0830005-000

REFERENCE MANUAL

SERIES 100 COMPUTER SYSTEMS

S115, S125, AND S135

Original Issue
March, 1978

HARRIS

COMMUNICATIONS AND
INFORMATION HANDLING

HARRIS CORPORATION Computer Systems Division
1200 Gateway Drive, Fort Lauderdale, Florida 33309 305/974-1700

LIST OF EFFECTIVE PAGES

TOTAL NUMBER OF PAGES IN THIS PUBLICATION IS: 159
CONSISTING OF THE FOLLOWING:

Page No.	Change No.	Page No.	Change No.	Page No.	Change No.
Title	Original				
A, B	Original				
i thru vi	Original				
1-1 thru 1-10	Original				
2-1 thru 2-20	Original				
3-1 thru 3-4	Original				
4-1 thru 4-20	Original				
5-1 thru 5-6	Original				
6-1 thru 6-4	Original				
7-1 thru 7-72	Original				
A-1 thru A-8	Original				
B-1 thru B-6	Original				

Insert Latest Revision Pages. Destroy Superseded Pages.

PROPRIETARY DATA

CONTENTS

CONTENTS (CONT'D.)

CONTENTS (CONT'D.)

CONTENTS (CONT'D.)

CONTENTS (CONT'D.)

ILLUSTRATIONS

ILLUSTRATIONS (CONT'D.)

TABLES

SECTION I
INTRODUCTION

SCOPE OF MANUAL

This manual contains reference material for the Series 100/S115, S125, and S135 Computer Systems designed and manufactured by Harris Corporation, Computer Systems Division, Fort Lauderdale, Florida. Included are descriptions of the overall computer organization, central processing unit (CPU), memory configurations, priority interrupt system, input/output (I/O) channels, and instruction set. Various hardware features and options are also described; application and programming examples are provided where appropriate.

The material in this manual is oriented toward the user/programmer with a knowledge of computer fundamentals and terminology.

SERIES 100/SYSTEMS 115, 125, and 135

The S115, S125 and S135 computer systems are members of the Harris Series 100 family. This family is comprised of high-performance, disc-based, virtual memory computer systems for performing concurrent time-sharing, multi-batch, remote job entry and real-time processing. The S115, S125, and S135 are building-block systems; each may be expanded to support a variety of applications and performance levels. Upgrades between systems are also available. Series 100 systems provide cost-effective solutions for distributed data processing, transaction oriented processing, and communications applications. Data Base Management and inquiry software is available for fast, efficient file maintenance and information retrieval. These multi-use systems are ideal for scientific, commercial and real-time applications — since they provide true multi-programming and multi-lingual capabilities.

BASIC COMPUTER ORGANIZATION

Basic Operation

Figure 1-1 illustrates the functional relationship between major units of a typical system. The major functional units include the central processing unit (CPU), memory, priority interrupt system, input/output (I/O) channels, programmer's control panel, and the optional Scientific Arithmetic Unit (SAU).

The computer has a 24-bit fixed word length, a multi-access bus structure, and an integral memory system. Operations are performed on, and from, 24-bit data and instruction words. In addition, the computer is capable of selective byte manipulation and performs Boolean functions on single, selected bits. Two's complement arithmetic is performed on parallel, binary, fixed-point operands. Concurrent floating-point arithmetic is performed by the SAU.

Data or instruction words may be retrieved from or stored in memory, retained in one of the CPU registers, or received from and transmitted to peripheral devices via the I/O channels. Prior to execution, instructions must be loaded into, and subsequently retrieved from, physical memory. Main memory is accessed on a double word-boundary. This arrangement permits an instruction prefetch which reduces the effective access time of the memory system. In addition, the CPU employs an asynchronous cycle that automatically adjusts to the timing of the addressed memory module. If, for example, memory contention occurs, the CPU waits at a predetermined point until memory becomes available.

Memory may be accessed at the word, double-word, byte, and bit levels by the standard instruction set. If virtual memory is disabled by the control panel key switch, memory is divided into 32K word sections. Up to 32K words per section may be directly addressed and up to 256K words can be accessed by indirect and indexed address references. Executable code is restricted to 65,536 (64K) words at any given time.

When virtual memory is enabled, two addressing modes are employed, User and Monitor. Addresses generated in the User Mode (called logical addresses) are translated into physical memory addresses by the virtual memory hardware. The logical address is translated to the physical address by selecting the appropriate 1024 (1K) word physical "page" and the offset within that page. The division of main memory into physical pages allows a program to be located in non-contiguous areas of memory, and to be transferred (in page increments) between memory and an external mass storage device under system control. When the virtual memory hardware detects a reference to a page which is not currently resident in main memory, a page fault occurs. This supports a demand-paging technique which allows portions of a program to be absent from memory while the program is running. The paging circuits are disabled in the Monitor Mode, thus addresses generated in the Monitor Mode are used directly as physical main memory addresses.

Central Processing Unit (CPU)

Included in the CPU are several general and special-purpose registers, an arithmetic section, timing and control logic, memory interface circuits, and I/O channel interface circuits. Special paging registers and control logic are provided for virtual memory operation. When the system includes an SAU, the CPU includes special circuits for CPU-SAU interface and communications.

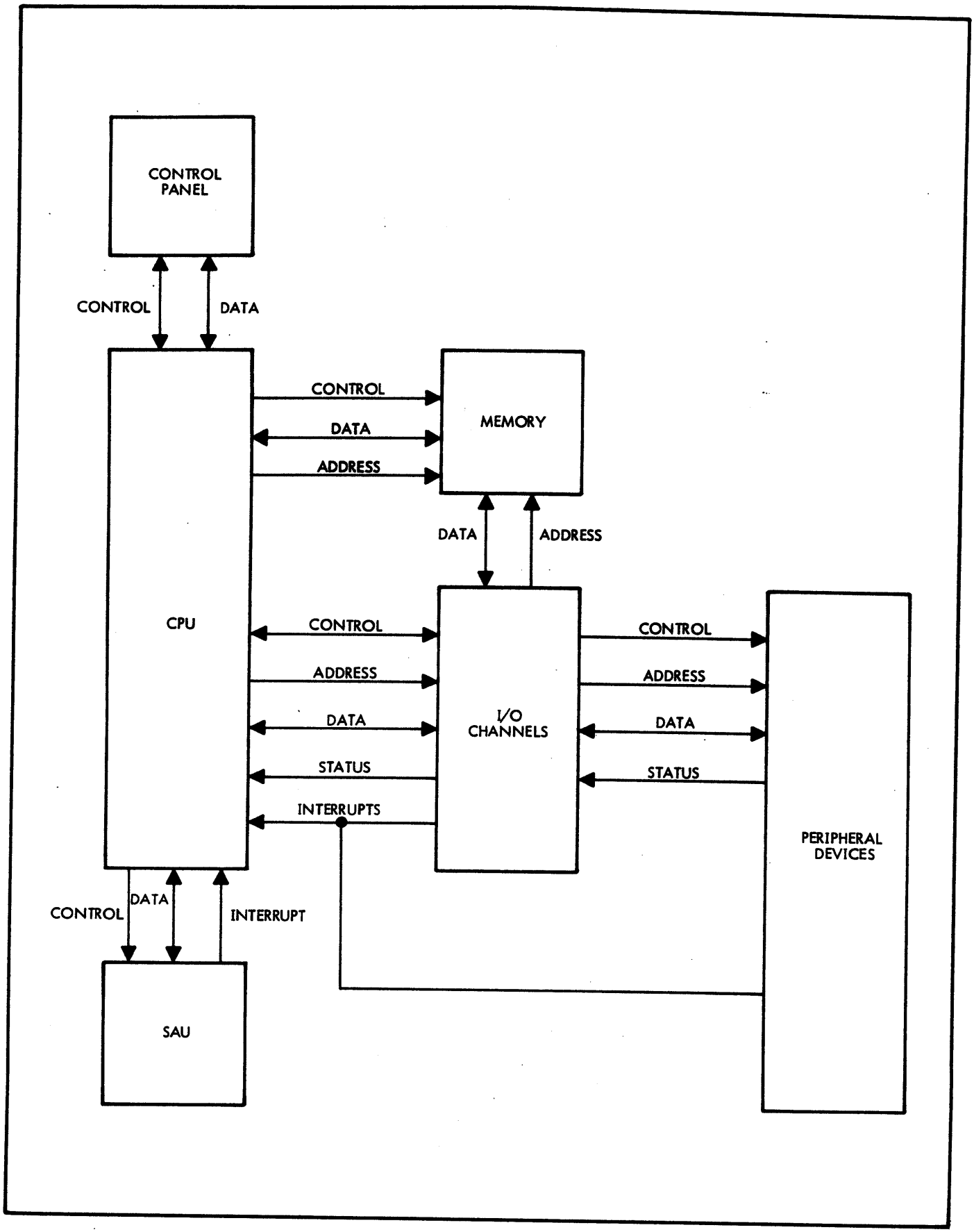


Figure 1-1. Major Functional Units

Five general-purpose registers are included in a basic CPU. These registers are employed in a variety of logical, arithmetic, and manipulative operations such as register-memory, memory-register, and register-register instructions. Three of the general-purpose registers can be used for indexing in memory addressing functions. One register serves as the I/O communication register during single-word input/output operations. A double-word register is formed by combining two 24-bit registers, and a byte register is created by using the eight least-significant bits of one general-purpose register. With the Interval Timer included in the CPU, the Timer (T) Register becomes a sixth general-purpose register in the Monitor Mode of operation. In the User Mode, the T Register cannot be accessed.

Among the special-purpose registers are those associated with integral CPU functions such as addressing, instruction decoding, and temporary storage during data manipulation. Additional special-purpose registers are those supplied with the bit (Boolean) processor, Interval Timer (T Register, timing applications), Program Restrict and Instruction Trap option, and the Program Halt and Address Trap.

The arithmetic section consists primarily of a 24-bit arithmetic logic unit (ALU) and several buses to permit data manipulation between the various registers and the ALU. Arithmetic functions performed include addition, subtraction, multiplication, division, and square root computation. In addition, the ALU output is employed in computing addresses during memory reference operations.

Instruction execution sequences are established and directed by the timing and control logic associated with the CPU. This logic includes a crystal-controlled clock generator that provides precise timing for all instruction functions. Instruction words are retrieved from memory and retained in an instruction register for the duration of the operation. The control logic decodes these instruction words and provides the internal commands necessary for execution. In the User Mode of operation, the paging control logic operates in conjunction with the basic CPU timing to implement address translation and demand paging techniques.

CPU memory interface circuits consist of address and data-handling buses and registers, and parity generation/checking or error checking and correction code logic. Memory interface circuits include a 48-bit data register that retains both the read and write data, an 18-bit address register to define the physical memory location to be accessed, data multiplexing logic to control read and write data handling, and address multiplexing and control logic for selecting the proper memory segment and a location within that segment. Data to be written (stored) in memory is applied via the system data bus. Data read (retrieved) from memory is applied to the CPU via the system data bus. Address inputs are applied to the memory interface via the system address bus. The address source may be the CPU Memory Address Register, Program Address Register (Program Counter), the ALU or Operand Register in the arithmetic section, one of the block transfer channels (DMACP, UBC, XBC, IBC), or, in the User Mode of operation, the paging logic addressing circuits.

Communications between the CPU and the I/O channels are conducted via the channel interface logic in the CPU. This logic makes use of the system buses and one of the general-purpose registers in order to implement data and address flow between the CPU and I/O channels. Although an I/O channel conducts channel-unit communications independently and asynchronously, input/output operations such as channel-unit selection and activation, function commands, and status testing are initiated under program control.

When the Scientific Arithmetic Unit (SAU) is employed, CPU-SAU communications are conducted via interface circuits both in the CPU and SAU. Single or double-precision transfers may be performed; with double-precision transfers requiring two sequences. All SAU instructions and data transfers are initiated on CPU timing. All concurrent floating-point arithmetic operations are performed on SAU timing.

Memory Units

Storage of information, both instruction and data words, is a function of the memory modules. Both semiconductor and magnetic core memory modules may be used in the memory configuration. Memory modules may all be of the same type or may be mixed. Memory modules can be mixed in the 125C and 135C systems. The 115, 125, and 135 systems do not support core. Semiconductor memory is available in increments of 16K words, while increments of 32K words are available with magnetic core modules. Total main memory capacity is 256K words. Refer to Section III for additional details concerning the memory system.

Input/Output Operation

Input/Output (I/O) operations consist of data, address, command, or status transfers between selected peripheral devices and the CPU or memory. All such operations are initiated under program control and are conducted, asynchronously, by an I/O channel. Various types of I/O channel boards may be installed in a system. All channels in the system can be active simultaneously, and each channel may communicate with a maximum of 16 devices (limited by transfer rates).

An I/O operation is initiated by selecting and activating a channel, and one of its assigned peripheral devices, through the execution of a computer input/output instruction. (The instruction set, includes seven input/output instructions.) A specific I/O operation may involve preparing a peripheral device for a subsequent communication, determining the operational status of a device, or initiating a data transfer. Once activated, the I/O channel provides complete functional control over the operation.

Data may be transferred on a single word basis (i.e., one data word per instruction) or automatically, in blocks of n words per operation. Block data transfers are performed by the Direct Memory Access Communication Processor (DMACP), Universal Block Channel (UBC), External Block Channel (XBC), or Integral Block Channel (IBC). Each available type of I/O channel permits data transfers to (input) and from (output) the computer.

I/O operations may also be conducted on an interrupt basis through the use of interrupt logic in the channel(s). The channel interrupt system can be placed under program control and selectively enabled or disabled by an input/output instruction. Peripheral device functions may be connected directly to the computer priority interrupt system, bypassing the channel interrupt logic.

Priority Interrupt System

Each system contains an interrupt system that allows additional program control of input/output and internal CPU operations, and immediate recognition of special external conditions on the basis of priority. Receipt and recognition of an interrupt trigger permits normal program flow to be diverted to a subroutine that services the interrupt and returns the program to its normal sequence at the point where the interruption occurred.

Two interrupt groups, 0 and 1, are available. Group 0 is reserved for internal CPU functions and is comprised of eight executive trap interrupt levels. All executive trap levels are associated with specific functions. Group 1 is reserved for external interrupts; the group may have up to 24 levels. A basic system is supplied with 16 external interrupt levels. Eight additional external interrupts are available.

Complete details pertaining to the priority interrupt system are contained in Section V.

Programmer's Control Panel

The Programmer's Control Panel contains the facilities for manually starting and halting operations, entering data into memory and the various registers, and selecting registers for display and/or entry. Indicators on the panel provide display for the contents of registers and memory, system status, and other important functions. Complete operating instructions for the control panel are contained in publication number 0840003.

STANDARD AND OPTIONAL FEATURES

Systems 115, 125, and 135 contain various hardware features. Many options are also available to enhance system performance. A brief description of standard features and options are provided in the following paragraphs. Unless otherwise indicated, additional details pertaining to system features and options are contained in Section II. Table 1-1 lists the standard hardware items provided with each system, while Table 1-2 provides a summary of the optional items available.

120 Hertz Clock

Continuously generated interrupt triggers are placed under software control by enabling or disabling the associated external interrupt level. By this method, the 120 Hertz Clock may be used for various timing operations. The clock continuously transmits 120 interrupt trigger pulses per second for 60 Hertz power, and 100 interrupt trigger pulses per second for 50 Hertz power.

Power Fail Shutdown and Restart

This feature provides a means for protecting operating programs in the event of a power failure and for restoring the original conditions when power levels return to normal. Two executive trap interrupt levels, one for power down and the other for power up, are supplied.

Firmware Bootstraps

Automatic program loading from a selected peripheral device is provided by the Firmware Bootstraps feature. Through the use of console switches, the appropriate bootstrap program is loaded into memory. Once loaded, the bootstrap program will automatically load a minimum of one record from the appropriate device. Programs stored in a PROM provide for loading from disc, paper tape, magnetic tape, punched cards, magnetic tape cassettes (paper tape emulation), or floppy disc.

Bit Processor

Capability is provided by the Bit Processor for selectively changing, testing, or performing logical operations on a single bit in memory.

