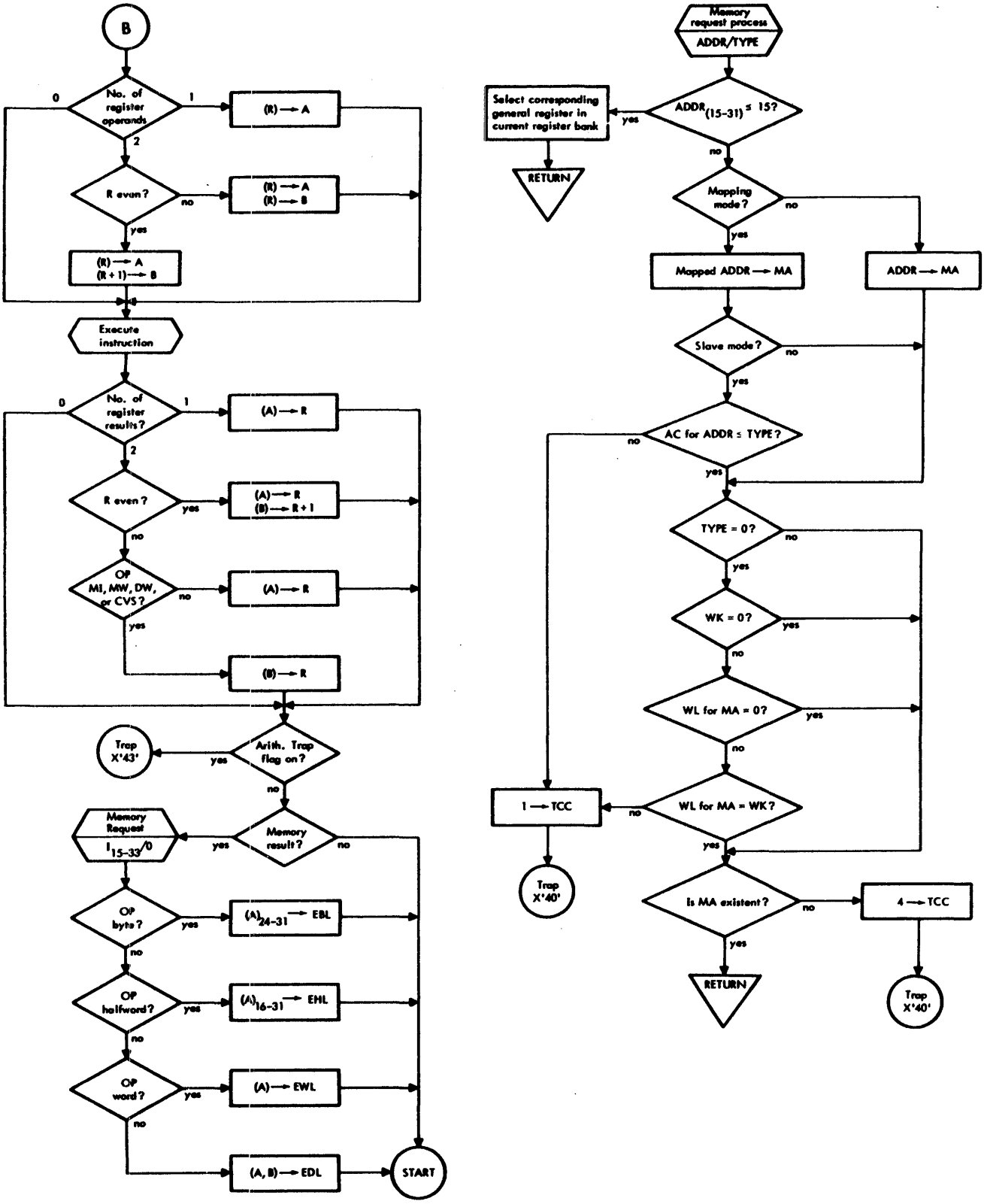
The triangular entry point labeled "EXU" indicates the point in the basic flow diagram at which an instruction (being executed as an operand of the EXECUTE instruction) is started.

The triangular entry point labeled "ANLZ" indicates the point in the basic flow diagram at which the effective address computation for the instruction being analyzed is started; the triangular exit points indicate the completion of the effective address calculation.

BASIC SIGMA 7 INSTRUCTION EXECUTION CYCLE

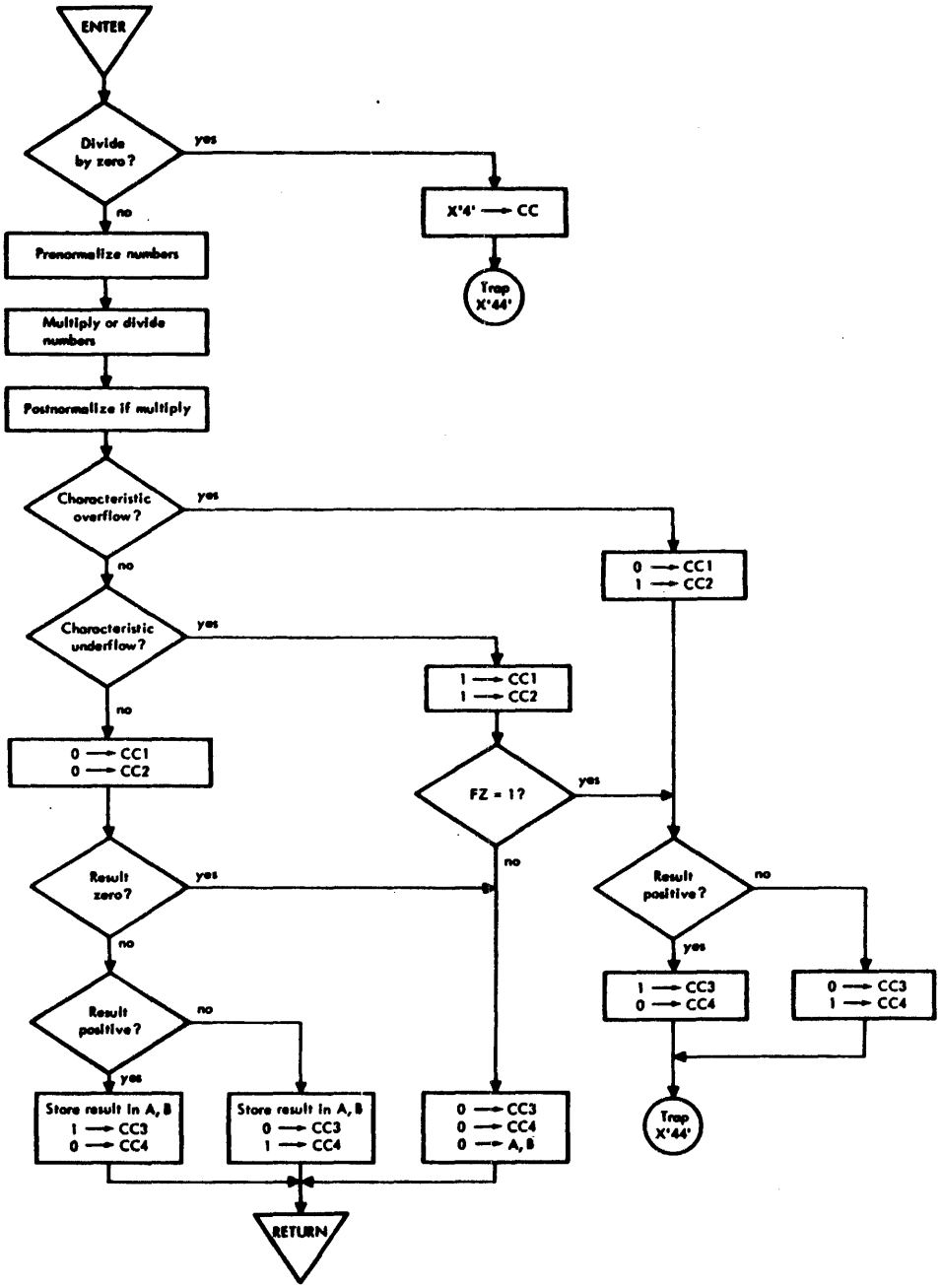


BASIC SIGMA 7 INSTRUCTION EXECUTION CYCLE (cont.)

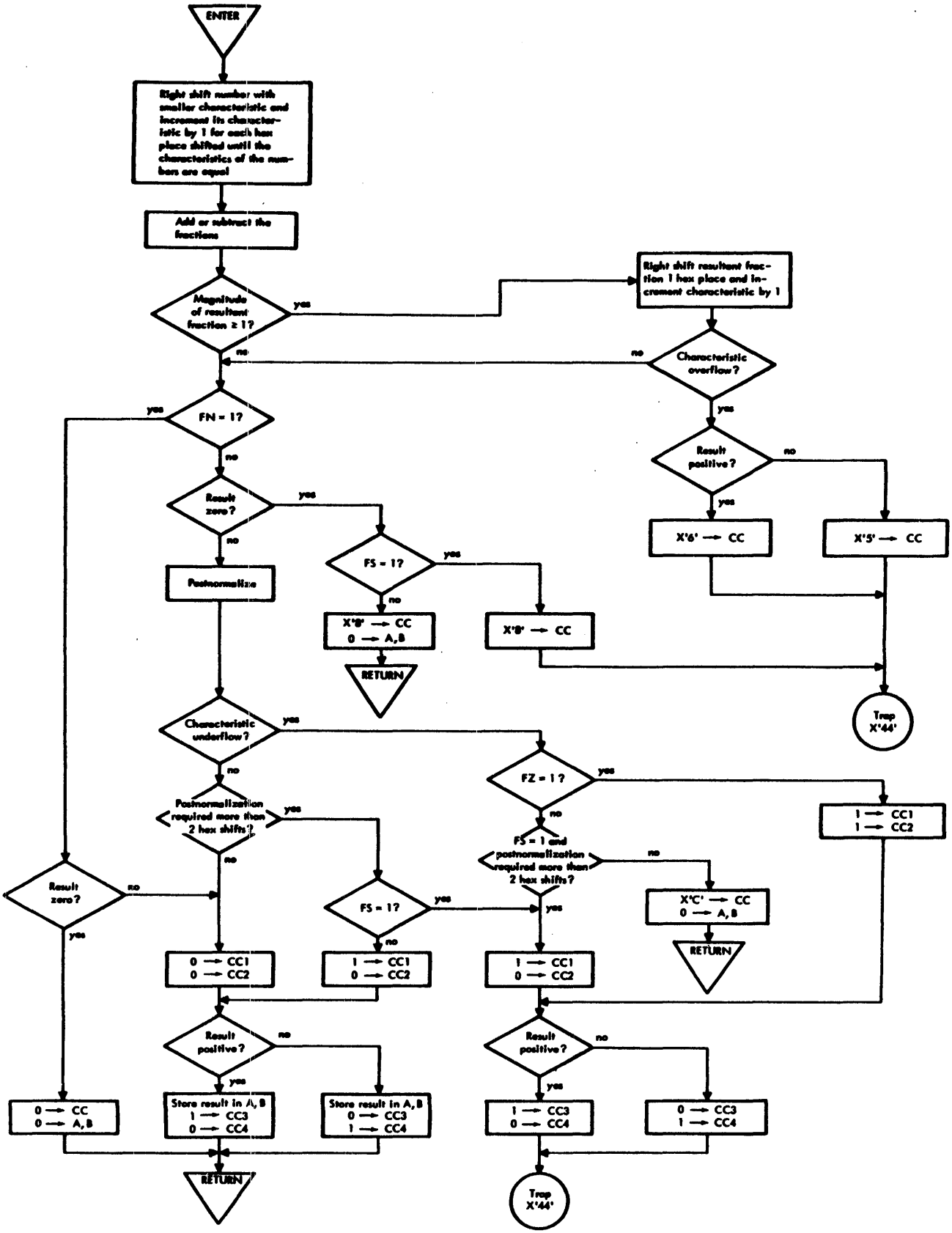


FLOATING- POINT INSTRUCTION EXECUTION

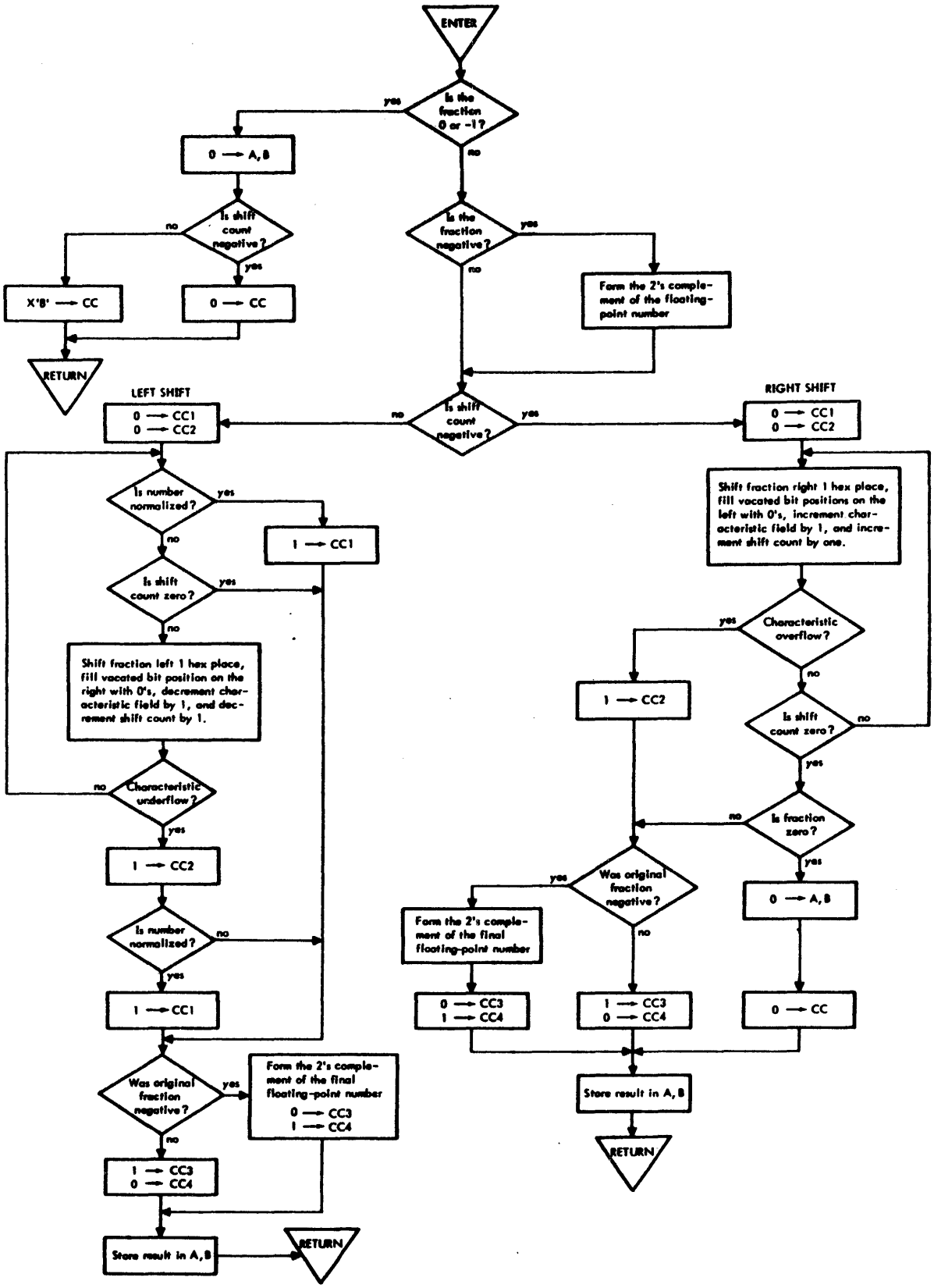
FLOATING-POINT MULTIPLICATION AND DIVISION



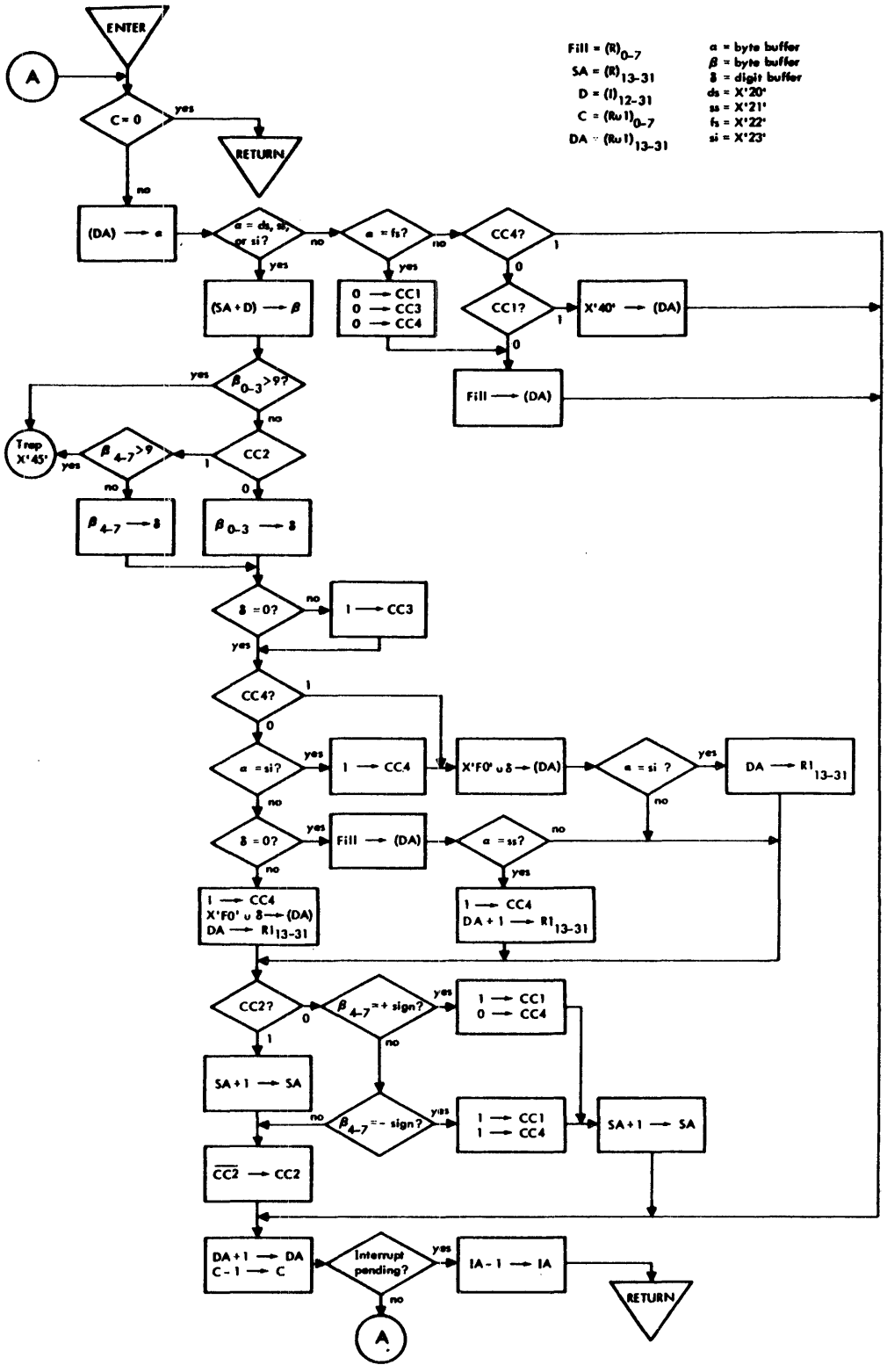
FLOATING-POINT ADDITION AND SUBTRACTION



FLOATING-POINT SHIFT



EDIT BYTE STRING INSTRUCTION EXECUTION



APPENDIX C. SIGMA 7 INSTRUCTIONS (MNEMONICS)

Mnemonic	Code	Instruction Name	Addressing Type	Page
AD	10	Add Doubleword	Doubleword	38
AH	50	Add Halfword	Halfword	37
AI	20	Add Immediate	Immediate, word	37
AIO	6E	Acknowledge I/O Interrupt (privileged)	Word	85
AND	4B	AND Word	Word	44
ANLZ	44	Analyze	Word	35
AW	30	Add Word	Word	38
AWM	66	Add Word to Memory	Word	41
BAL	6A	Branch and Link	Word	72
BCR	68	Branch on Conditions Reset	Word	71
BCS	69	Branch on Conditions Set	Word	71
BDR	64	Branch on Decrementing Register	Word	72
BIR	65	Branch on Incrementing Register	Word	71
CAL1	04	Call 1	Word	72
CAL2	05	Call 2	Word	72
CAL3	06	Call 3	Word	72
CAL4	07	Call 4	Word	72
CB	71	Compare Byte	Byte	42
CBS	60	Compare Byte String	Immediate, byte	60
CD	11	Compare Doubleword	Doubleword	43
CH	51	Compare Halfword	Halfword	43
CI	21	Compare Immediate	Immediate, word	42
CLM	19	Compare with Limits in Memory	Doubleword	44
CLR	39	Compare with Limits in Register	Word	44
CS	45	Compare Selective	Word	43
CVA	29	Convert by Addition	Word	47
CVS	28	Convert by Subtraction	Word	48
CW	31	Compare Word	Word	43
DA	79	Decimal Add	Byte	55
DC	7D	Decimal Compare } optional	Byte	56
DD	7A	Decimal Divide } optional	Byte	56
DH	56	Divide Halfword	Halfword	40
DL	7E	Decimal Load } optional	Byte	54
DM	7B	Decimal Multiply } optional	Byte	55
DS	78	Decimal Subtract } optional	Byte	55
DSA	7C	Decimal Shift Arithmetic } optional	Byte	56
DST	7F	Decimal Store } optional	Byte	54
DW	36	Divide Word	Word	40
EBS	63	Edit Byte String (optional)	Immediate, byte	62
EOR	48	Exclusive OR Word	Word	44
EXU	67	Execute	Word	71
FAL	1D	Floating Add Long } optional	Doubleword	51
FAS	3D	Floating Add Short } optional	Word	51
FDL	1E	Floating Divide Long } optional	Doubleword	52
FDS	3E	Floating Divide Short } optional	Word	52
FML	1F	Floating Multiply Long } optional	Doubleword	52
FMS	3F	Floating Multiply Short } optional	Word	52
FSL	1C	Floating Subtract Long } optional	Doubleword	52
FSS	3C	Floating Subtract Short } optional	Word	51
HIO	4F	Halt Input/Output (privileged)	Word	84
INT	6B	Interpret	Word	36
LAD	1B	Load Absolute Doubleword	Doubleword	32
LAH	5B	Load Absolute Halfword	Halfword	31
LAW	3B	Load Absolute Word	Word	31
LB	72	Load Byte	Byte	30