Stall Alarm

Certain operations in the instruction set and other internal conditions prohibit the recognition of external interrupts. A series of these instructions or conditions could, therefore, produce a situation where external interrupts are, in effect, "locked out". The Stall Alarm monitors all instructions and conditions in this interrupt-prohibiting category. If a series of these instructions or conditions have not been completed before the elapse of a predetermined time period, an executive trap interrupt is generated. The subsequent interrupt-processing routine may then examine the situation and take any necessary corrective action. The Stall Alarm includes the appropriate control logic and is furnished with the associated executive trap interrupt.

Table 1-1. Standard Features

Description	S115	S125	S135
Central Processor with hardware multiply/divide/square root, power supplies, and CPU cabinet	•	•	•
Semiconductor (MOS) Memory with error correction			
48K words	•	•	
128K words			•
Virtual Memory			
256K words	•		
1024K words		•	
4096K words			•
Programmer's Control Panel	•	•	•
16 Priority Interrupt Levels	•	•	•
120 Hertz Clock	•	•	•
*Power Fail Shutdown and Restart	•	•	•
Firmware Bootstraps	•	•	•
Bit Processor	•	•	•
Stall Alarm	•	•	•
Executive Traps	•	•	•
Interval Timer	•	•	•
Program Halt and Address Trap	•	•	•
Programmed Input Output Channel (PIOC)	•	•	•
Universal Block Channel (UBC)	•	•	•
Communications Multiplexer, less ports	•		
Direct Memory Access Communications Processor (DMACP) (includes 4 customer specified asynchronous ports or 1 synchronous port)		•	•
Interactive CRT Operator Console and controller	•	•	•
10.8 MByte Cartridge Disc and Controller	•	•	
40 MByte Storage Module Drive and Controller			•
9 Track, 800 BPI, 45 IPS Magnetic Tape Unit with cabinet (on UBC)	•	•	•

*MOS Data Save option is required for MOS Memory.

Table 1-2. Optional Features

Description	S115	S125	S135
16K word Semiconductor (MOS) Memory increment			
Up to 64K words	•		
Up to 208K words		•	
Up to 256K words			•
*32K word Core Memory increment		•	•
Scientific Arithmetic Unit		•	•
Program Restrict and Instruction Trap	•	•	•
Real Time Clock	•	•	•
Run Time Meter	•	•	•
Programmed Input Output Channels (PIOCs)	•	•	•
Universal Block Channels (UBC)	•	•	•
External Block Channels (XBC)	•	•	•
Integral Block Channels (IBC)	•	•	•
**Direct Memory Access Communications Processor (DMACP) channels	•	•	•
8 Priority Interrupt Levels	•	•	•
MOS Data Save	•	•	•
I/O Expansion Chassis		•	
Computer Link	•	•	•
Multi-CPU Channel Adapter	•	•	•

*S125C and S135C systems only

** Requires special power

Interval Timer

The programmable Interval Timer functions as an internal CPU timer that provides a method for regulating operating program segments and recording other intervals. Depending on the instruction used for its activation, the Interval Timer clocks either CPU time or real time. In addition to its timing applications, the Interval Timer provides the user with an additional 24-bit general purpose register that may be accessed through the standard instruction set when in the Monitor Mode of operation. The T Register may not be modified when in the User Mode.

Program Halt and Address Trap

This feature provides for a program halt or an executive trap interrupt to occur at a specified address and under certain conditions. The address trap is used as an on-line debugging aid for use in applications such as breakpoint tracing. An address may be defined under program control so that when the address is referenced, an interrupt will be generated at the assigned executive level. The address trap may be enabled or disabled under program control.

Input/Output Channels

Various types of I/O channels are available with a system. Each channel is designed for a particular input/output data transfer application. A brief description of each type follows. A more detailed discussion of the I/O channels is provided in Section IV.

Programmed Input/Output Channel (PIOC)

This is an I/O channel capable of implementing a single word, eight-bit, parallel data transfer between the CPU and a suitable peripheral device. This channel has provisions for installing up to four unit interface controllers on the I/O circuit board. In addition, the PIOC can drive up to 12 additional remote device controllers. This board also contains a programmable Interrupt Generator which may be used in multi-processor installations. If required, one or two Real Time Clocks may be installed on the board.

Universal Block Channel (UBC)

A UBC implements and controls automatic data transfers between memory and a suitable peripheral device. The UBC contains two I/O ports with data transferred through each port in a 24-bit parallel word format. Each port provides either command chained block transfers or programmed I/O transfers. Chained block transfer capability permits the transfer direction to be reversed and a subsequent data block to be automatically transferred, without program intervention. Addressing and block size (number of words transferred) are established under program control. Once initiated, all UBC operations proceed automatically. When operating in the chained block mode, two word (48 bits) transfers, to and from memory, take place. Each port is also capable of programmed I/O operations in which single word (24 bits) transfers take place between the CPU and peripheral device. The UBC can drive up to 16 external device controllers (limited by transfer rates).

Direct Memory Access Communications Processor (DMACP)

The DMACP is a multiport I/O channel dedicated to serial data communications. Direct Memory access is provided for up to eight communication devices. These devices can be either asynchronous or synchronous. Up to eight asynchronous interfaces can be used, or one synchronous and up to four asynchronous interfaces can be accommodated. The one synchronous interface takes the place of four asynchronous interfaces. Each interface is termed a port. Standard interfaces available are RS-232C, 20 ma current loop, and Harris differential.

Single word or block data transfers of 24-bits are performed between the DMACP and CPU. Single word transfers are used for status checking, initialization and control of the DMACP. Block mode transfers are used for data transfers between main memory and the communication devices attached to the ports. These transfers are under control of the microprocessor installed on the DMACP board and require no intervention by the CPU. Transfers between the DMACP and main memory are in the form of 24-bit words, while transfers between the DMACP and communication devices are in the form of 8-bit bytes.

External Block Channel (XBC)

An XBC is similar in operation to one port of a UBC except that address control and block length are provided by an external device, and that no command chaining capability is provided. One XBC board can support up to eight external device controllers.

Integral Block Channel (IBC)

An IBC performs automatic data transfers in a manner similar to one port of a UBC. The IBC contains provisions for the installation of up to two interface controllers directly on the channel board. The IBC transfers data in a multiplex mode between the device controllers and memory. Only data chaining may be performed by the IBC; no single word transfer or command chaining capability is provided for the IBC.

Communications Multiplexer

The Communications Multiplexer (Comm Mux) is a multiport I/O channel for multiple local and remote communications devices. Interface to the CPU is by means of the PIOC. Both synchronous and asynchronous line interfaces to the Comm Mux ports are available for modems or terminals. Synchronous ports are configured singly, with up to eight ports available per Comm Mux. Asynchronous ports are configured in pairs to provide a maximum of sixteen ports per Comm Mux. A Comm Mux may be configured with any combination of synchronous and asynchronous ports up to the board limit.

Scientific Arithmetic Unit (SAU)

Available as an option with the S125 and S135, the SAU provides concurrent floating-point arithmetic capability independent from the CPU. A special repertoire of instructions is provided for CPU-SAU transfers and for performing double-precision, floating-point computations.

The SAU contains its own registers for manipulating double-precision quantities and for selecting arithmetic status (condition) after the operation is completed. Data and condition information are displayed on the Programmer's Control Panel. An executive trap is provided with the SAU for detection of overflow/underflow conditions. Refer to Section VI for a more detailed description of the SAU.

Program Restrict and Instruction Trap

These two functions are supplied as one integral unit. The Program Restrict provision allows areas of memory to be selected under program control, protected from unauthorized access, and guarded against accidental modification. When the CPU is in a restricted environment, the Instruction Trap provides the means for preventing execution of a predefined group of instructions. Two registers, two executive trap interrupt levels, and the associated control logic comprise the option.

Real Time Clock

This option provides the programmer with general purpose clock pulses that are independent of the mainframe clock pulses. With an accuracy of .05%, the real time clock pulses are available whether the CPU is in standby or not. The timing pulses can be used to measure user's program running time, or to generate periodic interrupts. Programming is accomplished through normal input/output commands. One or two real time clocks may be installed on the Programmed Input Output Channel (PIOC) boards.

Run Time Meter

Available as an option, the Run Time Meter records the time that power is applied to the computer, i.e., whenever the circuit breaker is in the ON position. Elapsed time is measured in hours and tenths up to 99,999.9 hours.

MOS Data Save

The MOS Data Save option provides voltages necessary for the refresh circuits of the MOS Memory to maintain data integrity during ac power failures. Voltages are provided by a battery back-up system. Battery voltages are maintained by a trickle charge during periods of normal ac line voltage levels.

This option consists of a Master Module and up to seven Expansion Modules. Data save protection for one MOS memory board is provided by the Master Module. As memory is expanded, the proper number of Expansion Modules are added to provide the required data save voltages. Battery back-up is limited to eight MOS memory modules.

I/O Expansion Chassis

Available as an option with the S125, the expansion chassis increases the I/O capacity of the system. All necessary hardware and power supplies are provided with the option.

Computer Link

This option permits block data transfers between interconnected computers in a dual computer installation. The computer link is particularly useful in real-time control applications involving dual computers.

Multi-CPU Channel Adapter

Used in a multiple computer configuration, the multi-CPU channel adapter allows peripheral devices to be shared by two or more computers.

PERIPHERAL EQUIPMENT

The Harris Series 100 systems can be expanded and enhanced by selection of various peripheral equipment offered with each system, including:

- Moving Head Discs (40, 80, 150 and 300M Bytes)
- Cartridge Discs (10.8M Bytes)
- Fixed Head Discs (.5, .8, 1.1, 1.7 and 2.1M Bytes)
- Floppy Discs (310K Bytes)
- Magnetic Tapes (45, 75, 100, 150 and 200 ips)
- Card Readers (300, 600 and 1000 cpm)
- Card Reader/Punch (500/100 cpm)
- Line Printers (300, 600 and 900 lpm)
- Electrostatic printer/plotter (300, 500, 1000, and 1200 lpm)
- Paper Tape Devices
- Console Devices, Local and Remote Terminals
- Supplementary equipment to meet most custom requirements

SOFTWARE

VULCAN Operating System

The Virtual Memory Manager (VULCAN) is a priority-structured, demand paged, multi-programming operating system. VULCAN concurrently supports multi-stream batch processing, interactive terminal time-sharing, transaction-oriented processing, multiple remote job entry and real-time operations. Under VULCAN, the virtual memory hardware/software system is transparent to the user. Up to 768K bytes (K = 1024) per user is available — of which up to 192K bytes may be executable code.

Support Software

The field-proven VULCAN operating system supports seven languages, five support programs and a programmable macro job control command language. Also available as options are the Harris TOTAL data base management system, interactive retrieval language, four remote job entry support packages and two remote batch terminal host packages.

Languages

- FORTRAN IV Compiler with extensions
- Interactive BASIC Compiler with extensions
- 1974 ANSI COBOL Compiler
- RPG II Compiler
- Harris MACRO Assembler
- SNOBOL 4 Interpreter
- FORGO (Diagnostic Load-and-Go FORTRAN Compiler)

Support Programs

- Sort/Merge
- Indexed Sequential File Handler
- System Accounting
- Cross Reference
- VBUG Symbolic Debugger

Remote Job Entry (RJE) Support Packages

- IBM HASP II M/L
- IBM 2780
- CDC 200 UT
- UNIVAC 1004

Remote Batch Terminal (RBT) Host Packages

- IBM HASP II M/L
- IBM 2780

Data Base Management Systems (DBMS)

- TOTAL Basic
- TOTAL Central
- TOTAL – IQ

Harris TOTAL – the most widely used of all the Data Base Management Systems – is known for its efficient implementation, low memory requirements and ease of use. TOTAL DBMS supports network and hierarchal data structures.

SUMMARY OF CHARACTERISTICS

The major operating characteristics and pertinent technical specifications of the S115, S125, and S135 are summarized below.

Computer Organization	Microprogrammed, general-purpose digital computer, single address, multi-access central system bus structure, and buffered I/O channels.
CPU Microcycle Time	300 nanoseconds
CPU Word Length	24 bits
Arithmetic	Parallel, binary, two's complement number representation. Hardware multiply/divide/square root. Hardware double-precision, floating-point processor (SAU), optional (S125 and S135 only).

Instruction Execution Time (microseconds). Values are for MOS Memory. Minimum floating-point times given.

Instruction	Register to Register	Memory Reference	Double-Precision Floating Point
Arithmetic			
Add/Subtract	0.72	1.44	2.73
Multiply	6.12	6.54	6.23
Divide	12.72	13.14	11.43
Square Root	35.12	N/A	8.53
Algebraic Compare	1.02	1.44	2.43
Logical Compare	0.72	N/A	N/A
Input/Output	0.72	N/A	N/A
Shift n places	1.02 + 0.15n	N/A	N/A
Transfers	0.72	1.44	0.75

MAIN MEMORY

Semiconductor

Type	N-Channel MOS
Minimum Size	16K words
Maximum Size (S135 only)	256K words
Increment	16K words
Word Length	48 bits (double word)
Parity	One bit error correct
Access	Random
Power Fail	Battery back-up

Core

Type	3-wire, 3D planar core array
Minimum Size	32K words
Maximum Size (S135 only)	256K words
Increment	16K words
Word Length	48 bits (double word)
Parity	Data parity check
Access	Random
Power Fail	Data retention

ADDRESSING

Monitor Mode	Immediate Direct to 32K words Direct to 64K words via long address instructions Indirect to 256K words (data only) Indexed to 64K words
User Mode	Each user has an address space equivalent to the amount of physical memory. User addresses are translated to 256, 1024, or 4096-word physical pages. Hardware detects accesses to non-resident pages.

INPUT/OUTPUT CAPABILITY

Programmed Data Transfers	Single word to/from CPU register, 8 or 24 bits.
Automatic Data Transfer	Direct memory access via UBC, IBC, XBC and DMACP.

Single Channel Maximum Transfer Rates (words/sec.)

	Input	Output
UBC (1 port active)	1,336,000	1,000,000
IBC	Device Dependent	Device Dependent
XBC (no mainframe contention)	800,000	666,666
(with mainframe contention)	476,000	428,000
DMACP	2,700	2,700

Input/Output Command Modes

Normal	Normal operation for each channel type
Multiplex	Channel released to master/slave peripheral units. Not available on IBC, XBC, or DMACP
Offline	Channel drivers turned off allowing second CPU to share devices without need for peripheral switches. Not available on IBC.
Reset	Resets Multiplex or Offline Mode. Channel restored online and unit selected. Not available on IBC.

Priority Interrupt Structure

Internal	Maximum of eight executive traps. Dedicated memory locations.
External	Sixteen priority interrupt levels, standard. Optionally expandable to 24 priority interrupt levels. Dedicated memory locations.
Control	External interrupts may be individually armed, disarmed, enabled, inhibited or triggered under program control.
Power Fail Protection	Power fail shutdown and restart, standard.

Electrical requirements

Voltage	230 VAC or 208 VAC ±10% (115 VAC ±10%, optional on S115 and S125)
Frequency	60 ± 3 Hz (50 ± 3 Hz, optional)
Current	19 Amps RMS at 230V (maximum)
Power	4000 Watts
Phase	Single phase, 4-wire with NEMA L14-30R twist-lock connectors (250 VAC) on a 15 ft. power cord.

Environmental Requirements

Temperature	
Operating	50° F to 113° F (10° C to 45° C), ambient air
Storage	32° F to 122° F (0° C to 50° C), ambient air
Humidity	
Operating	20% to 80%, relative (non-condensing)
Storage	20% to 90%, relative (non-condensing)
Altitude	
Operating	-1000 to 6000 ft. (-305 to 1829 m)
Storage	-1000 to 15,000 ft. (-305 to 4572 m)
Heat Dissipation	13,650 BTU/hr. (3400 kg-cal/hr.)
Cooling	Forced air provided by internal fans on each chassis

SECTION II
CENTRAL PROCESSING UNIT

GENERAL DESCRIPTION

The Central Processing Unit (CPU) is a single-address, 24-bit parallel word-oriented, stored-program processor. Operations performed by the CPU include data transfers, arithmetic computation, and logical manipulation. These operations are defined by instructions stored in, and retrieved from, physical memory. The specified operation is performed on single-word, double-word, byte, or single bit operands stored in memory or contained in one of the CPU registers. Data word formats, as defined by both hardware and software, are illustrated in Figure 2-1.