LCD	1A	Load Complement Doubleword	Doubleword	31
LCF	70	Load Conditions and Floating Control	Byte	33

SIGMA 7 INSTRUCTIONS (MNEMONICS) (cont.)

Mnemonic	Code	Instruction Name		Addressing Type	Page
LCFI	02	Load Conditions and Floating Control Immediate		Immediate, word	33
LCH	5A	Load Complement Halfword		Halfword	31
LCW	3A	Load Complement Word		Word	31
LD	12	Load Doubleword		Doubleword	30
LH	52	Load Halfword		Halfword	30
LI	22	Load Immediate		Immediate, word	30
LM	2A	Load Multiple		Word	33
LPSD	0E	Load Program Status Doubleword	privileged	Doubleword	73
LRP	2F	Load Register Pointer	privileged	Word	75
LS	4A	Load Selective		Word	32
LW	32	Load Word		Word	30
MBS	61	Move Byte String		Immediate, byte	59
MH	57	Multiply Halfword		Halfword	39
MI	23	Multiply Immediate		Immediate, word	39
MMC	6F	Move to Memory Control (privileged)		Word	75
MSP	13	Modify Stack Pointer		Doubleword	69
MTB	73	Modify and Test Byte		Byte	41
MTH	53	Modify and Test Halfword		Halfword	41
MTW	33	Modify and Test Word		Word	42
MW	37	Multiply Word		Word	40
OR	49	OR Word		Word	44
PACK	76	Pack Decimal Digits (optional)		Byte	57
PLM	0A	Pull Multiple		Word	68
PLW	08	Pull Word		Word	67
PSM	0B	Push Multiple		Word	68
PSW	09	Push Word		Word	67
RD	6C	Read Direct (privileged)		Word	78
S	25	Shift		Word	45
SD	18	Subtract Doubleword		Doubleword	39
SF	24	Shift Floating		Word	46
SH	58	Subtract Halfword		Halfword	38
SIO	4C	Start Input/Output (privileged)		Word	81
STB	75	Store Byte		Byte	34
STCF	74	Store Conditions and Floating Control		Byte	35
STD	15	Store Doubleword		Doubleword	34
STH	55	Store Halfword		Halfword	34
STM	2B	Store Multiple		Word	35
STS	47	Store Selective		Word	34
STW	35	Store Word		Word	34
SW	38	Subtract Word		Word	38
TBS	41	Translate Byte String		Immediate, byte	61
TDV	4E	Test Device	privileged	Word	85
TIO	4D	Test Input/Output	privileged	Word	84
TTBS	40	Translate and Test Byte String		Immediate, byte	61
UNPK	77	Unpack Decimal Digits (optional)		Byte	57
WAIT	2E	Wait	privileged	Word	77
WD	6D	Write Direct	privileged	Word	78
XPSD	0F	Exchange Program Status Doubleword	privileged	Doubleword	73
XW	46	Exchange Word		Word	34

APPENDIX D. INSTRUCTION TIMING

This appendix shows the timing (in microseconds) for executing individual SIGMA 7 computer instructions under a variety of circumstances. All of the times are based on the assumption that whenever the CPU requests a service cycle from a particular memory bank, it never has to wait for such service due to other devices (such as IOPs) that are connected to that memory bank.

Execution times depend not only on the nature of the specific instructions, but also on the configuration of memory banks in the system, and the placement of instructions and operands. The following table provides a means of estimating instruction execution times for some of the possible combinations of memory bank configuration, data placement, and instruction type, where

MAX = Time with no memory overlap (i.e., all sequential memory accesses come from the same bank).

MIN = Time with complete memory overlap (i.e., all sequential memory accesses come from a bank not currently busy, that is, the bank being accessed is not being used by the CPU or any external IOP).

Memory Bank Configuration	Average Instruction Execution Time	
	Instructions that utilize byte, halfword, and word addressing	Instructions that utilize doubleword addressing
All instructions and operands are in the same memory bank.	MAX	MAX
All instructions are in one memory bank and all operands are in a different memory bank.	MIN	1/2 MAX + 1/2 MIN
All instructions and operands are in two interleaved memory banks.	1/2 MAX + 1/2 MIN	1/4 MAX + 3/4 MIN
All instructions and operands are in four interleaved memory banks.	1/4 MAX + 3/4 MIN	1/8 MAX + 7/8 MIN
All instructions are in one memory bank and all operands are in two interleaved memory banks. (Both operand memory banks are different from instruction memory bank.)	MIN	MIN

Basic timing information is summarized in the following two tables. A dash entry for any item indicates a nonapplicable or impossible condition for the instruction. Special notes (identified by numbers in the "Notes" column are given at the end of the table to which they apply. Table D-1 shows the execution times for instructions under the most common conditions that the user can expect to encounter in his program. Table D-2 shows the additional times that must be added to the basic times if (1) the instruction performs a register-to-register operation (i.e., accesses one or more of the general registers for an operand(s) or a direct address) or (2) the register pointer in the current program status doubleword selects one of the register blocks in the range from X'4' through X'1F' (4 through 31 decimal).

The times given in Table D-2, where the instruction performs a register-to-register operation, assume the following conditions:

1. The CPU is operating in the mapping mode with one memory bank so that no memory overlap occurs.
2. All instructions are in core memory.
3. In the case of an instruction with a direct address, its operand is in one or more of the general registers. For a push-down instruction with a direct address, however, its stack pointer doubleword is in the general registers and the stack is in core memory.
4. In the case of an instruction with an indirect address, the indirect reference is to one of the general registers, which contains the direct address of the operand. The resultant virtual address is assumed to be a core memory address. For a push-down instruction with an indirect address, therefore, both the stack pointer doubleword and the stack are assumed to be in core memory.

The timing data given below are for a typical system. A specific CPU may vary by up to ± 10% of the times shown.

For large core memory configurations, an additional .1 μsec per memory access may be encountered due to added cable lengths.

Table D-1. Basic Instruction Timing

Mnemonics	Notes	No Memory Overlap								Maximum Memory Overlap							
		No Map				Map				No Map				Map			
		Direct		Indirect		Direct		Indirect		Direct		Indirect		Direct		Indirect	
		No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index
AD		2.9	3.6	3.9	4.2	2.9	3.7	3.9	4.3	2.4	3.0	3.3	3.6	2.5	3.2	3.4	3.8
AH		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.9
AI		1.3	--	--	--	1.4	--	--	--	1.3	--	--	--	1.4	--	--	--
AIO	R / 0	6.9	6.9	7.5	7.5	6.9	6.9	7.5	7.5	6.6	6.6	7.2	7.2	6.7	6.7	7.3	7.3
AIO	R 0	6.1	6.1	6.7	6.7	6.1	6.1	6.7	6.7	6.1	6.1	6.7	6.7	6.1	6.1	6.7	6.7
AND		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.9
ANLZ	1	3.3	3.9	4.3	4.6	3.3	4.1	4.3	4.7	3.2	3.8	4.1	4.4	3.2	3.9	4.1	4.5
AW		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.9
AWM		3.0	3.6	3.9	4.2	3.1	3.8	4.0	4.4	2.6	3.3	3.6	3.9	2.9	3.6	3.8	4.2
BAL		2.3	2.3	2.9	2.9	2.4	2.4	3.0	3.0	2.2	2.2	2.8	2.8	2.3	2.3	2.9	2.9
BCR	branch	1.0	1.6	2.0	2.3	1.0	1.7	2.0	2.4	0.9	1.5	1.8	2.2	0.9	1.6	1.8	2.3
BCR	no branch	2.0	2.6	3.0	3.3	2.1	2.8	3.1	3.5	1.9	2.5	2.8	3.1	2.0	2.7	2.9	3.3
BCS	branch	1.0	1.6	2.0	2.3	1.0	1.7	2.0	2.4	0.9	1.5	1.8	2.2	0.9	1.6	1.8	2.3
BCS	no branch	2.0	2.6	3.0	3.3	2.1	2.8	3.1	3.5	1.9	2.5	2.8	3.1	2.0	2.7	2.9	3.3
BDR	branch	1.4	1.7	2.4	2.4	1.4	1.8	2.4	2.5	1.4	1.7	2.3	2.3	1.4	1.8	2.3	2.4
BDR	no branch	2.4	2.7	3.4	3.4	2.5	2.9	3.5	3.6	2.3	2.6	3.2	3.2	2.4	2.8	3.4	3.4
BIR	branch	1.4	1.7	2.4	2.4	1.4	1.8	2.4	2.5	1.4	1.7	2.3	2.3	1.4	1.8	2.3	2.4
BIR	no branch	2.4	2.7	3.4	3.4	2.5	2.9	3.5	3.6	2.3	2.6	3.2	3.2	2.4	2.8	3.4	3.4
CAL 1-4		3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.2	3.2	3.2	3.2	3.2	3.2	3.2	3.2
CB		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.9
CBS	2	4.1 +3.9N	--	--	--	4.2 +4.1N	--	--	--	4.1 +3.9N	--	--	--	4.2 +4.1N	--	--	--
CD		2.9	3.6	3.9	4.2	2.9	3.7	3.9	4.3	2.4	3.0	3.3	3.6	2.5	3.2	3.4	3.8
CH		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.9
CI		1.9	--	--	--	2.0	--	--	--	1.8	--	--	--	1.9	--	--	--
CLM		2.9	3.6	3.9	4.2	2.9	3.7	3.9	4.3	2.4	3.0	3.3	3.6	2.5	3.2	3.4	3.8
CLR		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.8	2.4	2.8	3.1	1.8	2.6	2.8	3.2
CS		3.0	3.6	4.0	4.3	3.1	3.8	4.1	4.5	2.9	3.5	3.8	4.1	3.0	3.7	3.9	4.3
CVA	3	17.1 +0.6N	17.1 +0.6N	17.6 +0.6N	17.6 +0.6N	17.1 +0.7N	17.1 +0.7N	17.8 +0.7N	17.8 +0.7N	17.1 +0.5N	17.1 +0.5N	17.3 +0.6N	17.3 +0.6N	17.2 +0.6N	17.2 +0.6N	17.3 +0.7N	17.3 +0.7N
CVS		34.7	34.7	35.2	35.2	38.4	38.4	38.5	38.5	33.2	33.2	33.7	33.7	36.8	36.6	36.7	36.7
CW		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.9
DA	4	19.2 +0.3D	19.2 +0.3D	20.0 +0.3D	20.0 +0.3D	19.4 +0.3D	19.4 +0.3D	20.6 +0.3D	20.6 +0.3D	19.2 +0.3D	19.2 +0.3D	20.0 +0.3D	20.0 +0.3D	19.4 +0.3D	19.4 +0.3D	20.6 +0.3D	20.6 +0.3D
DC	4	11.8 +0.3D	11.8 +0.3D	12.3 +0.3D	12.3 +0.3D	12.1 +0.3D	12.1 +0.3D	12.8 +0.3D	12.8 +0.3D	11.8 +0.3D	11.8 +0.3D	12.3 +0.3D	12.3 +0.3D	12.1 +0.3D	12.1 +0.3D	12.8 +0.3D	12.8 +0.3D
DD	5	29.7 +0.8K	29.7 +0.8K	30.3 +0.8K	30.3 +0.8K	30.8 +0.8K	30.8 +0.8K	31.4 +0.8K	31.4 +0.8K	29.7 +0.8K	29.7 +0.8K	30.3 +0.8K	30.3 +0.8K	30.8 +0.8K	30.8 +0.8K	31.4 +0.8K	31.4 +0.8K
DH		12.4	13.0	13.4	13.7	12.4	13.2	13.4	13.8	12.4	13.0	13.3	13.6	12.4	13.1	13.3	13.7
DL	4	11.8 +0.3D	11.8 +0.3D	12.4 +0.3D	12.4 +0.3D	11.8 +0.3D	11.8 +0.3D	12.5 +0.3D	12.5 +0.3D	11.8 +0.3D	11.8 +0.3D	12.4 +0.3D	12.4 +0.3D	11.8 +0.3D	11.8 +0.3D	12.5 +0.3D	12.5 +0.3D
DM	6	61.2 +0.4DN	61.2 +0.4DN	61.8 +0.4DN	61.8 +0.4DN	62.3 +0.4DN	62.3 +0.4DN	62.9 +0.4DN	62.9 +0.4DN	61.2 +0.4DN	61.2 +0.4DN	61.8 +0.4DN	61.8 +0.4DN	62.3 +0.4DN	62.3 +0.4DN	62.9 +0.4DN	62.9 +0.4DN
DS	4	19.2 +0.3D	19.2 +0.3D	19.7 +0.3D	19.7 +0.3D	19.3 +0.3D	19.3 +0.3D	19.7 +0.3D	19.7 +0.3D	19.2 +0.3D	19.2 +0.3D	19.7 +0.3D	19.7 +0.3D	19.3 +0.3D	19.3 +0.3D	19.7 +0.3D	19.7 +0.3D
DSA		20.3	20.3	20.9	20.9	20.3	20.3	21.0	21.0	20.2	20.2	20.6	20.6	20.2	20.2	20.9	20.9
DST	7	11.3 +0.7D	11.3 +0.7D	12.0 +0.7D	12.0 +0.7D	11.3 +0.7D	11.3 +0.7D	12.1 +0.7D	12.1 +0.7D	11.3 +0.7D	11.3 +0.7D	12.0 +0.7D	12.0 +0.7D	11.3 +0.7D	11.3 +0.7D	12.1 +0.7D	12.1 +0.7D