In addition to the general and special-purpose registers, the CPU contains an arithmetic section that performs the actual computation and logical manipulation of operands, and a control section that retrieves and decodes instructions from memory and directs the functional processes of the system. The control section also includes the paging logic that implements the memory address translation and demand-paging operations. The CPU contains interface elements for communications with the other computer elements; e.g., memory, the I/O channels, the control panel, and the optional Scientific Arithmetic Unit (SAU).

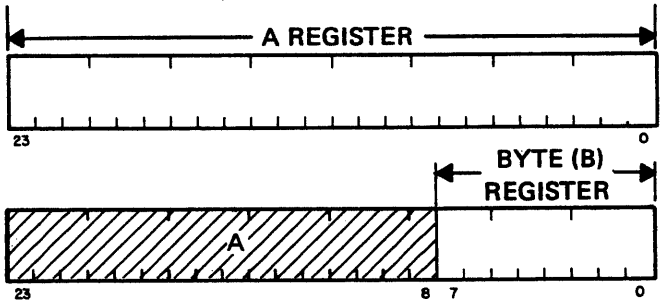
BASIC CPU REGISTERS

Introduction

The following paragraphs provide a brief description of the basic registers in a CPU. Registers associated with the priority interrupt system and SAU are described elsewhere, in the appropriate sections of this manual.

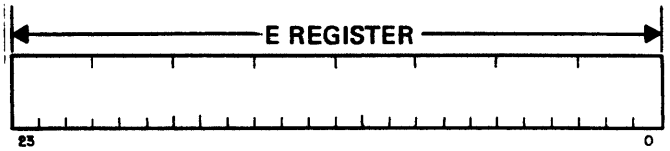
A and B Registers

Serving as the principal arithmetic accumulator, the 24-bit A Register also functions as the input/output communication register during programmed (single-word) transfers between the CPU and peripheral devices. The A Register has complete arithmetic and shift capability. Bits 7-0 of A form an 8-bit pseudoregister, termed the B (Byte) Register. Both the A and B registers are accessible to the user by means of the instruction set and the Programmer's Control Panel.



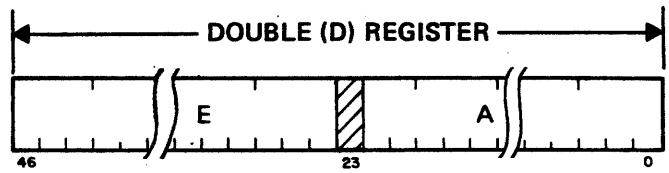
E Register

Employed as an extension of the A Register for increased arithmetic and shift capability, the 24-bit E Register also functions as a general-purpose storage element during various instructions. The E Register is accessible through both the instruction set and the Programmer's Control Panel.



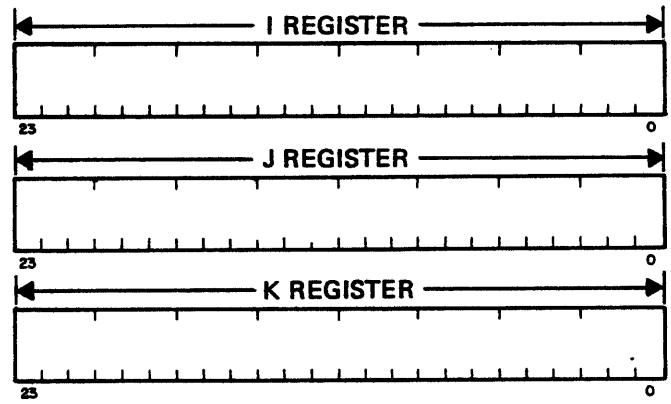
D Register

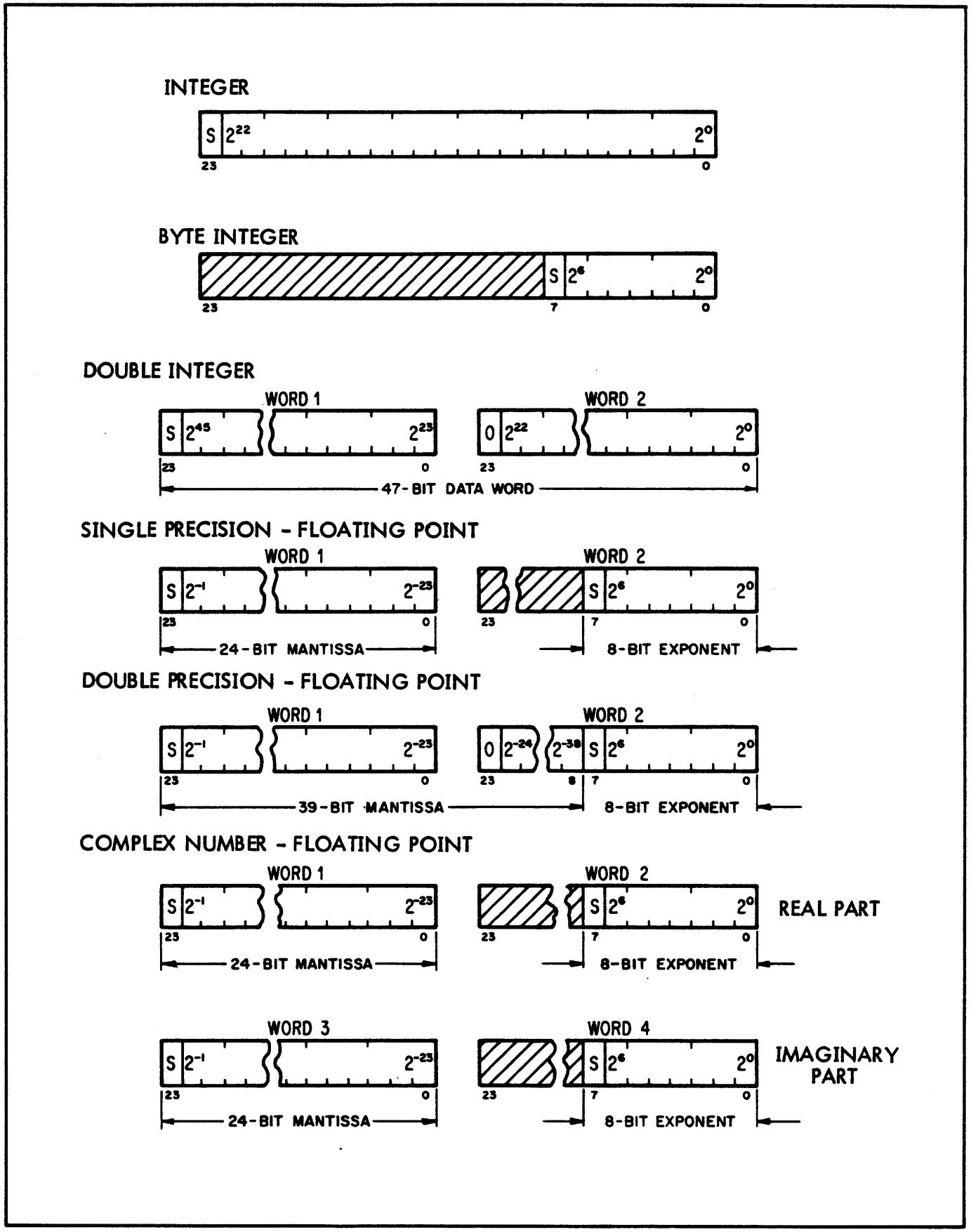
The D (Double) Register is a 47-bit pseudoregister formed by combining A and E to provide double-precision arithmetic and shift capability. The A and E Registers form the least-and most-significant halves, respectively, of the 47-bit double-precision quantity (bit 23 of A is not used). Several instructions provide direct access to the D Register; Programmer's Control Panel entry, however, must be accomplished by accessing the E and A Registers in the proper format.



I, J, and K Registers

Each of these is an independent, 24-bit general-purpose register that can also be employed as an index register for address modification. The I, J, and K Registers are directly accessible through the instruction set and the Programmer's Control Panel.





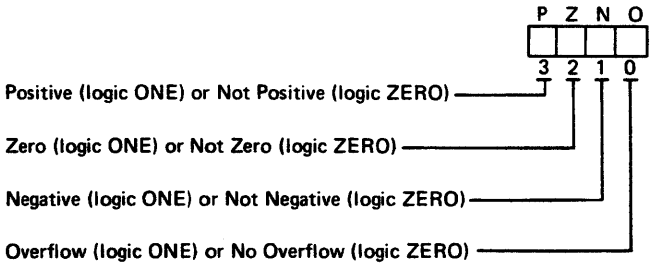
MI60-033-7708

Figure 2-1. Data Word Formats

Condition Register

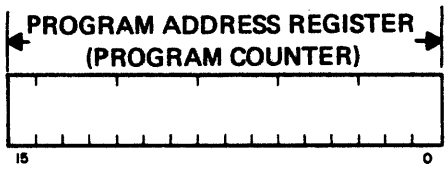
A 4-bit element that stores the results of specific operations, the Condition (C) Register is accessible by means of several instructions. Display for the C Register is provided by the Programmer's Control Panel.

CONDITION (C) REGISTER



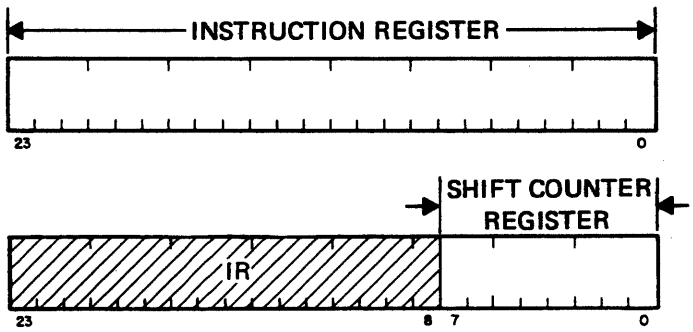
Program Address Register

Also called the Program Counter, the 16-bit Program Address (P) Register retains the memory address from which the current instruction is fetched. A maximum of 65,536 memory locations can be accessed via the P Register. The register can be loaded with a TOC instruction and its contents saved with a BSL instruction. The contents of the P Register can be modified through the execution of any of several branch instructions. The Programmer's Control Panel provides direct entry and display for the P Register.



Instruction Register and Shift Counter Register

Once an instruction has been fetched from memory, it is retained in the 24-bit Instruction Register during decoding and execution. The Instruction Register is not programmable. Bits 7-0 of the register serve as the Shift Counter Register and is programmable via all shift instructions. Entry and display of the register is provided through the Programmer's Control Panel.



ADDRESSING FUNCTIONS

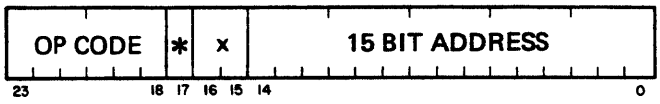
The virtual memory operates in two distinct addressing modes, "Monitor" and "User". In the Monitor Mode, the paging logic is disabled and addressing is identical to that of a non-virtual memory system. In the User Mode, the paging logic is activated and all effective address references generated by the CPU are subjected to a memory translation technique. All effective memory addresses are translated. This includes addresses defined by the Program Counter and all memory reference instructions, including indirect and indexed operations.

Basic Addressing Technique

Total memory available to the CPU is 262, 144 (256K) words. Because of the CPU's basic architecture and the corresponding addressing technique, executable code (programs) is confined to the lower 64K (0-65,536 words) of memory. However, memory above 65K may be addressed by means of special indirect references. Figure 2-2 illustrates the memory referencing sequence.

Direct Addressing

A standard memory reference instruction format is shown below. The 15-bit address field (bits 14-0) in the instruction word provides direct access to 32,768 (32K) words.

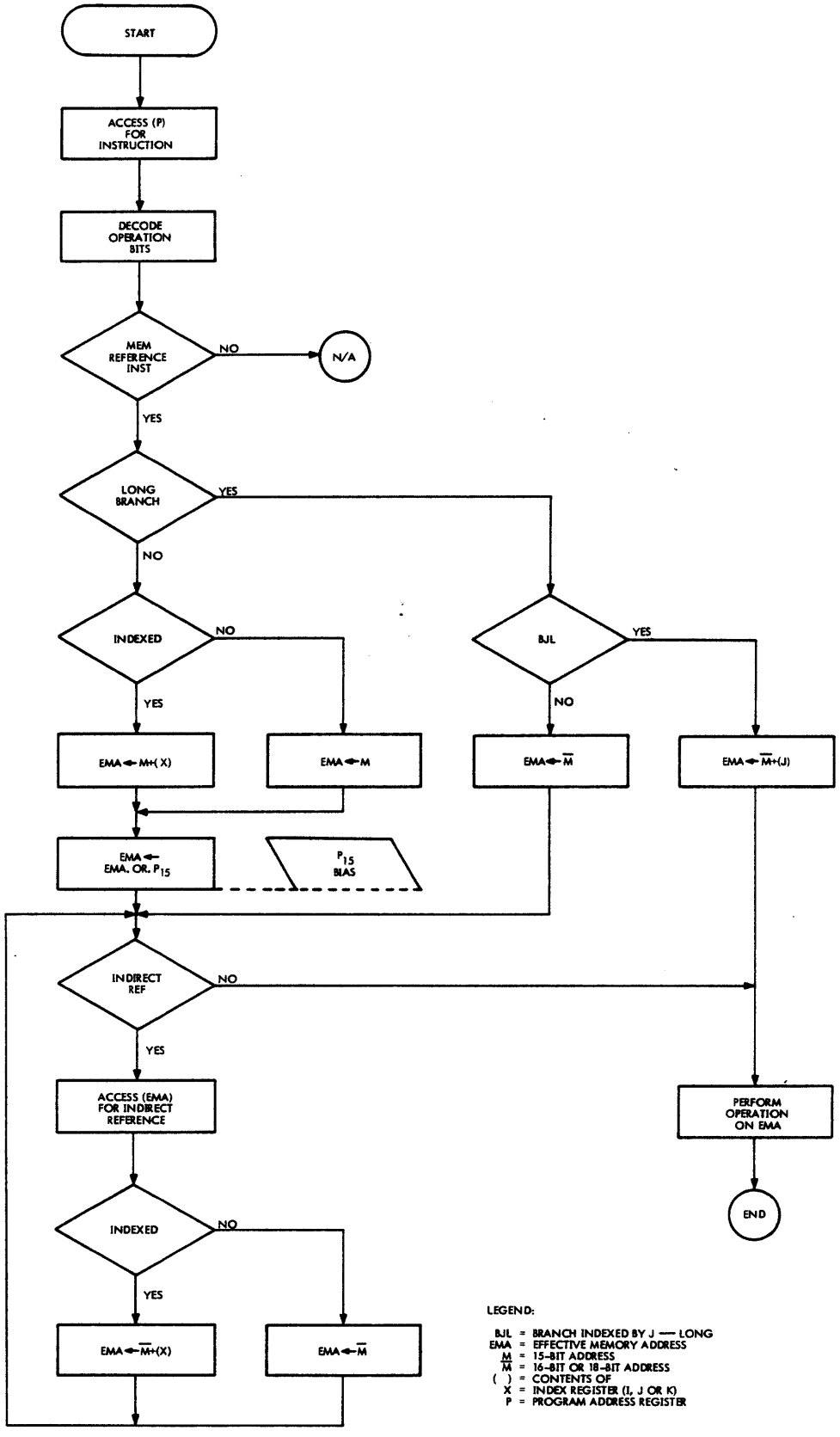


The addressing logic divides the lower 64K of memory into two areas; 0-32K and 32K-64K. Under this method, the most-significant bit (P15) of the Program Counter is used to bias all direct address references. P15 = 0 specifies an address in the lower 32K, while P15 = 1 designates a location in the upper 32K of the 0-64K memory increment. By performing a logical-OR function between the immediate (direct) address reference and P15, instructions may directly address up to 32K words within their respective sections of memory.

NOTE

An instruction in the last location of the lower memory section should not reference another address in the lower section. By the time the effective address is computed, the Program Counter will have advanced to bias the immediate address reference by 100000_8 to specify an effective address in the upper memory section.