Table D-1. Basic Instruction Timing (cont.)

Mnemonics	Notes	No Memory Overlap								Maximum Memory Overlap							
		No Map				Map				No Map				Map			
		Direct		Indirect		Direct		Indirect		Direct		Indirect		Direct		Indirect	
		No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index
DW		12.6	13.2	13.5	13.8	12.5	13.2	13.6	13.9	12.5	13.1	13.4	13.6	12.5	13.2	13.5	13.8
EBS	8	4.1 +6.8N	--	--	--	4.2 +7.1N	--	--	--	4.1 +6.8N	--	--	--	4.2 +7.1N	--	--	--
EOR		1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.1	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.9
EXU	9	1.3	1.6	2.2	2.2	1.3	1.8	2.2	2.4	1.2	1.6	2.1	2.2	1.3	1.8	2.2	2.4
FAL min	10	4.1	4.7	5.0	5.3	4.2	4.9	5.1	5.5	4.1	4.7	5.0	5.3	4.2	4.9	5.1	5.5
FAL max	11	13.7	14.2	14.6	14.8	13.8	14.4	14.7	15.1	13.7	14.2	14.6	14.8	13.8	14.4	14.7	15.1
FAL typical	12	5.0	5.5	5.9	6.1	5.1	5.7	6.0	6.4	5.0	5.5	5.9	6.1	5.1	5.7	6.0	6.4
FAS min	10	3.3	3.9	4.2	4.6	3.3	4.0	4.2	4.7	3.3	3.9	4.2	4.6	3.3	4.0	4.2	4.7
FAS max	11	8.2	8.9	9.1	9.5	8.2	9.0	9.1	9.6	8.2	8.9	9.1	9.5	8.2	9.0	9.1	9.6
FAS typical	12	4.0	4.6	4.9	5.3	4.0	4.7	4.9	5.4	4.0	4.6	4.9	5.3	4.0	4.7	4.9	5.4
FDL min	13, 14	25.4	26.1	26.4	26.7	25.5	26.1	27.0	26.8	25.4	26.1	26.4	26.7	25.5	26.1	27.0	26.8
FDL max	11	34.7	35.4	35.7	36.0	34.8	35.4	36.3	36.1	34.7	35.4	35.7	36.0	34.8	35.4	36.3	36.1
FDS min	13, 14	12.4	13.3	13.4	13.7	12.4	13.4	13.4	13.8	12.4	13.3	13.4	13.7	12.4	13.4	13.4	13.8
FDS max	11	16.6	17.5	17.6	17.9	16.6	17.6	17.6	18.0	16.6	17.5	17.6	17.9	16.6	17.6	17.6	18.0
FML min	13, 14	9.1	9.8	10.0	10.4	9.2	10.0	10.2	10.6	9.1	9.8	10.0	10.4	9.2	10.0	10.2	10.6
FML max	11	14.7	15.4	15.6	16.0	14.8	15.6	15.8	16.2	14.7	15.4	15.6	16.0	14.8	15.6	15.8	16.2
FMS min	13, 14	6.0	6.6	6.9	7.2	6.0	6.8	6.9	7.4	6.0	6.6	6.9	7.2	6.0	6.8	6.9	7.4
FMS max	11	8.8	9.4	9.7	10.0	8.8	9.6	9.7	10.2	8.8	9.4	9.7	10.0	8.8	9.6	9.7	10.2
FSL min	10	4.1	4.7	5.0	5.3	4.2	4.9	5.1	5.5	4.1	4.7	5.0	5.3	4.2	4.9	5.1	5.5
FSL max	11	13.7	14.2	14.6	14.8	13.8	14.4	14.7	15.1	13.7	14.2	14.6	14.8	13.8	14.4	14.7	15.1
FSL typical	12	5.0	5.5	5.9	6.1	5.1	5.7	6.0	6.4	5.0	5.5	5.9	6.1	5.1	5.7	6.0	6.4
FSS min	10	3.3	3.9	4.2	4.6	3.3	4.0	4.2	4.7	3.3	3.9	4.2	4.6	3.3	4.0	4.2	4.7
FSS max	11	8.2	8.9	9.1	9.5	8.2	9.0	9.1	9.6	8.2	8.9	9.1	9.5	8.2	9.0	9.1	9.6
FSS typical	12	4.0	4.6	4.9	5.3	4.0	4.7	4.9	5.4	4.0	4.6	4.9	5.3	4.0	4.7	4.9	5.4
HIO	R = even, ≠0	9.7	9.7	10.3	10.3	9.7	9.7	10.3	10.3	9.4	9.4	10.0	10.0	9.5	9.5	10.1	10.1
HIO	R odd	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9
HIO	R 0	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7
INT		2.4	3.0	3.4	3.6	2.5	3.2	3.4	3.8	2.3	2.9	3.2	3.5	2.4	3.1	3.3	3.7
LAD		3.4	4.0	4.3	4.6	3.4	4.2	4.4	4.8	3.1	3.7	4.0	4.3	3.2	3.9	4.2	4.6
LAH		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.1
LAW		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.1
LB		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.1
LCD		2.9	3.6	3.9	4.2	2.9	3.7	3.9	4.3	2.4	3.0	3.3	3.6	2.5	3.2	3.4	3.8
LCF		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.1
LCFI		1.3	--	--	--	1.4	--	--	--	1.3	--	--	--	1.4	--	--	--
LCH		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.1
LCW		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.1
LD		2.9	3.6	3.9	4.2	2.9	3.7	3.9	4.3	2.4	3.0	3.3	3.6	2.5	3.2	3.4	3.8
LH		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.1
LI		1.3	--	--	--	1.4	--	--	--	1.3	--	--	--	1.4	--	--	--
LM	15	2.3 +1.0N	2.3 +1.0N	3.0 +1.0N	3.0 +1.0N	2.4 +1.1N	2.4 +1.1N	3.0 +1.1N	3.0 +1.1N	2.2 +1.0N	2.2 +1.0N	2.8 +1.0N	2.8 +1.0N	2.3 +1.1N	2.3 +1.1N	2.8 +1.1N	2.8 +1.1N
LPSD		4.4	4.4	5.0	5.0	4.7	4.7	5.2	5.2	4.4	4.4	5.0	5.0	4.7	4.7	5.2	5.2
LRP		2.2	2.8	3.1	3.4	2.3	3.0	3.2	3.6	2.2	2.8	3.1	3.4	2.3	3.0	3.2	3.6

Table D-1. Basic Instruction Timing (cont.)

Mnemonics	Notes	No Memory Overlap								Maximum Memory Overlap							
		No Map				Map				No Map				Map			
		Direct		Indirect		Direct		Indirect		Direct		Indirect		Direct		Indirect	
		No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index
LS		2.5	3.1	3.4	3.7	2.6	3.3	3.5	3.9	2.5	3.1	3.4	3.7	2.6	3.3	3.5	3.9
LW		1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.2	1.4	2.0	2.3	2.6	1.5	2.2	2.4	3.0
MBS word	2	4.2 +0.8N	--	--	--	4.4 +0.8N	--	--	--	4.2 +0.8N	--	--	--	4.4 +0.8N	--	--	--
MBS byte	2	4.2 +3.4N	--	--	--	4.3 +3.4N	--	--	--	4.2 +3.4N	--	--	--	4.3 +3.4N	--	--	--
MH		3.8	4.4	4.8	5.1	3.9	4.7	4.9	5.3	3.8	4.4	4.8	5.1	3.9	4.7	4.9	5.3
MI		5.0	--	--	--	5.1	--	--	--	5.0	--	--	--	5.1	--	--	--
MMC	15	3.0 +3.0N	--	--	--	3.1 +3.1N	--	--	--	3.0 +2.9N	--	--	--	3.1 +3.0N	--	--	--
MSP		7.6	8.2	8.5	8.8	8.0	8.7	8.9	9.3	7.4	8.0	8.3	8.6	8.0	8.7	8.9	9.3
MTB	R ≠ 0	3.6	4.2	4.6	4.9	3.7	4.4	4.7	5.1	3.6	4.2	4.6	4.9	3.7	4.4	4.7	5.1
MTB	R = 0	2.6	3.2	3.6	3.9	2.7	3.5	3.7	4.1	2.6	3.2	3.6	3.9	2.7	3.5	3.7	4.1
MTH	R ≠ 0	3.6	4.2	4.6	4.9	3.7	4.4	4.7	5.1	3.6	4.2	4.6	4.9	3.7	4.4	4.7	5.1
MTH	R = 0	2.6	3.2	3.6	3.9	2.7	3.5	3.7	4.1	2.6	3.2	3.6	3.9	2.7	3.5	3.7	4.1
MTW	R ≠ 0	2.8	3.4	3.7	4.0	3.9	3.6	3.8	4.2	2.6	3.3	3.6	3.9	3.9	3.6	3.8	4.2
MTW	R = 0	2.3	2.9	3.2	3.6	2.4	3.1	3.4	3.8	2.3	2.9	3.2	3.6	2.4	3.1	3.4	3.8
MW		5.0	5.6	5.9	6.2	5.1	5.8	6.0	6.5	5.0	5.6	5.9	6.2	5.1	5.8	6.0	6.5
OR		1.8	2.4	2.7	3.0	1.8	2.5	2.7	3.2	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.8
PACK	16	12.0 +0.6N	12.0 +0.6N	12.6 +0.6N	12.6 +0.6N	12.0 +0.6N	12.0 +0.6N	12.8 +0.6N	12.8 +0.6N	12.0 +0.6N	12.0 +0.6N	12.6 +0.6N	12.6 +0.6N	12.0 +0.6N	12.0 +0.6N	12.8 +0.6N	12.8 +0.6N
PLM	15	10.0 +1.0N	10.0 +1.0N	10.8 +1.0N	10.8 +1.0N	10.5 +1.1N	10.5 +1.1N	11.1 +1.1N	11.1 +1.1N	9.5 +1.0N	9.5 +1.0N	10.0 +1.0N	10.0 +1.0N	10.2 +1.0N	10.2 +1.0N	10.7 +1.1N	10.7 +1.1N
PLW		10.8	10.8	11.4	11.4	11.2	11.2	11.8	11.8	10.2	10.2	10.8	10.8	10.8	10.8	11.4	11.4
PSM	15	8.7 +1.0N	8.7 +1.0N	9.4 +1.0N	9.4 +1.0N	9.0 +1.0N	9.0 +1.0N	9.7 +1.0N	9.7 +1.0N	8.3 +0.8N	8.3 +0.8N	9.0 +0.8N	9.0 +0.8N	8.6 +1.0N	8.6 +1.0N	9.6 +1.0N	9.6 +1.0N
PSW		9.8	9.8	10.5	10.5	10.2	10.2	10.9	10.9	9.3	9.3	9.8	9.8	9.8	9.9	10.5	10.5
RD	internal	2.5	2.5	3.1	3.1	2.5	2.5	3.1	3.1	2.5	2.5	3.1	3.1	2.5	2.5	3.1	3.1
RD	external 17	2.8 +0.4N	2.8 +0.4N	3.4 +0.4N	3.4 +0.4N	2.8 +0.4N	2.8 +0.4N	3.4 +0.4N	3.4 +0.4N	2.8 +0.4N	2.8 +0.4N	3.4 +0.4N	3.4 +0.4N	2.8 +0.4N	2.8 +0.4N	3.4 +0.4N	3.4 +0.4N
S left	18	2.1 +0.1N	2.1 +0.1N	2.7 +0.1N	2.7 +0.1N	2.2 +0.1N	2.2 +0.1N	2.8 +0.1N	2.8 +0.1N	2.1 +0.1N	2.1 +0.1N	2.7 +0.1N	2.7 +0.1N	2.1 +0.1N	2.1 +0.1N	2.7 +0.1N	2.7 +0.1N
S right	18	2.1 +0.2N	2.1 +0.2N	2.8 +0.2N	2.8 +0.2N	2.2 +0.2N	2.2 +0.2N	2.9 +0.2N	2.9 +0.2N	2.1 +0.2N	2.1 +0.2N	2.8 +0.2N	2.8 +0.2N	2.2 +0.2N	2.2 +0.2N	2.9 +0.2N	2.9 +0.2N
SD		2.9	3.6	3.9	4.2	2.9	3.7	3.9	4.3	2.4	3.0	3.3	3.6	2.5	3.2	3.4	3.8
SF left	single 19	2.6 +0.2N	2.6 +0.2N	3.2 +0.2N	3.2 +0.2N	2.7 +0.2N	2.7 +0.2N	3.3 +0.2N	3.3 +0.2N	2.6 +0.2N	2.6 +0.2N	3.2 +0.2N	3.2 +0.2N	2.7 +0.2N	2.7 +0.2N	3.3 +0.2N	3.3 +0.2N
SF right	single 19	2.4 +0.6N	2.4 +0.6N	3.0 +0.6N	3.0 +0.6N	2.6 +0.6N	2.6 +0.6N	3.2 +0.6N	3.2 +0.6N	2.4 +0.6N	2.4 +0.6N	3.0 +0.6N	3.0 +0.6N	2.6 +0.6N	2.6 +0.6N	3.2 +0.6N	3.2 +0.6N
SF left	double 19	4.0 +0.2N	4.0 +0.2N	4.6 +0.2N	4.6 +0.2N	4.1 +0.2N	4.1 +0.2N	4.7 +0.2N	4.7 +0.2N	4.0 +0.2N	4.0 +0.2N	4.6 +0.2N	4.6 +0.2N	4.1 +0.2N	4.1 +0.2N	4.7 +0.2N	4.7 +0.2N
SF right	double 19	3.8 +0.6N	3.8 +0.6N	4.4 +0.6N	4.4 +0.6N	3.9 +0.6N	3.9 +0.6N	4.6 +0.6N	4.6 +0.6N	3.8 +0.6N	3.8 +0.6N	4.4 +0.6N	4.4 +0.6N	3.9 +0.6N	3.9 +0.6N	4.6 +0.6N	4.6 +0.6N
SH		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.4	2.0	2.3	2.6	1.5	2.2	2.4	2.8
SIO	R = even, ≠0	10.6	10.6	11.2	11.2	10.6	10.6	11.2	11.2	10.3	10.3	10.9	10.9	10.4	10.4	11.0	11.0
SIO	R = odd	9.5	9.5	10.1	10.1	9.5	9.5	10.1	10.1	9.5	9.5	10.1	10.1	9.5	9.5	10.1	10.1
SIO	R = 0	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7
STB		3.0	3.0	3.6	3.6	3.1	3.1	3.7	3.7	2.9	2.9	3.5	3.5	3.0	3.1	3.6	3.7
STCF		3.0	3.0	3.6	3.6	3.1	3.1	3.7	3.7	2.9	2.9	3.5	3.5	3.0	3.1	3.6	3.7
STD		3.6	3.6	4.2	4.2	3.7	3.7	4.3	4.3	3.2	3.2	3.3	3.7	3.5	3.5	3.3	3.9

Table D-1. Basic Instruction Timing (cont.)