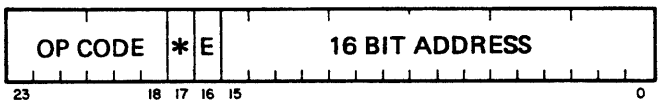
Modification of a 15-bit direct address by means of the indirect (*) and/or indexing (X) features can permit an instruction to address any memory location up to 256K words.



MI60-003-10688

Figure 2-2. Memory Referencing Sequence

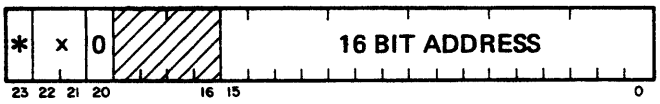
A special group of "long branch" instructions permit direct addressing up to 64K words. The instruction word format for this group is shown below. Note that these instructions may be modified by indirect references (*), but have no provision for indexing. Long branch instructions are not biased by P15. Bit 16 is used to extend the Op Code.



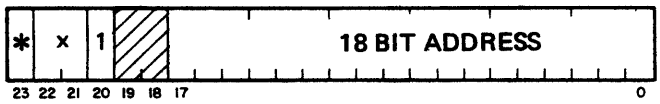
Indirect Addressing

Indirect address references permit the CPU to access up to 256K words of memory. When a memory reference instruction is decoded, bit 17 (*) of the instruction word is examined. If bit 17 is set (ONE), an indirect address reference is indicated. An indirect reference signifies that the effective address (defined by the instruction word plus any index count) contains a second address rather than an operand. The word retrieved from memory when the effective address is cycled is treated as an indirect address word. Indirect address word formats are illustrated below.

STANDARD INDIRECT FORMAT



LONG ADDRESS FORMAT



The standard indirect format, with its 16-bit address field, permits access of up to 64K words. Up to 256K words can be accessed by the 18-bit field in the long address word. Neither type of indirect address is affected by the P15 address bias bit.

Bit 23 (*) of either indirect format may be set to specify another level of indirect addressing. Each level of indirect reference may be individually indexed to provide further address modification.

Indexing

A direct or indirect address reference may be modified by indexing. This operation adds the address in the current instruction or indirect reference to the contents of a specified index register (I, J, or K) to determine an effective address. A two-bit field (X) in the instruction or indirect reference specifies which register will be employed in each indexing operation. Figure 2-3 provides some examples of indexed addressing.

In the lower 32K memory section (P15 = 0), immediate address references may be indexed to access up to 65,536 words. However, instructions in the 32K-64K section of memory (P15 = 1) may not reference the lower section by indexing since all immediate address references will be biased by 100000_8.

Addressing, User Mode

Introduction

Paging is a hardware addressing scheme that allows a program's memory area to be discontiguous. Program segments may be absent from physical memory while other portions of the program are being executed. This aspect of the paging operation, termed "demand-paging", also allows the computer to execute programs larger than the available physical memory; hence, the term "Virtual Memory". The following paragraphs discuss the paging hardware and describe the basic functions of the VM.

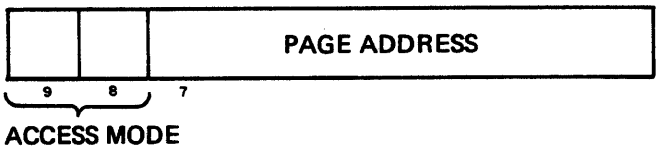
Virtual Memory Registers

Various registers are supplied with the VM paging logic. A brief description of each is provided in the following paragraphs. Entry and display of all VM registers is provided on the Programmer's Control Panel.

Virtual Address Registers (VARs)

At the user's option, a total of 256, 1,024, or 4,096 of these 10-bit VARs are supplied. The eight least-significant bits (7-0) retain the address of a physical memory page, while bits 9 and 8 define the manner in which the specified page may be accessed. The access modes and their corresponding bit configurations are defined in the paragraph describing demand paging operation. Specific operations within the VM instruction repertoire provide transfers to and from the VARs.

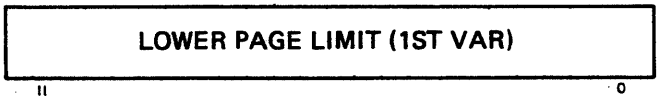
VIRTUAL ADDRESS REGISTER (VAR)

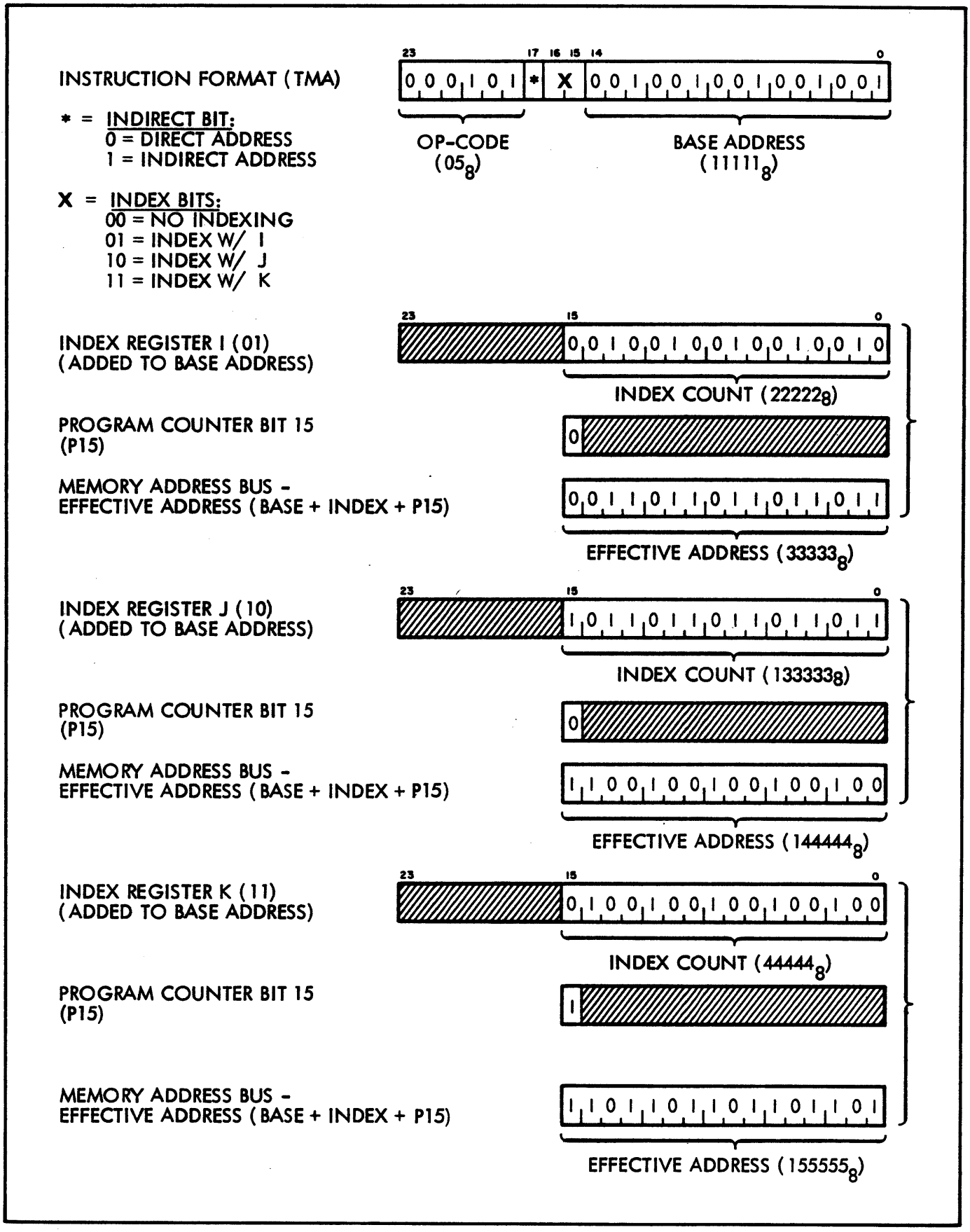


Virtual Base Register (VBR)

The 12-bit VBR retains the lower page limit of the user program; i.e., the address of the first assigned VAR for the currently-executing program. Special VM instructions provide for loading the VBR and retrieving its contents.

VIRTUAL BASE REGISTER (VBR)





M160-028C-1176

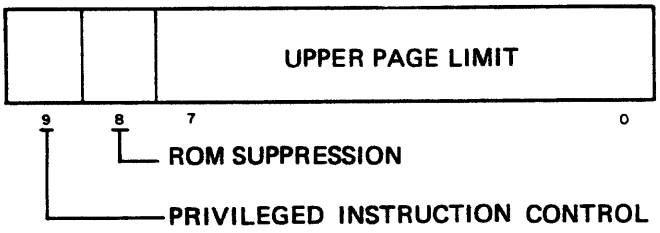
Figure 2-3. Examples of Indexed Addressing

Virtual Limit Register (VLR)

Bits 7-0 of the 10-bit VLR define the upper page limit of a user program, i.e., the number of VARs minus 1 which the program may reference; bits 9 and 8 provide special controls. When bit 9 is a ONE, any of a group of privileged instructions (see paragraph describing instruction trap provision) may be executed without generating an instruction trap interrupt. When bit 8 is a ONE, the Release Operand Mode (ROM) instruction will be suppressed.

The VLR may be loaded or its contents may be retrieved by specific VM instructions.

VIRTUAL LIMIT REGISTER (VLR)



Virtual Usage Registers (VURs)

A total of 256 of these one-bit registers are supplied; one is associated with each physical page of memory. Each time a given memory page is accessed by a CPU instruction, a ONE is stored in the appropriate VUR. The VURs may be selectively tested and cleared under program control.

Virtual Not-Modified Registers (VNRs)

A total of 256 of these one-bit registers are supplied; one is associated with each physical page of memory. Each time data is written (stored) in a given memory page by an instruction reference, a ONE is stored in the appropriate VNR. The VNRs may be selectively tested and cleared under program control.

Virtual Usage Base Register (VUB)

This 8-bit register retains the address of one of the VURs or VNRs (equivalent to the associated physical page). This address is used as a pointer to access the appropriate VUR or VNR during the Query Virtual Usage Register (QUR) or Query Not-Modified Register (QNR) instruction. The VUB can be loaded or its contents retrieved by special VM instructions.

VIRTUAL USAGE BASE REGISTER (VUB)

Virtual Source Register (VSR)

This 12-bit register retains the address of one of the VARs and is used as a pointer for retrieving data from the VARs during a Transfer 2 Virtual Address Registers to Double (TRD) instruction. The VSR can be loaded under program control.

VIRTUAL SOURCE REGISTER (VSR)

Virtual Destination Register (VDR)

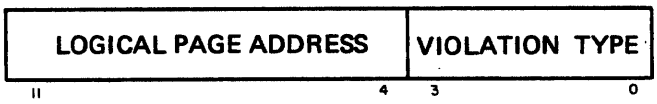
The 12-bit VDR retains the address of one of the VARs, and is used as a pointer for storing data in the VARs during Transfer A to 1 Virtual Address Register (TAR) and Transfer Double to 2 Virtual Address Registers (TDR) instructions. A special VM instruction provides program-controlled loading of the VDR.

VIRTUAL DESTINATION REGISTER (VDR)

Virtual Demand Page Register (VPR)

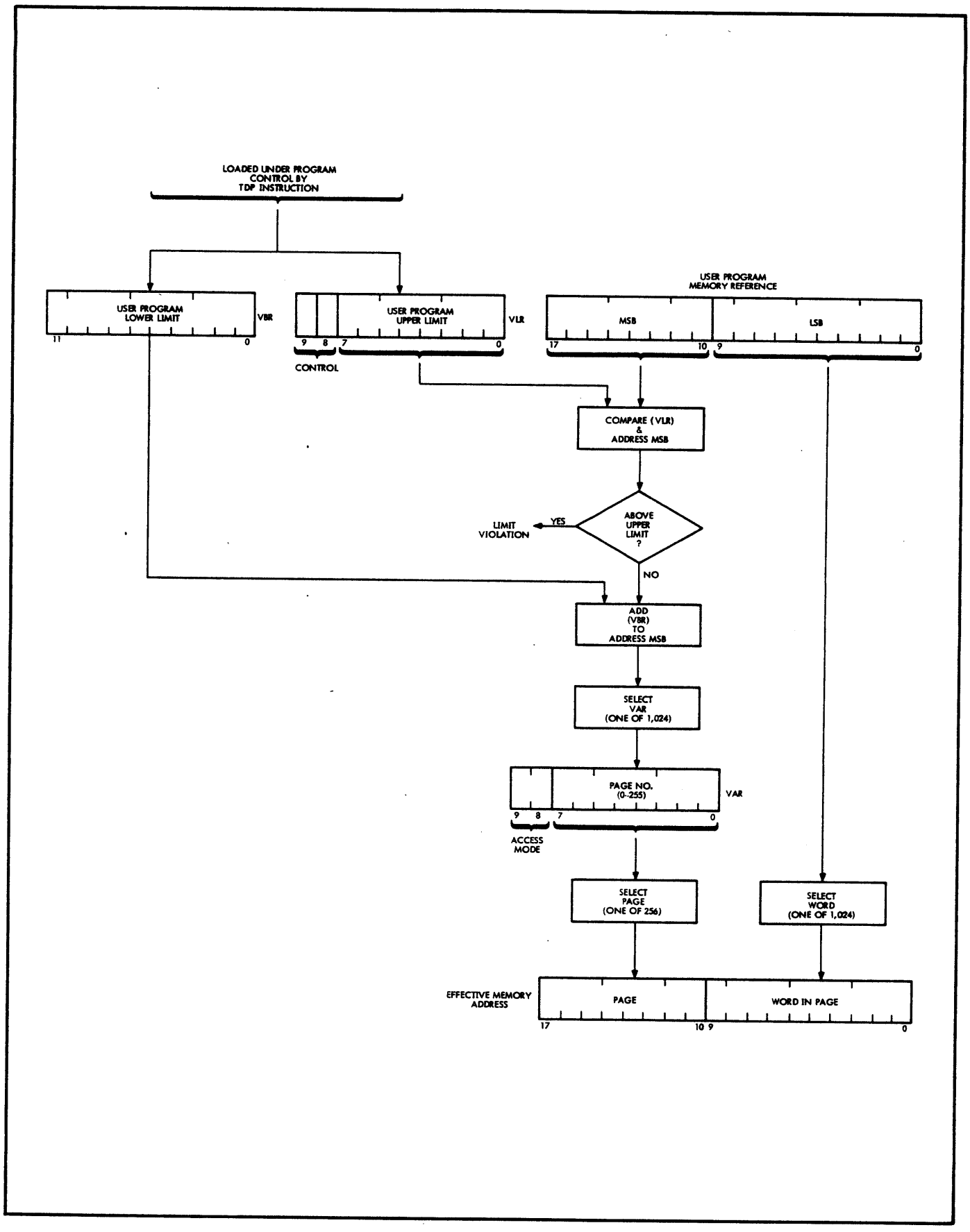
A special register (VPR) is used in the virtual memory system to copy the logical page address (bits 11-4) of the user program for each memory reference so that if a particular cycle causes a fault, the operating system knows which logical page is involved and the condition that caused the fault. The address of the VAR that created a demand page or limit register violation is the contents of the VBR plus the contents of the VPR. Bits 3-0 identify the type of violation. The contents of the VPR may be retrieved under program control.

VIRTUAL DEMAND PAGE REGISTER (VPR)



Basic Address Translation

In a VM system, memory is divided into 1,024 (1K)-word "pages", A translation scheme is applied to the most-significant bits of all memory references. This scheme consists of adding a base address (VBR contents) to the address bits and subsequent translation by special registers to select a page of memory. The remaining bits of the original memory reference are used to select a specific word within the selected page. Figure 2-4 illustrates the address translation scheme of the VM logic. Figure 2-5 provides an example of the address translation using a standard memory reference instruction.

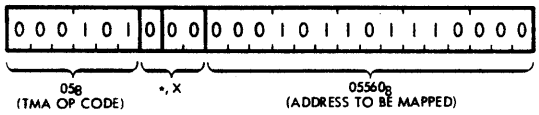


MI1820-176

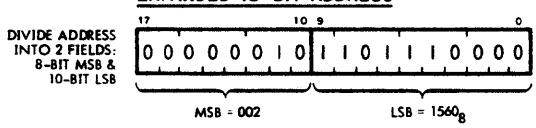
Figure 2-4. Basic Address Translation, VM User Mode

LOCATION (OCTAL)	LABEL	MNEMONIC	OPERAND
00005		TMA	XYZ
05560	XYZ	DATA	5

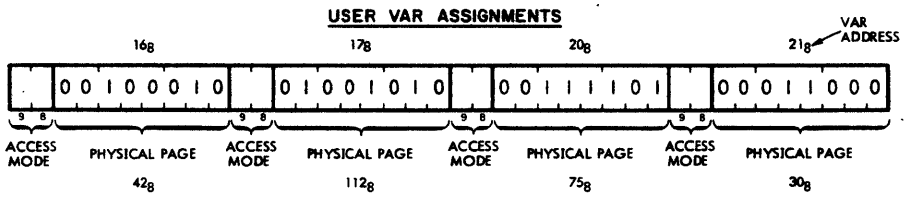
MACHINE LANGUAGE REPRESENTATION FOR TMA INSTRUCTION



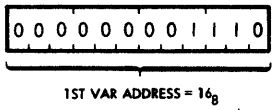
EXPANDED 18-BIT ADDRESS



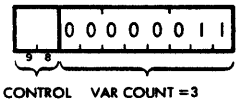
USER VAR ASSIGNMENTS



VBR



VLR



1. (VBR) + MSB = VAR ADDRESS OR $16_8 + 2_8 = 20_8$
2. (VAR ADDRESS) = PHYSICAL PAGE OR 75_8
3. MEMORY ADDRESS IS WORD 1560_8 (LSB) OF PAGE 75_8
4. PHYSICAL ADDRESS IS 173560_8

MI1829C-977

Figure 2-5. Address Translation Example, VM User Mode

Address translation is implemented via the 10-bit Virtual Address Registers (VARs) and the Virtual Base and Virtual Limit Registers (VBR and VLR). Each VAR has a unique number, or address, from 0 through 256, 0 through 1,024, or 0 through 4,096 depending on the system in use. A specific VAR is selected by adding the eight most-significant bits (MSB) of the 18-bit memory reference to the contents of the VBR. The selected VAR, in turn, contains an address corresponding to 1-of-256, 1,024-word pages.

In practice, the user program is assigned (by the software operating system) a group of sequential VARs. The lower limit of the user program area, and the base for computing VAR addresses, is established by loading the VBR with the first VAR address in the group. The user program upper limit is established by the VLR contents corresponding to the number (quantity) of assigned VARs. Since the MSB value is added to the VBR to compute VAR addresses, the VLR must contain a quantity that is one less than the number of VARs assigned to the user's program. Referring to Figure 2-5, VAR address 16_8 is specified when the MSB value equals 0, 17_8 when MSB equals 1, 20_8 when MSB equals 2, and 21_8 when MSB equals 3. In this example, the VLR would be preloaded with a count of 3. When the MSB value exceeds this count, a limit violation will be generated. See paragraph describing demand paging operation. The VARs, VBR, and VLR are loaded under program control.

Demand Paging

Demand paging is the aspect of the VM hardware that permits a portion of the user's program to be absent from physical memory (and located instead on a disc mass-storage device) while the program is being executed. When the address translation logic detects a reference to a non-resident page, a special hardware interrupt (Group 0, Level 2) is triggered. Subsequent processing by the operating system may then access the desired page and load it into physical memory. If sufficient memory space is not available, the operating system may interchange inactive resident program segments with the incoming page(s) or programs (i.e., transfer the inactive segments to the disc storage device). Once the correct program sequence is loaded into physical memory, the user's program may continue its normal sequence.

A non-resident page is signified by ZEROs in bit positions 9 and 8 of the selected VAR. Each time a VAR is accessed, these bits are examined by the paging control logic to determine if a demand page is required. The address of the VAR that contains the non-resident page is stored in bits 11-4 of the Virtual Demand Page Register (VPR). The address stored in the VPR is relative to the lower limit of the user program (stored in the VBR) and is equal to the MSB value of the address reference (Figure 2-5).

The interrupt generated at Group 0, Level 2 may reflect a limit register or restrict mode violation as well as a demand page. Bits 3-0 of the VPR define which condition generated the interrupt; these are examined by the operating system to determine what steps are to be taken in processing the interrupt. Entry into an interrupt-processing routine requires saving a return address; usually, the interrupt address plus one. Certain situations require reexecution of the instruction that created the demand page or violation; consequently, the program counter must be adjusted to fetch the instruction again. The operating system makes the appropriate adjustment based on the code in VPR bits 3-0. Table 2-1 defines the VPR status and control bits.

The paging logic provides a program restrict system that permits pages of memory to be protected from unauthorized access. A user's program area is defined by the contents of the Virtual Base Register (VBR) and Virtual Limit Register (VLR). The VBR defines the lower page limit in the user's program while the VLR defines the last page, or upper limit. No user's programs can reference any memory location below the lower page limit because all addresses are biased by the VBR's contents during the address translation operation. Any attempt to reference memory above the upper limit will result in a limit register violation and trigger the Group 0, Level 2 executive trap interrupt.

Each page of memory can be further protected by placing it in one of three access modes. Bits 9 and 8 of the VARs contain the access mode bits for the associated page. Any attempt to access the selected page in any manner other than that specified in the mode bits will result in triggering the Group 0, Level 2 executive trap. The access mode bits are defined below.

Mode	Bit 9	Bit 8	Description
0	0	0	Page Missing —page is not contained in physical memory (demand page).
1	0	1	Unrestricted —instructions may be executed within the page and data may be loaded from or stored within the page.
2	1	0	Execute/Read —instructions may be executed within the page or data loaded from the page; data may not be stored within the page.
3	1	1	Read —data may be loaded from the page; instructions may not be executed within the page and data may not be stored within the page.

The program restrict functions are enabled only when the VM system is in the User Mode. The paging logic is manually enabled/disabled by the PROG REST/OFF/VM key switch located on the control panel.

Table 2-1. VPR Status Bits/Definitions and Functions

Condition	VPR Bits 3 2 1 0	Type of Violation	Instruction or Sequence Causing Violation	Program Counter Adjustment
1	0 0 0 1	Demand Page	Operand address of EXM or branch instruction	Do not change.
2	0 0 1 0	Demand Page	Operand address of memory reference instruction; OR USP or AOM instruction; OR Indirect chain.	Decrement by one.
3	0 0 1 1	Demand Page	ROM instruction	Decrement by two.
4	0 1 0 1	Mode 3*	Same as Condition #1	Do not change.
5	0 1 0 1	Mode 3*	Same as Condition #2	Decrement by one.
6	0 1 1 1	Mode 3*	Same as Condition #3	Decrement by two.
7	1 0 0 1	Mode 2*	Same as Condition #1	Do not change.
8	1 0 1 0	Mode 2*	Same as Condition #2	Decrement by one.
9	1 0 1 1	Mode 2*	Same as Condition #2	Decrement by two.
10	1 1 1 0	Limit Register	Same as Condition #1	Do not change.
11	1 1 1 0	Limit Register	Same as Condition #2	Decrement by one.
12	1 1 1 1	Limit Register	Same as Condition #3	Decrement by two.

*Page Access Mode Violation

Instruction Trap Provision

An instruction trap function is included as an integral part of the paging hardware. The trap prevents the execution of certain, predetermined, instructions. When the trap is enabled, any attempt to execute one of the designated instructions will result in an executive trap interrupt at Group 0, Level 3.

The instruction trap function is automatically enabled when the paging logic is placed in the User Mode. When enabled, the trap will analyze bit 9 in the Virtual Limit Register (VLR). When VLR bit 9 is set (ONE), the following instructions may be executed without generating an instruction trap violation. If bit 9 is reset (ZERO) and the instruction trap is enabled, a violation will occur when an attempt is made to execute any of the following instructions.

HaLT (HLT)
Output Data Word (ODW)
Input Data Word (IDW)
Output Command Word (OCW)
Input Status Word (ISW)
Output Address Word (OAW)
Input Address Word (IAW)
Input Parameter Word (IPW)
Hold eXternal Interrupts (HXI)
Release eXternal Interrupts (RXI)
Unitarily Arm group 1 interrupts (UA1)
Unitarily Disarm group 1 interrupts (UD1)
Unitarily Enable group 1 interrupts (UE1)
Unitarily Inhibit group 1 interrupts (UI1)
Transfer Double to group 1 (TD1)
Transfer Double to group 1 (TD4)
Transfer Double to Limit Register (TDL)

If the instruction trap is enabled, the VM group of instructions will result in a violation (VLR bit 9 has no effect on this group) if the user program attempts to execute them.

NOTE

Any attempt to execute an Interval Timer start or stop instruction in the User Mode when VLR bit 9 is reset causes the instruction to be treated like a NOP. No interrupt is generated. The following instructions are affected:

Hold Interval Timer (HIT)
Release Processor Time (RPT)
Release Clock Time (RCT)
Any register to register instruction that loads the T Register, such as a TAT instruction, etc.

Paging System Control

When a master clear is generated, the Monitor Mode will be established. The paging logic will remain in the Monitor Mode until placed in the User Mode. The User Mode is established under program control (i.e., via the RUM instruction). The RUM (Release User Mode) instruction causes the User Mode to be established at the completion of the instruction following the RUM. (This instruction should, in practice, always be an unconditional branch.) After the new program address has been calculated, the User Mode will be activated. The RUM instruction, together with the following instruction, will be handled like an EXM with respect to a demand page (VPR bits 0 and 1 will be set to ONE and ZERO, respectively). Refer to Table 2-1.

A BLU (Branch and Link-Unrestricted) instruction will automatically establish the Monitor mode; the BLU's 5-bit effective memory address will not be mapped. Bit 20 of the J Register will be set (ONE) if the BLU was executed in the User Mode, and reset (ZERO) if the BLU was executed in the Monitor Mode.

When an interrupt occurs, the Monitor Mode will be established; the hardware-generated EXM (EXecute Memory) instruction will not be translated. The BSL (Branch and Save Return-Long) to the dedicated interrupt location will transmit the paging mode at the time of the interrupt to the BSL's effective memory address. Bit 20 will be set (ONE) if the system was in the User Mode and reset (ZERO) if it was in the Monitor Mode. If a demand page request occurs while executing a ROM instruction, the paging mode will be recorded as Monitor (i.e., bit 20 of the BSL's effective memory address will be reset). When returning from an interrupt routine via a BRL (indirect) instruction, bit 20 of the entry point will be tested, and the User or Monitor Mode will be re-established accordingly.

Virtual Memory Instruction Set

A virtual memory instruction set is provided for program control of paging functions. These instructions can only be executed in the Monitor Mode. If an attempt is made to execute any of these instructions while in the User Mode, an instruction trap interrupt will be generated. A detailed description of each of these instructions is provided in Section VII of this manual.

Transfer Double to Source and destination registers (TDS)
Transfer Source and destination registers to Double (TSD)
Transfer A to 1 virtual address Register (TAR)
Transfer Double to 2 virtual address Registers (TDR)
Transfer 2 virtual address Registers to Double (TRD)
Transfer Double to Paging limit registers (TDP)
Transfer Paging limit registers to Double (TPD)
Transfer Usage base register and demand page register to Double (TUD)
Transfer E to Usage base register (TEU)
Query Virtual Usage Register (QUR)
Query Not-modified Register (QNR)
Release Operand Mode (ROM)
Release User Mode (RUM)

BIT PROCESSOR

General Description

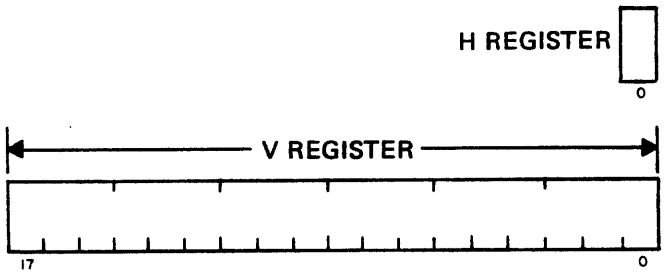
The bit processor consists of the single-bit H Register, an 18-bit V Register (base register), and the associated control logic. The bit processor provides the capability to selectively change, store, or test a bit from memory.

Bit Processor Registers

Two registers are associated with the bit processor feature. A single-bit element, the H Register, retains the bit selected for use in the operation. The 18-bit V Register is employed to store a base address that is, in turn, used to define a memory location from which the designated bit will be retrieved. Both the H and V Registers are directly programmable via the special group of bit processor

instructions. Provision is made on the Programmer's Control Panel for entry and display of the bit processor registers. The registers are entered and displayed simultaneously, with the V Register contents displayed in bit positions 17-0 of the display register indicators and the H Register contents displayed in bit position 23.

BIT PROCESSOR REGISTERS



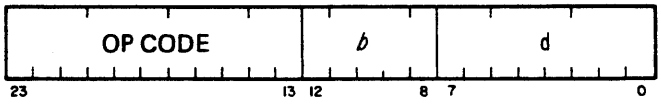
Operational Description

The 18-bit V Register is loaded with a base address which specifies a memory location to be manipulated. This is accomplished by transferring an 18-bit memory address from the K Register. The instruction word further defines the memory location, the specific bit, and the operation to be performed.

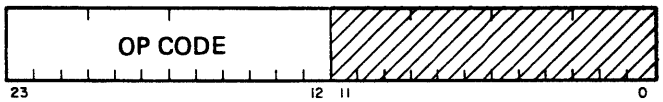
After the operation is performed on the selected bit, the results are displayed in the Condition Register.

Program Control

Two types of instructions are associated with bit processor operations. The first (shown below) specifies a displacement (bits 7-0) to be added to the base address (V Register contents) to specify the location to be accessed. Bits 12-8 (binarily coded) are used to select a specific bit to be used in the operation. The Op Code is defined in bits 23-13.



The second word format is used for bit movement or transfers where a specific bit from memory is not required. Bits 23-12 contain the Op Code; the remaining bits are undefined.



Bit Processor Instruction Set

The bit processor (Boolean function) group of instructions include branches, logical manipulation, and interrogation of a specified bit selected from an effective memory address or the H Register. The following instructions are included in the bit processor group.

Dot Memory with H (DMH)
Dot Not (memory) with H (DNH)
Flag Bit of Memory (FBM)
Negate of H to H (NHH)
Or Memory with H (OMH)
Or Not (Memory) with H (ONH)
Query bit of H (QBH)
Query bit of Memory (QBM)
Transfer Flag to H (TFH)
Transfer H to Memory (THM)
Transfer K to V (TKV)
Transfer Memory to H (TMH)
Transfer V to K (TVK)
Transfer Zero to H (TZH)
eXclusive-or Memory with H (XMH)
eXclusive-or Not (memory) with H (XNH)
Zero Bit of Memory (ZBM)

PROGRAM RESTRICT AND INSTRUCTION TRAP

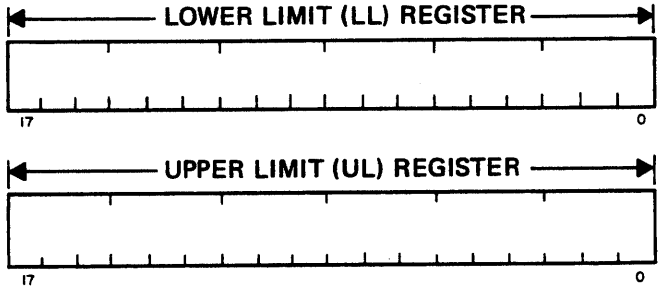
General Description

The Program Restrict and Instruction Trap option allows areas of memory to be selected under program control and protected from unauthorized access. The instruction trap provides a means for preventing the execution of a predetermined group of instructions. Two registers, two executive trap interrupt trigger circuits, and the associated control logic comprise this option. Control is maintained by two instructions; Transfer Double Register to Limit Register (TDL), and Transfer Limit Register to Double Register (TLD).

Program Restrict Registers

Two 18-bit registers are provided with the Program Restrict and Instruction Trap. These registers define the upper (UL Register) and lower (LL Register) limits of a memory area that is authorized access without restrictions. Both registers are programmable and may be entered and displayed via the Programmer's Control Panel.