Mnemonics	Notes	No Memory Overlap								Maximum Memory Overlap							
		No Map				Map				No Map				Map			
		Direct		Indirect		Direct		Indirect		Direct		Indirect		Direct		Indirect	
		No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index	No Index	Index
STH		3.0	3.0	3.6	3.6	3.1	3.1	3.7	3.7	2.8	2.8	3.5	3.9	3.0	3.0	3.6	4.0
STM	15	2.1 +1.0N	2.1 +1.0N	2.8 +1.0N	2.8 +1.0N	2.2 +1.0N	2.2 +1.0N	2.8 +1.0N	2.8 +1.0N	2.1 +0.8N	2.1 +0.8N	2.8 +0.8N	2.8 +0.8N	2.2 +0.9N	2.2 +0.9N	2.2 +0.9N	2.2 +0.9N
STS		3.7	4.3	4.7	5.0	3.8	4.5	4.8	5.2	3.5	4.0	4.4	4.6	3.6	4.3	4.5	4.9
STW		2.6	2.6	3.2	3.2	2.7	2.7	3.3	3.3	2.3	2.3	2.9	2.9	2.6	2.7	3.2	3.3
SW		2.0	2.6	2.9	3.2	2.0	2.7	2.9	3.3	1.4	2.0	2.3	2.6	1.5	2.2	2.4	3.0
TBS	2	3.0 +4.2N	--	--	--	3.2 +4.4N	--	--	--	3.0 +4.2N	--	--	--	3.2 +4.4N	--	--	--
TDV	R even, ≠0	9.7	9.7	10.3	10.3	9.7	9.7	10.3	10.3	9.4	9.4	10.0	10.0	9.5	9.5	10.1	10.1
TDV	R odd	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9
TDV	R = 0	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7
TIO	R = even, ≠0	9.7	9.7	10.3	10.3	9.7	9.7	10.3	10.3	9.4	9.4	10.0	10.0	9.5	9.5	10.1	10.1
TIO	R odd	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9	8.3	8.3	8.9	8.9
TIO	R = 0	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7	7.1	7.1	7.7	7.7
TTBS	2	3.2 +4.3N	--	--	--	3.2 +4.6N	--	--	--	3.2 +4.3N	--	--	--	3.2 +4.6N	--	--	--
UNPK	20, 21	11.6 +1.3N	11.6 +1.3N	12.1 +1.4N	12.1 +1.4N	11.9 +1.3N	11.9 +1.3N	12.3 +1.3N	12.3 +1.3N	11.4 +1.3N	11.4 +1.3N	12.0 +1.3N	12.0 +1.3N	11.8 +1.3N	11.8 +1.3N	12.2 +1.3N	12.2 +1.3N
WAIT	21	1.9	1.9	2.6	2.6	1.9	1.9	2.7	2.7	1.8	1.8	2.4	2.4	1.9	1.9	2.5	2.5
WD	internal	2.5	2.5	3.1	3.1	2.5	2.5	3.1	3.1	2.5	2.5	3.1	3.1	2.5	2.5	3.1	3.1
WD	external 17	2.8 +0.4N	2.8 +0.4N	3.4 +0.4N	3.4 +0.4N	2.8 +0.4N	2.8 +0.4N	3.4 +0.4N	3.4 +0.4N	2.8 +0.4N	2.8 +0.4N	3.4 +0.4N	3.4 +0.4N	2.8 +0.4N	2.8 +0.4N	3.4 +0.4N	3.4 +0.4N
XPSD	I_{10} 0	6.5	6.5	7.1	7.1	6.5	6.5	7.1	7.1	6.1	6.1	6.6	6.6	6.1	6.1	6.7	6.7
XPSD	I_{10} 1	6.5	6.5	7.1	7.1	6.7	6.7	7.3	7.3	6.1	6.1	6.6	6.6	6.5	6.5	7.1	7.1
XW		3.0	3.6	3.9	4.2	3.1	3.8	4.0	4.4	2.6	3.3	3.6	3.9	2.9	3.6	3.8	4.2

Notes:
1. Add 0.6 if analyzed instruction is indirect. Subtract 0.3 if it is LCFI, AI, LI, CBS, MBS, or EBS.
2. N = number of destination bytes processed.
3. N = number of 1's in the word converted.
4. D = number of digits (including the sign) in the effective decimal operand.
5. K = (D + 6) (16 - Q); D = same as note 4; Q = number of leading zeros in the quotient.
6. D = same as note 4; N = number of nonzero decimal digits in the decimal accumulator.
7. D = number of digits (including the sign) to be stored.
8. N = number of bytes in the editing pattern.
9. Add execution time for subject instruction.
10. No pre-alignment or post-normalization required.
11. Un-normalized operands.
12. One hexadecimal pre-alignment and one hexadecimal post-normalization.
13. Nonzero, normalized operands.
14. Minimum time is also typical time.
15. N = number of words moved.
16. N = number of bytes in zoned number in memory.
17. N = integer (0, 1, 2, . . .), dependent on delay in external device.
18. N = number of bit positions shifted.
19. N = number of hexadecimal positions shifted.
20. N = number of bytes to be stored in memory.
21. Minimum time.

Table D-2. Additional Instruction Timing
(Add to times in Table D-1)

Mnemonic	Register-to-register Operations					Register pointer selects register block X'4' - X'1F'				
	Notes	Direct		Indirect		Notes	Direct		Indirect	
		No Index	Index	No Index	Index		No Index	Index	No Index	Index
AD		2.2	1.4	1.2	1.2	22	0.5	0.3	0.9	0.6
AD		--	--	--	--	23	0.5	0.4	1.0	0.7
AH		1.2	0.5	1.2	1.3		0.4	0.3	0.8	0.6
AI		--	--	--	--		0.1	--	--	--
AIO		0	0	1.5	1.5		0.6	0.6	0.9	0.9
AND		1.2	0.5	1.2	1.3		0.4	0.3	0.8	0.6
ANLZ		1.4	0.6	1.3	1.3		0.9	0.7	1.6	1.3
AW		1.2	0.5	1.2	1.3		0.4	0.3	0.8	0.6
AWM		2.2	1.6	1.3	1.3		0.4	0.3	0.8	0.6
BAL		0.7	0.7	1.4	1.4		0.4	0.4	0.7	0.7
BCR	branch	1.3	0.7	1.3	1.4		0.3	0.3	0.7	0.6
BCR	no branch	2.1	1.9	1.3	1.3		--	--	--	--
BCS	branch	1.3	0.7	1.3	1.4		0.3	0.3	0.7	0.6
BCS	no branch	2.5	1.9	1.3	1.3		--	--	--	--
BDR	branch	1.4	0.9	1.4	1.4		0.3	0.3	0.7	0.6
BDR	no branch	2.4	2.1	1.2	1.3		--	--	--	--
BIR	branch	1.4	0.9	1.4	1.4		0.3	0.3	0.7	0.6
BIR	no branch	2.4	2.1	1.2	1.3		--	--	--	--
CAL 1, 2, 3, 4		0	0	1.4	1.4		0.4	0.4	0.7	0.7
CB		1.3	0.6	1.3	1.3		0.4	0.3	0.8	0.6
CBS	24	0.7N	--	--	--		0.6	--	--	--
CD		2.2	1.4	1.2	1.2		0.4	0.3	0.8	0.6
CH		1.3	0.6	1.3	1.3		0.4	0.3	0.8	0.6
CI		--	--	--	--		0.4	--	--	--
CLM		1.5	1.2	1.2	1.2		0.4	0.3	0.8	0.6
CLR		1.3	0.7	1.4	1.4		0.4	0.3	0.8	0.6
CS		1.4	0.7	1.3	1.3		0.4	0.3	0.8	0.6
CVA	30	--	--	1.4	1.4		0.4	0.4	0.7	0.7
CVS	30	--	--	1.4	1.4		0.4	0.4	0.7	0.7
CW		1.3	0.6	1.3	1.3		0.4	0.3	0.8	0.6
DA		0.1D	0.1D	1.5	1.5		0.4	0.4	0.7	0.7
DC		0.1D	0.1D	1.5	1.5		0.4	0.4	0.7	0.7
DD		3.5	3.5	1.5	1.5		0.4	0.4	0.7	0.7
DH		1.5	0.7	1.4	1.4		0.4	0.3	0.8	0.6
DL		0.1D	0.1D	1.5	1.5		0.4	0.4	0.7	0.7
DM		3.5	3.5	1.5	1.5		0.4	0.4	0.7	0.7
DS		0.1D	0.1D	1.5	1.5		0.4	0.4	0.7	0.7
DSA		0	0	1.4	1.4		0.4	0.4	0.7	0.7
DST		0.3D	0.3D	1.5	1.5		0.4	0.4	0.7	0.7
DW		1.5	0.8	1.4	1.4		0.4	0.3	0.8	0.6
EBS	25	0.4N	--	--	--		0.3	--	--	--
EOR		1.4	0.7	1.4	1.5		0.4	0.3	0.8	0.6
EXU	26	1.5	0.7	1.5	1.5	26	0.4	0.3	0.8	0.6