PROGRAM RESTRICT REGISTERS



Operational Description

The CPU operates with the program restrict system enabled or disabled depending on the position of the PROG REST/OFF/VM key switch. When the restrict system is disabled, all memory is accessible.

When the restrict system is enabled, the computer operates in three distinct modes as established under program control. The three program restrict modes are defined below.

a. Unrestricted (Mode 0) — Programs working in this mode may access and alter any location in memory.

b. Restricted/Privileged (Mode 1) — Programs operating in this mode may access and load from any location in memory; however, Mode 1 programs may not alter the contents of, or transfer control to, any memory location outside the specified limits.

c. Restricted/Unprivileged (Mode 2) — Programs operating in this mode may not reference, in any manner, any memory location outside the specified limits.

Once established, the restrict mode is maintained by two flags operating concurrently. The mode flags can be set to one-of-three significant states to establish Mode 0, 1, or 2.

Control over the restrict system is maintained by the program restrict flag (PRF). The PRF operates under the condition and in the manner outlined below.

a. The PRF may be set only when the restrict system is enabled (i.e., when the PROG REST/OFF/VM key switch is in the PROG REST position).

b. When in Mode 1 or 2, the PRF is set when an instruction transfers control into the restricted area of memory.

c. When MASTER CLEAR is depressed, the PRF is set and Mode 2 is established automatically. MASTER CLEAR also clears the limit registers. Once a violation is made, the only way of recovering manually is by placing the PROG REST/OFF/VM key switch to the OFF position and then depressing the MASTER CLEAR switch.

d. Priority interrupts reset (turn off) the PRF, rendering the mode flags ineffective until control is returned to the restricted area and the PRF is set again.

e. When the PRF is set, executive trap interrupt 2 (Group 0, Level 2) occurs if a restrict violation takes place, except when operating in the Halt Mode. (A restrict violation consists of any attempt to violate the conditions established by the modes.) The one exception to this is the Branch and Link Unrestricted (BLU) instruction. The BLU instruction has been implemented as an executive call to allow restricted programs to communicate directly with the resident operating system without being trapped. The BLU instruction resets the PRF and transfers control unconditionally to the address specified by the instruction.

Program Control

The restricted area of memory is defined by two special registers, the Lower Limit (LL) and Upper Limit (UL) Registers. Each register retains an 18-bit address that defines one limit of the restricted area.

Two instructions, Transfer Double to Limit Registers (TDL) and Transfer Limit Registers to Double (TLD), are provided for operating the limit registers. The limits are defined by executing a TDL instruction where D = E and A; bits E17-E0 specify the lower limit and A17-A0 specify the upper limit. The TDL instruction also establishes the restrict mode by setting the mode flags with Bits A21 and A22. The bit configuration determines which mode will be established as shown below.

A22	A21	Mode
0	0	0
0	1	1
1	0	2
1	1	0

NOTE

If an attempt is made to execute the TDL instruction while in Mode 1 or 2, the instruction trap is activated.

The TLD instruction provides a method of saving the contents of the limit registers plus the status of the mode flags. The contents of the LL Register are transferred to bits E17-E0 and the contents of the UL Register are transferred to A17-A0. The mode flag bit configuration is retained in bits A22 and A21.

Instruction Trap

The instruction trap is enabled and disabled by the PROG REST/OFF/VM key switch. When the PRF is off, the instruction trap is inhibited.

If an attempt is made to execute any of the instructions listed below with the PRF on, an interrupt occurs at Group 0, Level 3. The interrupt routine may then examine the trapped instruction and determine what action is to be taken. The affected instructions are:

Halt (HLT)
Output Data Word (ODW)
Input Data Word (IDW)
Output Command Word (OCW)
Input Status Word (ISW)
Output Address Word (OAW)
Input Address Word (IAW)
Input Parameter Word (IPW)
Hold eXternal Interrupts (HXI)
Release eXternal Interrupts (RXI)
Unitarily Arm group 1 interrupts (UA1)
Unitarily Disarm group 1 interrupts (UD1)
Unitarily Enable group 1 interrupts (UE1)
Unitarily Inhibit group 1 interrupts (UI1)
Transfer Double to group 1 (TD1)
Transfer Double to group 1 (TD4)
Transfer Double to Limit Registers (TDL)

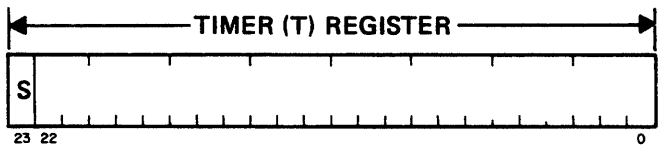
INTERVAL TIMER

General Description

The programmable interval timer consists of a 24-bit register (T Register), a clock, and associated control logic. The timer can be preset and subsequently released, under program control, to measure elapsed processor (CPU) time or clock (real) time.

Timer Register

Supplied with the interval timer, the 24-bit Timer (T) Register operates as a counter in two distinct modes of operation. When not used for timing functions, the T Register functions as an additional general-purpose register that can be accessed through the instruction set when operating in the Monitor Mode. Entry and display for the T Register is provided via the Programmer's Control Panel.



Operational Description

A self-contained clock generates the 1 microsecond pulses used to strobe the timer. In either mode of operation, a negative count is loaded into the T Register and is decremented once for each elapsed period of 1 microsecond. When the count reaches zero, an executive trap interrupt is generated at Group 0, Level 5. A maximum count of $16,777,215_{10}$ (77777777_8) may be loaded into the register. With a resolution of 1 microsecond per count, a maximum time interval of 16.777215 seconds is available.

Program Control

Interval timer operation is controlled by three instructions: Hold Interval Timer (HIT); Release Processor Time (RPT); or Release Clock Time (RCT). A HIT instruction will prohibit the start of any timing sequence or halt any in-process timing operation until the timer is released by a RPT or RCT instruction. The RPT instruction releases the timer for measuring elapsed processor (CPU) time. In this mode, counting is inhibited during block I/O channel DMA operations, whenever any interrupt is active or the CPU is halted. Clock (real) time operation, where the timer counts continuously regardless of CPU condition, is initiated by an RCT instruction.

STALL ALARM

The stall alarm is enabled and disabled by the OFF/CP.LK.ST.AL. key switch on the control panel. When the stall alarm is disabled, normal CPU operations take place. Once the stall alarm is enabled, a 128-cycle counter is activated whenever certain instructions are executed or certain operating conditions are encountered. The counter is incremented once each CPU cycle until the specified instruction(s) or conditions are removed. If the instructions/conditions are still present after 128 machine cycles, an executive trap interrupt is generated at Level 4 of Group 0.

The following instructions and/or CPU conditions will activate the stall alarm counter.

Unitarily Arm group 1 interrupts (UA1)
Unitarily Disarm group 1 interrupts (UD1)
Unitarily Enable group 1 interrupts (UE1)
Unitarily Inhibit group 1 interrupts (UI1)
Transfer Double to group 1 interrupts (TD1)
Transfer Double to group 1 interrupts (TD4)
Update Stack Pointer (USP)
Branch and Save return – Long (BSL)
Hold eXternal Interrupts (HXI)
Hold interrupts and Transfer I to memory (HTI)
Hold interrupts and Transfer to J memory (HTJ)
Hold interrupts and Transfer K to memory (HTK)
EXecute Memory (EXM)
Release eXternal Interrupts (RXI)
Transfer Registers to Memory (TRM)
Transfer Memory to Registers (TMR)
Branch and Reset Interrupt Long (BRL)
A halt condition
An indirect memory cycle

Each of the preceding instructions or conditions prohibit the recognition of external interrupts for a period of one cycle following completion of the instruction. Executing a series of these instructions sequentially will lock out external interrupts for the entire series. Multilevel indirect addressing can produce a similar effect, since the instruction must satisfy all address references before completion. A halt condition – whether as a result of

programmed halt, operator action, or memory parity error — also prohibits external interrupt recognition by the CPU.

If a power failure occurs, the stall alarm becomes disabled. However, when power is restored, the stall alarm is re-enabled and operations continue in a normal routine.

With the exception of an EXM instruction or an indirect cycle, the monitored operation is allowed to complete its sequence before the executive trap assumes control. An EXM chain (where an EXM instruction references another EXM which, in turn, specifies a third, etc.) has the same overall effect as an indirect chain in that all references must be completed before the sequence is complete. Therefore, if an EXM or indirect cycle is in process when the executive trap is generated, the stall alarm logic automatically terminates the sequence. If a block controller channel is transferring data into memory when the executive trap interrupt is generated, the current cycle is completed before termination occurs and the trap takes control. If a halt condition is in effect when the executive trap interrupt is generated, the stall alarm logic automatically forces the CPU into a run mode.

120 HERTZ CLOCK

This clock continuously transmits 120 or 100 mainframe interrupt signals per second, depending on power line frequency. The interrupt signal is controlled completely by enabling (or disabling) the assigned CPU interrupt level. The first interrupt following an enable signal will occur in less than 1/120 (1/100) of a second because the clock never stops transmitting signals; however, all subsequent interrupts will be precisely 1/120 (1/100) seconds apart.

The accuracy in using this clock is a function of the user interrupt routine logic. For example, if the clock is used to update a "time-in-seconds" counter by adding one count every 120 (100) interrupts, the "current time" at any given query will be accurate within 1 second. If, however, the counter is updated each interrupt — 1/120 (1/100) — and divided by 120 (100) when "current time" is queried, the accuracy will be within 1/120 (1/100) of 1 second.

A simple example of coding, where the clock is assigned to priority interrupt Group 1, Level 22, is as follows:

```
*  . . . . . .                    . Initialize Clock Routine
INITCT      TMA = B22              . (A) = Bit 22
            TME = B22              . (E) = Bit 22
            UA1                    . Arm L22/G1
            UE1                    . Enable L22/G1
            TZM CLOCK T            . Zero Clock Time
            BUC 0, J

*  . . . . . .                    . Interrupt Routine
CLOCK IR    ***                    . Enter
            AUM CLOCK T            . Increment Clock Time
            BRL* CLOCK IR          . Restore C register and Exit

*  . . . . . .                    . Current Time Routine
CTIME       TMA CLOCK T
            ESA
            DVO 120
            BUC 0, J
                                   . Return: (A) = Seconds
                                             (E) = Remainder
```

FIRMWARE BOOTSTRAP

The firmware bootstrap automatically stores in memory a loader program that permits a more complex program to be stored. Any program can be loaded as long as it is in bootstrap format; however, the most common application is to load a loader program which allows other programs, operating systems, diagnostics, or other data to be stored in selected memory locations. Eight sources are selectable for transferring a program to memory via a selected peripheral device: paper tape; cassette (paper tape emulation); disc; card reader (word mode); card reader (block mode); magnetic tape; and flexible diskette. The operation of the bootstrap is implemented at the control panel. The specific device may be selected with the BOOTSTRAP SELECT switches and stored in memory by depressing the BOOTSTRAP ENA switch. A description of the bootstrap operation and individual bootstrap programs are documented in the Operator's Manual, 0840003.

POWER FAIL SHUTDOWN AND RESTART

This feature provides the capability of saving the operating program in the event of a power failure and provides program restoration and restart when power levels return to normal. This feature is applicable to core memories or to semiconductor memories with battery back-up. It is not applicable to semiconductor memories without battery back-up.

The shutdown circuits monitor the input ac power source for amplitude fluctuations. A decrease in ac voltage below the specified level causes an executive trap interrupt to be generated at Group 0, Level 0. If semiconductor memory with battery back-up is installed in the computer, the memory will be switched to battery. One millisecond after the interrupt, a Master Clear signal is generated to complete the shutdown process. In the one millisecond interval between interrupt and final shutdown, the interrupt-processing routine must save the operating program along with parameters for returning to the point of interrupt.

When the ac power level returns to its nominal level, a restart signal is generated to begin the restore process. The restart signal generates an executive trap interrupt at Group 0, Level 1.

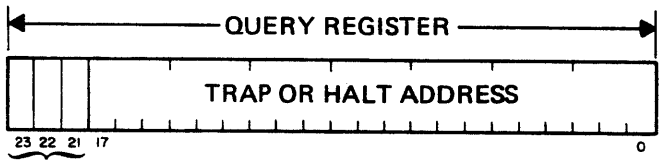
PROGRAM HALT AND ADDRESS TRAP

General Description

This feature provides address trap or program halt functions as desired by the user. Memory reference addresses are compared to a preset address. A comparison between the reference and preset address causes an executive trap interrupt to be generated or a program halt to occur depending on the state of mode control bits. Hardware includes a register, an 18-bit comparator, an interrupt trigger circuit, and associated control logic.

Query Register

A 21-bit address Query Register is supplied with the program halt and address trap. Bits position 17 through 0 contain either the trap address or the program halt address. Bits 23 through 21 are the halt or address trap control bits. When an address is reached in program that coincides with the address stored in the Query Register, the machine halts or an interrupt is generated. The Query Register may be loaded under program control or via the Programmer's Control Panel.



Operational Description

The Query Register is loaded with the Transfer Memory to Query Register (TMQ) instruction. This instruction transfers the contents of the selected memory location to the Query Register. The 18 least-significant bits, representing the trap or halt address of the memory word, are loaded into bit positions 17-0 of the Query Register. Memory word bits 20-18 are not used and bits 23-21 of the memory word are loaded into bit positions 23 through 21 of the Query Register. These three bits determine the mode of operation to be performed and have functions as follows:

Bit 23 = ONE Disable Address Trap
Bit 23 = ZERO Enable Address Trap

Bit 22 = ONE Trap on Write only
Bit 22 = ZERO Trap each time selected address is referenced

Bit 21 = ONE Trap or Halt during User Mode only
Bit 21 = ZERO Trap or Halt during Monitor Mode only

When the Program Halt Enable switch on the Programmer's Control Panel is enabled (PH ENA in the up position), the contents of the Query Register are compared with the program address. When they compare, the machine is halted. If virtual memory is enabled and the address trap disabled (bit 23 = ONE), a compare will cause the machine to halt in the User Mode if bit 21 is set. If bit 21 is reset (ZERO), the machine halts in the Monitor Mode.

When the PH ENA switch on the control panel is disabled, the address trap is enabled or disabled with bit 23 of the Query Register. The address trap is enabled when bit 23 is reset, or ZERO. Each time a referenced memory address corresponds with the address stored in the Query Register, an executive trap interrupt at Group 0, Level 7 is generated to inform the CPU. When bit 23 is set (ONE), the address trap is disabled. Disabling the trap inhibits the executive trap interrupt.

Additional control of the address trap is provided with bits 22 and 21. With the trap enabled and bit 22 set (ONE), the executive trap interrupt is generated when a write operation is made to the referenced location. If bit 22 is reset (ZERO), the interrupt is triggered whenever the referenced location is accessed. With the virtual memory enabled and bit 21 set (ONE), the address trap is enabled during User Mode of operation; if bit 21 is reset, the trap is enabled during the Monitor Mode. The memory address is taken from the CPU at a point prior to the address translation when virtual memory is enabled.

Memory addresses that result from DMA operations by block controller channels are not affected by the address trap.

Program Control

With the Query Register loaded and the address trap enabled, an interrupt is generated (in accordance with the control bit settings) each time a reference is made to the memory location corresponding to the address stored in the Query Register. If it is desired that a reference to the selected memory location be recognized only once, a second TMQ instruction should be executed following the first interrupt to set bit 23 of the Query Register to a ONE. This disables the address trap.

When the trap occurs, the instruction causing the trap is terminated and execution of that instruction is not completed.

REAL TIME CLOCK

General Description

The Real Time Clock option consists of a 100 kHz crystal-controlled clock, a counter, and associated control logic. All components are mounted on a board which is designed to plug into the internal controller locations of the Programmed Input Output Channel (PIOC) board. Each PIOC can accommodate one or two Real Time Clocks. Although this option has no peripheral device connected to it, programming is accomplished via normal I/O instructions. More than two Real Time Clocks may be used; the limiting factor being the number of PIOCs used in the system. An external interrupt is provided which is configured in the same manner as any input/output interrupt, i.e., the interrupt can be assigned to any level in Group 1 except 0 and 1. The interrupt is generated when the clock count reaches ZERO and the interrupt is enabled.

Operational Description

By means of the Real Time Clock, the programmer is provided with an interval timer which operates independent of CPU timing and provides output pulses when the CPU is either in the Run or Halt condition. Elapsed time is measured by counting down the pulses in the counter. A

selected time interval is preset in the counter by loading up to three, 8-bit bytes into the counter. Clock output pulses occur at 10 microsecond intervals. A maximum time period of 167 seconds is available when the counter is loaded with all bits set in the three bytes. Thus, the programmer can preset the clock for time intervals from 10 microseconds to 167 seconds in 10 microsecond increments. Since the Real Time Clock is asynchronous with CPU timing, the period may be off by 10 microseconds on the first count-down cycle.

Command and Status Word Formats

As a result of the CPU issuing an Output Command Word (OCW) instruction, a command word is transferred from the A Register to the Real Time Clock. The command word initiates operation of the clock, and provides the necessary set-up and control functions. A description of the function performed by each bit of the command word is given below.

7	6	5	4	3	2	1	0
Run/ Hold	Load Preset Count	Enable Snapshot	Enable Bits 0-3	Byte Count 2^1	Byte Count 2^0	Enable Auto Restart	Enable Interrupt

Bit 0 (1) Enable count zero interrupt
(0) Disable count zero interrupt

Bit 1 (1) Enable Automatic Restart of preset count
(0) Go into hold mode at count of zero

Bit 2, 3 Byte count for input and output

Bit 4 (1) Sample bits 3-0
(0) Hold bits 3-0 unchanged

Bit 5 (1) Enable count snapshot output
(0) No action

Bit 6 (1) Enable loading of preset count
(0) No action

Bit 7 (1) Enable count down
(0) Hold count down

An Input Status Word (ISW) instruction generated by the CPU results in the status word being transferred from the Real Time Clock to the A Register. The clock status word consists of bit 0 only. It is set to the ONE state whenever the clock module is plugged into the PIOC board, indicating to the CPU that it is on-line.

Program Control

Real Time Clock operation is controlled with four instructions: Output Command Word (OCW), Input Status Word (ISW), Output Data Word (ODW), and Input Data Word (IDW). Each Real Time Clock is addressed by a channel-unit code combination in the same manner as any I/O device. If one Real Time Clock is installed, a unit code of 00, 01, or 02 is assigned according to its plug-in location. If two Real Time Clocks are installed, unit codes of 00 and 02 are assigned. Access to the clock is via the A Register as in normal I/O operation.

Preset Count Loading

To initialize the Real Time Clock, an OCW instruction is generated by the CPU to transfer the command word with bit 6 = 1, and the desired byte count in bits 2 and 3. The CPU then provides the specified number of ODW instructions (one per byte) to transfer the bytes to the clock, with the most-significant byte transferred first. When the byte count is satisfied, an OCW instruction may be given to transfer a command word with bit 7 =1. This enables the counter to start counting down. If bit 7 =0 in any command word, counting is inhibited until a command word with bit 7 =1 is received. If a byte count less than three is specified, the unused bytes in the counter are set to ZERO.

Automatic Count Restart

If bit 1 = 0 in the command word, the automatic count restart is enabled. This causes the Real Time Clock to automatically reload the last preset count into the counter and restart the count after the interrupt is given.

Snapshot Output

During Real Time Clock operation, the current count status is made available to the CPU by means of the Snapshot mode of operation. Snapshot output is initiated with an OCW instruction and bit 5 = 1 in the command word. This loads the 24 bit current count into a register. IDW instructions, one per byte, transfer the contents of the register to the CPU, the most-significant byte being transferred first. This operation does not affect the counting as long as bit 7 = 1 in the command word. If an interrupt is generated during the Snapshot mode of operation, the mode is terminated as the count is known to be zero.

If snapshots are performed in a program with automatic count restart selected, snapshot time prior to automatic restart may be 10 microseconds different from snapshot time after automatic restart. This is because of the 10 microsecond time frame used in the Real Time Clock. Additionally, if a snapshot is performed at the trailing end of a time out, before restarting or auto-restarting, the snapshot bytes may be all zeroes. To minimize the possibility of the foregoing occurrences, the snapshot of any time must be accomplished in the least machine time possible. An example of programming code that may be used to do a snapshot in the shortest period of machine time follows.

```
          ....
SNSH      DAC       *
          TRM       SAVE      Save contents of register
          TOA       '240      Run, Snapshot command
          OCW       C/U       Output command
          IDW       C/U       Input most-significant byte
          BNZ       *-1       Possible wait
          TAI                 Store most-significant byte in I register
          IDW       C/U       Input middle byte
          BNZ       *-1       Possible wait (needed if other units on channel)
          TAJ                 Store middle byte in J register
          IDW       C/U       Input least-significant byte
          BNZ       *-1       Possible wait (needed if other units on channel)
          TAK                 Store least-significant byte in K register
          TIA                 Restore most-significant byte in A register
          LLA       8         Shift over 8 bits
          TJB                 OR in middle byte into A register
          LLA       8         Shift over 8 bits
          TKB                 OR in least-significant byte into A register
          TAM       TIME      Store whole word of time for later use
          TMR       SAVE      Restore registers
          BUC*      SNSH      Exit
SAVE      BLOK      5         Register save area
TIME      DATA      0         Register save area
          ......
```

Selection Sampling

Selection Sampling is included as a feature of the Real Time Clock for the convenience of the programmer. Since the programmer would normally want to keep command word bits 3-0 constant while he uses bits 7-5, bits 3-0 are sampled only when command word bit 4 = 1.

SECTION III
MEMORY SYSTEM

MEMORY SYSTEM DESCRIPTION

Introduction

Storage for the instruction and data words is provided for by the main memory system. Memory modules are available with either semiconductor or magnetic core storage elements. Main memory may consist of all semiconductor modules, all core modules, or a mixture of semiconductor and core. System memory capacity from 48K to 256K is available, where each word is 24 bits wide. Error detection and correction circuits are available with semiconductor modules. An optional data save unit is also available for semiconductor memory modules to retain information in the event of a facility power loss.

Data Transfers

Data transfers are over a 48-bit, asynchronous, bidirectional system data bus. Other buses provided include an 18-bit system address bus and a system control bus. All functional elements in the computer system communicate with each other through the system buses. The asynchronous bus system allows each system element to function at its own rate, independently of the other system elements. For example, concurrent direct memory access I/O transfer, CPU instruction execution and SAU double-precision floating point operation. All buses are located on the backplane which is common to all boards in the system. This interconnection scheme eliminates the need for discrete wiring between the various boards in the system.

Transfer of data between the CPU and memory is over 24 of the data bus lines. CPU and Programmed Input Output Channel (PIOC) data transfers use 8 of the data bus lines. Data transfers between memory and the Integral Block Channel (IBC), External Block Channel (XBC), and Direct Memory Access Communication Processor (DMACP) is via 24 data bus lines. Universal Block Channel (UBC) data transfers to and from memory occur on all 48 lines. All block I/O channels, once initialized, can perform blocked data transfers between memory and the peripheral device without CPU intervention.

SEMICONDUCTOR MEMORY MODULES

General Description

The basic storage element of the semiconductor memory module is an N-channel metal oxide semiconductor (MOS), 4K by 1-bit random access memory (RAM). A dynamic device, the RAM requires a periodic rewrite or refresh cycle to retain the stored data. It is also volatile – its data content is lost when power is removed from the device. As in all semiconductor memories, the RAM has a non-destructive readout as opposed to a magnetic core memory which has a destructive readout. In addition to the RAM storage elements, each semiconductor memory module contains an address register, a memory data register, timing and control circuits, and, data error correction circuits.

Each memory board is configured as a 16K word by 29-bit memory module which can also be operated as an 8K by 58-bit memory module to provide a double-word fetch. Minimum memory size is 48K words which can be expanded to 256K words in 16K increments.

Semiconductor memory has a cycle time of 450 nanoseconds and an access time of 300 nanoseconds.

Operating Modes

Operating modes of the semiconductor memory modules include the Read Mode, Write Mode, and Power Fail Refresh Mode. In either Read or Write Mode, a double word of 48 bits or a single word of 24 bits may be selected. In the Power Fail Refresh Mode of operation, memory operations are discontinued but data stored in memory is saved until normal power is restored if the data save unit is installed.

On a write to memory operation, data on the system data bus is loaded into the memory data register. Data is then transferred from the register to the location in memory specified by the address bits on the system address bus.

A read operation causes data in the location specified by the address on the system address bus to be transferred from memory to the memory data register. A single- or double-word transfer is then made from the data register to the system data bus.

When input power to the computer falls below specified levels, a power fail safe signal is transmitted to the memory modules. This signal inhibits any additional memory cycles but allows completion of the current memory cycle. If the optional battery backup is not installed in the system, data stored in the RAMs is lost. With the battery backup installed, the power fail safe signal causes memory to go into the Power Fail Refresh Mode of operation. In this mode of operation, data stored in the RAMs is periodically renewed or restored by a refresh only circuit. By this means, data is not lost as a result of an ac power failure but is saved for a period of up to two hours or until normal power is restored.

Error Correction

Single-bit error correction is provided by error correction circuits contained on each semiconductor memory board. All one-bit errors and some two-bit errors are corrected by this circuit. An external interrupt is generated and the parity error (PE) indicator on the control panel is lighted whenever a parity error is detected, whether it is corrected or not. Detection of a parity error does not halt the machine.

All write operations to the memory will store either 24 or 48 bits of data and 5 bits of Hamming Code parity for each 24-bit data word. These parity bits are generated by the memory module whenever data is asserted for a write operation. All read operations of memory regenerate the Hamming Code from the stored data bits and compare this with the stored Hamming Code. This comparison generates an address code that points to the bit in error, and the correction circuit corrects the error. The corrected data is stored in the memory data register and is written into the appropriate memory location. This corrected data may then be obtained from the memory data register by a read operation to the same address, with no other operation intervening.

MAGNETIC CORE MEMORY MODULE

General Description

The core memory module stores data in magnetic cores configured in a single planar array. A magnetic core is a non-volatile device, therefore, data is not lost when power is interrupted. A core memory module also contains address registers, an address comparator, data registers, parity generators, and parity checkers.

A core memory module is operated as a 32K-word by 25-bit memory or as a 16K-word by 25-bit memory, where each word is comprised of 24 data bits and 1 parity bit.

Core memory has a cycle time of 500 nanoseconds and a normal access time of 240 nanoseconds.

Operating Modes

Single or double words may be placed on the data bus during a read operation. If a single word read operation is specified, two words (even and odd addresses) are retrieved from core and are loaded into the memory data register. Then, according to the address, the even or odd word is gated to the bus. If the addressed word contains an error, the parity error signal is asserted. A double-word read operation places the addressed words into the data register and onto the data bus. If an error is detected in either word, the parity error signal is asserted.

Single or double words may be stored in memory during a write operation. For a single-word write operation, two words (even and odd) are accessed from storage. The 24-bit word at the addressed location is cleared. The other word is placed in the data register along with the 24-bit word which is to be written into the addressed location. Then a parity bit is generated for the new word and both words are written into memory. In a double-word write operation, both the odd and even single words on the data bus are loaded into the data register. An odd parity bit is generated for each word, and then a clear write operation is performed to store the two single words in the addressed location.

If input power is interrupted, a power fail safe signal allows completion of the current cycle but prohibits any additional cycles. When a parity error is detected, an external interrupt is generated and the PE indicator on the control panel is lighted, but the machine is not halted.

Fast Access Operation

A memory module always operates on two, 24-bit words at a time. These two words have the same address, except for the least significant address bit which defines the "even word" or the "odd word". If the specified word is at location 00 (even word), for example, the words at locations 00 and 01 are accessed simultaneously. If location 01 contains the specified word, the same two locations are accessed.

Each memory module has a 48-bit data register, termed a Content Addressable Buffer (CAB), to improve system performance by reducing the effective cycle time of the computer. Each memory access fetches and loads the two, 24-bit words into the CAB. When the CPU requests a word from memory, a memory access is performed and the word is transferred over the system data bus to the CPU. However, if the CPU requires the next sequential word, it is transferred from the CAB to the CPU without requiring a second memory access. The CAB significantly reduces the fetch and execute time for sequential instructions.

Fast access operation makes use of the Memory Data Register (CAB), an address register, and comparison logic to reduce the effective cycle time of the computer. The address register retains the address of the last 24-bit word memory location accessed. A new address to memory is compared to the address stored in the address register. If the new address is not equal to the previous address stored in the address register, and if memory is not busy, a normal memory cycle occurs. If the new address is equal to the previous address, and memory is not busy, the data word is gated from the Memory Data Register to the system data bus immediately. In this case no memory cycle occurs and no parity or data correction time is lost since these tasks were done during the previous access.

ERROR REPORTING

Memory errors are reported to the system by means of the external priority interrupt structure. When semiconductor memory corrects an error, the parity error signal generated is termed a "soft" parity error. If the error is not corrected by the error correction circuits, the parity error signal is referred to as a "hard" parity error. Since magnetic core memory contains no error correction circuits, they generate only hard parity errors. Each time a hard parity error occurs, a Group 1, Level 0 interrupt is generated. For each soft parity error generated, the Group 1, Level 1 interrupt is triggered. A count of the number of hard and soft interrupts is recorded by software. The operating system then responds according to the type and number of errors recorded.

INTERLEAVING

Interleaving between memory modules is available in either two-way or four-way configurations, and up to sixteen modules can be interleaved. Interleaving can provide an improvement in system performance, but such improvement is entirely dependent on the nature of the program being executed.

SECTION IV
INPUT/OUTPUT CHANNELS

GENERAL DESCRIPTION

The input/output (I/O) structure of the computer system combines the characteristic economy of unit I/O systems with the speed of a channel I/O system. This configuration, in conjunction with the I/O instructions, permits maximum flexibility in I/O communications. The relationship between the CPU and the I/O structure is illustrated in Figure 4-1. The elements comprising the I/O structure are described in the following paragraphs.

The basic I/O structure allows single word data transfers between the Central Processing Unit (CPU) and a peripheral unit. It also allows I/O command and test operations to be program controlled. Block I/O channels may be used to control the transfer of blocks of data between the CPU and the peripheral units without program intervention.

The I/O structure involves communication (such as data transfers, addresses, and command status information) between the CPU and a peripheral unit by way of a channel. The CPU communicates with a specific channel and the channel, in turn, communicates with a peripheral unit. The I/O structure varies with CPU configurations to accommodate an applicable number of input/output channel (IOC) boards, all of which can be active concurrently. A channel can communicate with from one to sixteen peripheral units using standard I/O instructions. Only one peripheral unit per channel can be connected; however, all units can be active at any given time.

Communications between the I/O structure and the CPU may also be conducted on an interrupt basis. Logic in the channel and unit allows unit interrupts to be placed under program control and selectively enabled or disabled by executing the appropriate I/O instruction. An alternate method permits unit functions to be wired directly to the CPU priority interrupt structure and used as interrupt triggers.

The I/O interface is the link between each peripheral unit and its channel. The interface and its associated unit control facilities provide the physical means for connecting the peripheral device to the I/O structure and the logic capability that allows the unit to adapt the standard I/O controls to its specific requirements. The interface facilities and unit control logic are normally integrated with the peripheral unit. However, some controllers are available as options to the Integral Block Channel (IBC) and 8-bit Programmed Input/Output Channel (PIOC) boards.

BASIC I/O CONCEPTS

The I/O structure implements basic concepts to perform input/output operations between the CPU and a variety of channels and units. These basic concepts and their applicability are described in the following paragraphs.

Addressing

a. Channel Addresses — The I/O channels must each be addressed via a unique address contained in each I/O instruction. A channel is patched, or switched, to recognize its assigned address. The recognition of this code in an I/O instruction activates channel logic to execute the instruction. No other channel will respond.

b. Unit Addresses — Since a channel is capable of communicating with one or more unit controllers, any instructions involving the transfer of data, commands, or status must necessarily contain an address applicable to the unit involved. The unit address is contained in the format of the following instructions (reference Section VII for formats).