Mnemonic	Register-to-register Operations					Register pointer selects register block X'4' - X1F'				
	Notes	Direct		Indirect		Notes	Direct		Indirect	
		No Index	Index	No Index	Index		No Index	Index	No Index	Index
FAL		2.3	1.6	1.5	1.5		0.4	0.3	0.8	0.6
FAS		1.5	0.8	1.5	1.5		0.4	0.3	0.8	0.6
FDL		2.3	1.6	1.5	1.5		0.4	0.3	0.8	0.6
FDS		1.5	0.8	1.5	1.5		0.4	0.3	0.8	0.6
FML		2.3	1.6	1.5	1.5		0.4	0.3	0.8	0.6
FMS		1.5	0.8	1.5	1.5		0.4	0.3	0.8	0.6
FSL		2.3	1.6	1.5	1.5		0.4	0.3	0.8	0.6
FSS		1.5	0.8	1.5	1.5		0.4	0.3	0.8	0.6
HIO		0	0	1.5	1.5		0.6	0.6	0.9	0.9
INT		1.4	0.7	1.4	1.4		0.4	0.3	0.8	0.6
LAD		2.3	1.5	1.3	1.3		0.4	0.3	0.8	0.6
LAH		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
LAW		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
LB		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
LCD		2.2	1.4	1.2	1.2		0.4	0.3	0.8	0.6
LCF		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
LCFI		--	--	--	--		0.1	--	--	--
LCH		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
LCW		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
LD		2.2	1.4	1.2	1.2		0.4	0.3	0.8	0.6
LH		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
LI		--	--	--	--		0.1	--	--	--
LM		0.8N	0.8N	1.3	1.3		0.4	0.4	0.7	0.7
LPSD		1.8	1.8	1.5	1.5		0.4	0.3	0.8	0.6
LRP		1.5	0.7	1.5	1.5		0.4	0.4	0.7	0.7
LS		1.5	0.8	1.5	1.5		0.5	0.4	1.0	0.7
LW		1.4	0.7	1.5	1.5		0.4	0.3	0.8	0.7
MBS	27	0.2N	--	--	--		0.6	--	--	--
MBS	28	0.3N	--	--	--		--	--	--	--
MH		1.5	0.8	1.5	1.5		0.4	0.3	0.8	0.6
MI		--	--	--	--		0.4	--	--	--
MMC		0.8N	--	--	--		0.6	0.6	0.9	0.9
MSP		3.5	3.5	1.5	1.5		0.4	0.4	0.7	0.7
MTB	R≠0	2.1	1.4	1.5	1.5		0.4	0.3	0.8	0.6
MTB	R=0	1.5	0.8	1.5	1.5		--	--	--	--
MTH	R≠0	2.1	1.4	1.5	1.5		0.4	0.3	0.8	0.6
MTH	R=0	1.5	0.8	1.5	1.5		--	--	--	--
MTW	R≠0	2.4	1.7	1.5	1.5		0.4	0.3	0.8	0.6
MTW	R=0	1.5	0.8	1.5	1.5		--	--	--	--
MW		1.5	0.8	1.5	1.5		0.4	0.3	0.8	0.6
OR		1.4	0.7	1.5	1.5		0.4	0.3	0.8	0.6
PACK		0.2N	0.2N	1.5	1.5		0.4	0.4	0.7	0.7
PLM		3.2	3.2	1.1	1.1		0.4	0.4	0.7	0.7

Table D-2. Additional Instruction Timing (cont.)
(Add to times in Table D-1)

Mnemonic	Register-to-register Operations					Register pointer selects register block X'4' - X'1F'				
	Notes	Direct		Indirect		Notes	Direct		Indirect	
		No Index	Index	No Index	Index		No Index	Index	No Index	Index
PLW		3.5	3.5	1.5	1.5		0.4	0.4	0.7	0.7
PSM		3.1	3.1	1.4	1.4		0.4	0.4	0.7	0.7
PSW		3.1	3.2	1.1	1.1		0.4	0.4	0.7	0.7
RD		0	0	1.5	1.5		0.4	0.4	0.7	0.7
S		0	0	1.5	1.5		0.4	0.4	0.7	0.7
SD		2.2	1.4	1.2	1.2		0.4	0.3	0.8	0.6
SF		0	0	1.5	1.5		0.4	0.4	0.7	0.7
SH		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
SIO		0	0	1.5	1.5		0.6	0.6	0.9	0.9
STB		0.5	0.6	1.4	1.5		0.3	0.3	0.6	0.6
STCF		0.5	0.6	1.4	1.5		0.3	0.3	0.6	0.6
STD		1.7	1.7	0.5	1.1		0.3	0.3	0.6	0.6
STH		0.5	0.5	1.4	1.4		0.3	0.3	0.6	0.6

Mnemonic	Register-to-register Operations					Register pointer selects register block X'4' - X1F'				
	Notes	Direct		Indirect		Notes	Direct		Indirect	
		No Index	Index	No Index	Index		No Index	Index	No Index	Index
STM		0.8N	0.8N	0.9	0.9		0.4	0.4	0.7	0.7
STS		2.3	1.5	1.3	1.2		0.6	0.4	1.0	0.7
STW		0.8	0.9	1.4	1.5		0.3	0.3	0.6	0.6
SW		1.2	0.5	1.3	1.4		0.4	0.3	0.8	0.6
TBS	29	1.8N	--	--	--		0.6	--	--	--
TDV		0	0	1.5	1.5		0.6	0.6	0.9	0.9
TIO		0	0	1.5	1.5		0.6	0.6	0.9	0.9
TTBS	29	0.8N	--	--	--		0.6 +0.2N	--	--	--
UNPK		0.5N	0.5N	1.1	1.1		0.4	0.4	0.7	0.7
WAIT		0	0	1.3	1.3		0.4	0.4	0.7	0.7
WD		0	0	1.5	1.5		0.4	0.4	0.7	0.7
XPSD		3.5	3.5	1.3	1.3		0.4	0.4	0.7	0.7
XW		2.2	1.5	1.3	1.3		0.4	0.3	0.8	0.6

Notes:
22. No memory overlap.
23. Maximum memory overlap.
24. One byte string is in registers.
25. Decimal number is in registers.
26. Add factor for object instruction.
27. Word mode – one byte string in registers.
28. Byte mode – one byte string in registers.
29. Byte string to be translated in registers.
30. CVA and CVS instructions require a 32-word table and should not be used in register-to-register operations. The indirect word, however, may be located in a register.

INDEX

A

B

C

D

E

F

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

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 59153 LOS ANGELES,CA 90045

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
5250 W. CENTURY BOULEVARD
LOS ANGELES, CA 90045

ATTN: PROGRAMMING PUBLICATIONS

CUT ALONG LINE

FOLD ALONG LINE

FOLD ALONG LINE

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 155 Gordon Baker Road, Willowdale, Ontario M2H 3N7
In the U.K.: Great West Road, Brentford, Middlesex TW8 9DH
In Australia: 124 Walker Street, North Sydney, N.S.W. 2060
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

35325, 3.5C882, Printed in U.S.A.

XG46-00