Output Data Word (ODW) — PIOC, IBC, UBC, XBC and DMACP

Output Data Word (ODW) — PIOC, UBC, XBC and DMACP

Input Data Word (IDW) — PIOC, UBC and DMACP

Input Status Word (ISW) — PIOC, IBC, UBC, XBC and DMACP

Output Address Word (OAW) — IBC, UBC, XBC and DMACP

Input Address Word (IAW) — IBC, UBC, and DMACP

Input Parameter Word (IPW) — IBC, UBC, and DMACP

NOTE

The inclusion of unit addresses in the IBC channel OAW, IAW, and IPW instructions has no transfer-to-unit control. The IBC channel contains the capability to concurrently store block transfer parameters for all unit controllers on its interface and the parameters must be addressed to reserved storage areas.

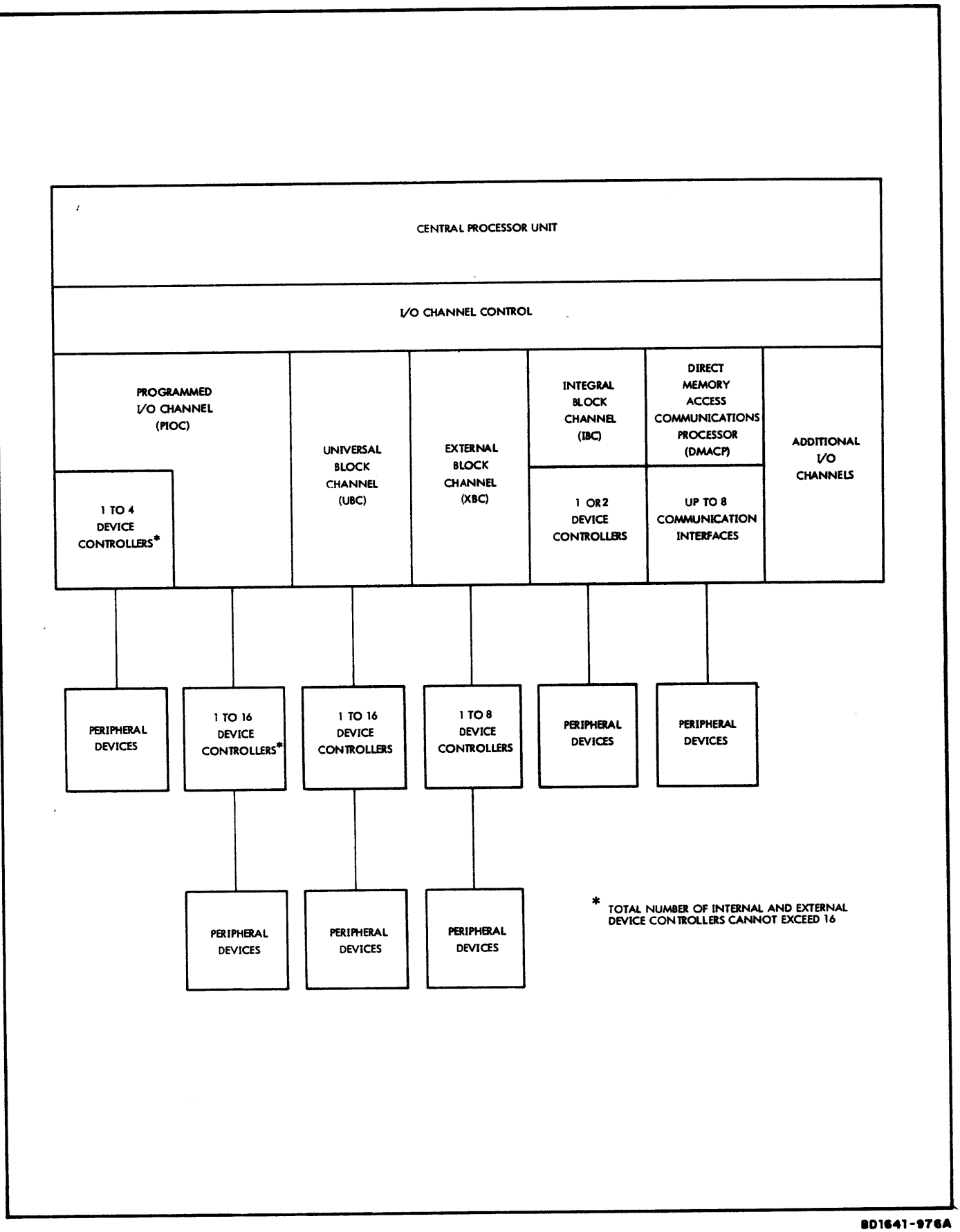


Figure 4-1. Computer I/O Structure Block Diagram

An instruction containing a unit address, sent to any channel other than the IBC channel is compared to the unit code of the previous instruction. If a non-compare is detected, the channel does not execute the ISW or IDW instruction. Instead, a disconnect/connect sequence is entered in order to connect the addressed unit. A non-compare detected during OCW and ODW instructions forces the disconnect/connect sequence also, but the channel loads the data/command, if not previously set busy, and holds the data/command until the addressed unit is "connected" to its interface. The transfer is then completed and the channel returns to a "not busy" condition.

Disconnect/Connect Sequences

Each IOC performs disconnect/connect sequences if the unit address contained in the instruction differs from the previously loaded address. In disconnect/connect sequences occurring during input instructions, the channel is prevented from setting the "ready" line to the CPU to verify that the instruction was executed. This requires a Branch on Not Zero (BNZ) instruction execution after each I/O instruction for a repetition of nonexecuted instructions. Timeout routines sequenced by the CPU may then detect channel/unit hangups and execute Input Status Word (ISW) instructions to pinpoint conditions.

The IBC channel is not equipped to sequence disconnect/connect operations; in this channel the unit is automatically connected to the channel for the purpose of instruction execution except during the time that data transfers are taking place.

Block I/O Channel Priority

A programmable matrix is contained on each IOC capable of performing block transfer operations. The matrix is provided to resolve contention for simultaneous memory cycle requests. The block I/O channels are assigned priority levels and the highest priority channel requesting a memory cycle inhibits any lower priority channel(s) from sensing a "memory cycle granted" signal from the CPU. A system should be configured to assign high speed devices a lower priority level than relatively lower speed devices. Also, no unused priority levels should appear between any two channel levels. The priority matrix is patched on UBC and XBC channels, and is switch-selectable on the DMACP and IBC channels.

Synchronization (Handshake) Conditions

With few exceptions, all data and command sequences are synchronized via "handshake" operations. This convention ensures that the connected unit has received the command or data in output transfers or frees the unit to load new words in input transfers. If the unit is unable to accept the command/data, the channel sets itself busy and will honor no output transfer operation except for the OCW instruction in which "Override" is specified. The normal handshake function is modified in XBC and IBC channel operations and is described following the conventional handshake functions.

Output Transfer Synchronization

The output transfer handshakes are performed in OCW/ODW single-word transfer operations and in output block transfers of block I/O channels. In single-word transfers, if the channel is not busy executing a previous output instruction, the command/data is loaded into the channel's output buffer and the "Output Command Here" or "Output Data Here" line is raised to the unit. The channel sets itself busy to inhibit any new output transfer operations. When the unit gates the command/data into its own registers, it returns an "Accepted" signal. This signal resets the channel busy condition and the channel is free for a new transfer.

In block transfer sequences, the channel, having been previously initiated for output transfer operations, automatically sequences memory request operations. When the memory cycle is granted, the channel places the transfer address on line and loads the word from the specified address. The channel then raises the data transfer handshake line and, when the unit "accepts" the data, fetches another word from memory. The channel remains "busy" and the sequences continue until the transfer is completed or overridden.

Input Transfer Synchronization

A channel cannot execute an IDW instruction until it senses that the "Data Available" line from the unit has been set true. In normal operations the channel automatically transfers the input to the CPU and raises the "Data Accepted" handshake line. The unit drops "Data Available" to prepare a new word for transfer.

An input block transfer begins when a unit raises its "Data Available" line after the channel and unit have been commanded to the input mode. The channel loads the data into its input buffer and raises its "Data Accepted" line to the unit. The channel then sequences a memory cycle with the CPU to store the input word at the address specified by the Transfer Address Register (TAR). The channel will not honor any subsequent store requests until the memory cycle has been completed.

XBC Channel Synchronization

The block transfer sequence control is under the control of the external units in XBC applications. The unit may be commanded to the block-transfer mode via an OCW instruction and may require parameter inputs but, once initiated, the device controls the transfers. In executing the OCW instruction the channel uses the conventional

"Command Data Here" handshake signal and the unit returns "Accepted" to signal loading of the command. If required by the unit, the channel executes an OAW instruction to provide the Transfer Address (TA) to the unit. The channel raises "Address Word Here" which signals the unit to "accept" the address. This may be followed by an ODW instruction in which the word count is sent to the unit. The channel raises "Word Count Here" which the unit "accepts".

Data transfers to/from memory begin when the unit sets a priority-structured "Data Transfer Request" line to the channel. If the channel is not busy executing an instruction or servicing a higher-priority request, the channel raises its "Send" line. The unit responds via its "Ready" line. The unit then places the transfer address on line for channel storage and sets a transfer direction control line, the "In" line. If the "In" line is received in its true state, the channel loads the data from the unit, sets itself busy, and requests a memory cycle for storing the data in memory. When the "In" line is received set false, the channel requests a memory cycle for access purposes and, when the cycle is granted, the channel loads the data word from the address furnished by the unit. The channel then pulses its "Output Data Here" line to load the data into the unit.

IBC Channel Synchronization

The IBC channel is sequenced for block transfers via the units' "Data Transfer Request" lines. (See previous paragraph for similar transfer capability.) The channel also specifies the transfer direction, but this is a reflection of the command word to the unit. In normal operation, channel parameters are loaded via the conventional block-transfer-initiate sequences into RAM locations reserved for units served by the channel. The unit, however, may be commanded to an external addressing mode in which it loads the unit's Transfer Address Register (TAR) and controls whether the TAR and/or Word Count Register (WCR) are incremented/decremented, respectively.

The IBC channel does not "shake hands" with the unit during command transfers; the command is automatically loaded by the unit controller since the channel "selects" the unit, bypassing the usual disconnect/connect sequence.

UBC Channel Synchronization

UBC channel boards contain two logical channels which share the unit bus via assigned scan cycles derived from internal timing. Each channel communicates with an addressed unit for transfer and handshake purposes only during its assigned scan cycle. If a data transfer occurs, the scan cycle is extended until the handshake takes place or is timed out. For OCW/ODW instructions the handshake sequences are as previously described. For IDW/ISW instructions, the channel must have first established that a "status ready" condition exists. This condition requires that the channel has iteratively received status (automatically) or data (if available) from the addressed unit during the most recent two assigned scan cycles. If this is true, the channel automatically transfers the status to the CPU during an ISW instruction; however, no handshake is sequenced with the unit. If the input data has been loaded by the channel, the data is transferred to the CPU during an IDW instruction, and the channel signals "acceptance" during the next assigned scan cycle.

The same handshake sequences occur during block transfers, but the channel is capable of 48-bit (double) word transfers to/from memory. This allows the channel to shake hands with the unit twice for each memory cycle requested, transferring a 24-bit word with each handshake.

Timing

All of the I/O channels except the UBC, DMACP and IBC depend solely on computer clock pulses for execution of single-word instructions or, where applicable, block-transfer operations. The UBC, DMACP and IBC channels are synchronized to CPU timing for some sequences but may provide other sequences via independent internal timing.

Block Transfer Memory Access

Block I/O operations are controlled by the channel after it has been initiated under program control. The channel, therefore, accesses memory for read/write operations and must request memory cycles for this purpose. In memory transfers, the requested memory cycle is automatically granted unless the CPU is in an error correct cycle.

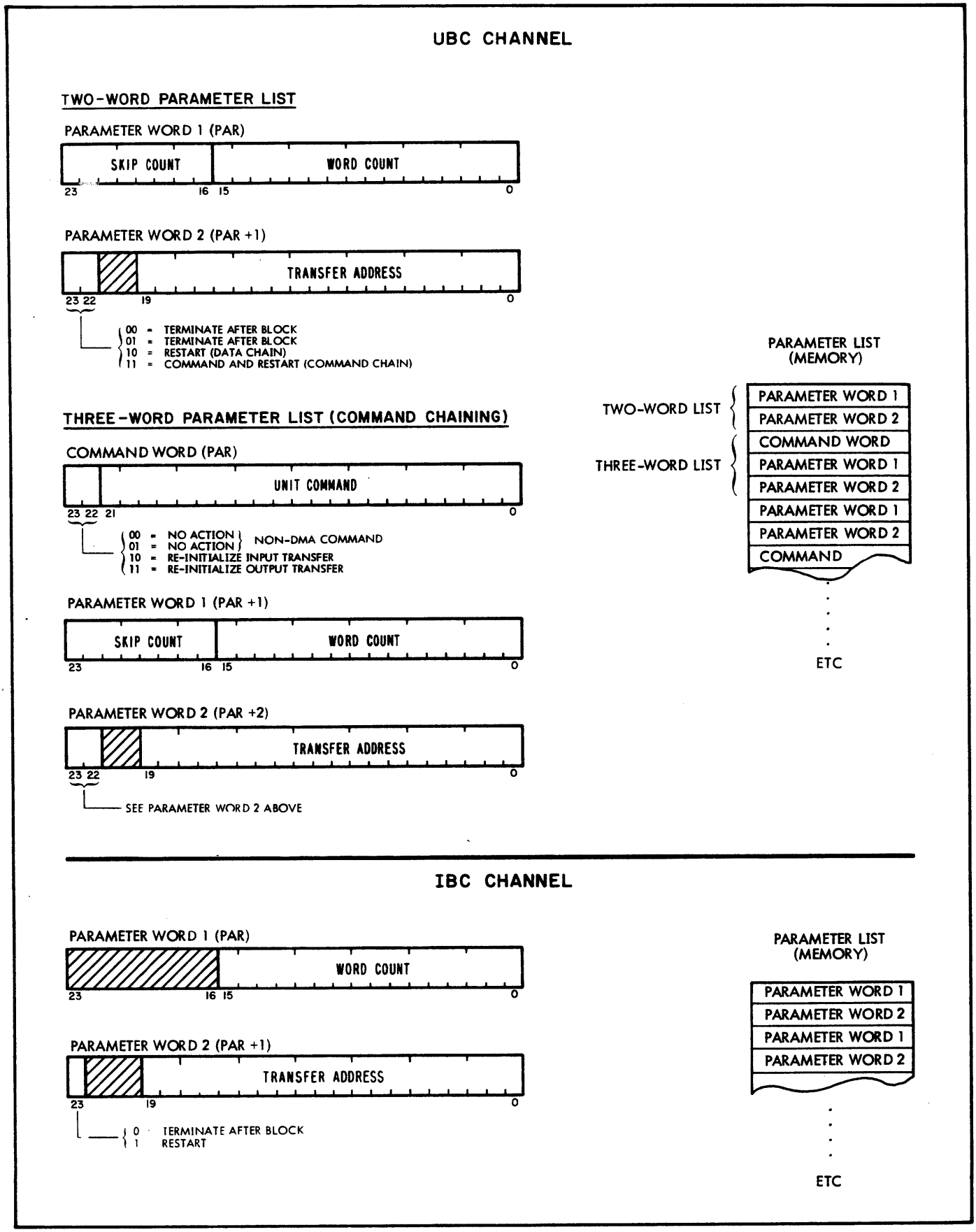
When a memory cycle is granted by memory, the control signal is permitted into the highest priority channel generating a cycle address. The "memory granted" signal activates the channel to load the word from memory (output transfer) or transfer a previously-loaded word from the unit to memory for storage (input transfer).

Block Transfer Parameters

The UBC, DMACP, and IBC are initiated for block-transfer operation via an OCW instruction. The command word itself must have bit 23 set to activate the block-transfer mode. These conditions activate the channel to sequence two simultaneous memory requests for parameters. The designated parameter words are illustrated in Figures 4-2 and 4-3.

UBC Channel Parameter Words

The UBC channel parameter word formats are illustrated in Figure 4-2. In this channel the OAW instruction preceding the OCW used to initiate the block transfer control causes the first parameter address (PA) to be loaded into a parameter address register (PAR). This allows the parameters to be located in a separate "list", but the list must be located in the lower 65K of memory. Each time the PAR is addressed for a parameter word, the channel increments the PAR for subsequent parameters.



MI1821-976B

Figure 4-2. UBC and IBC Parameter Word Formats

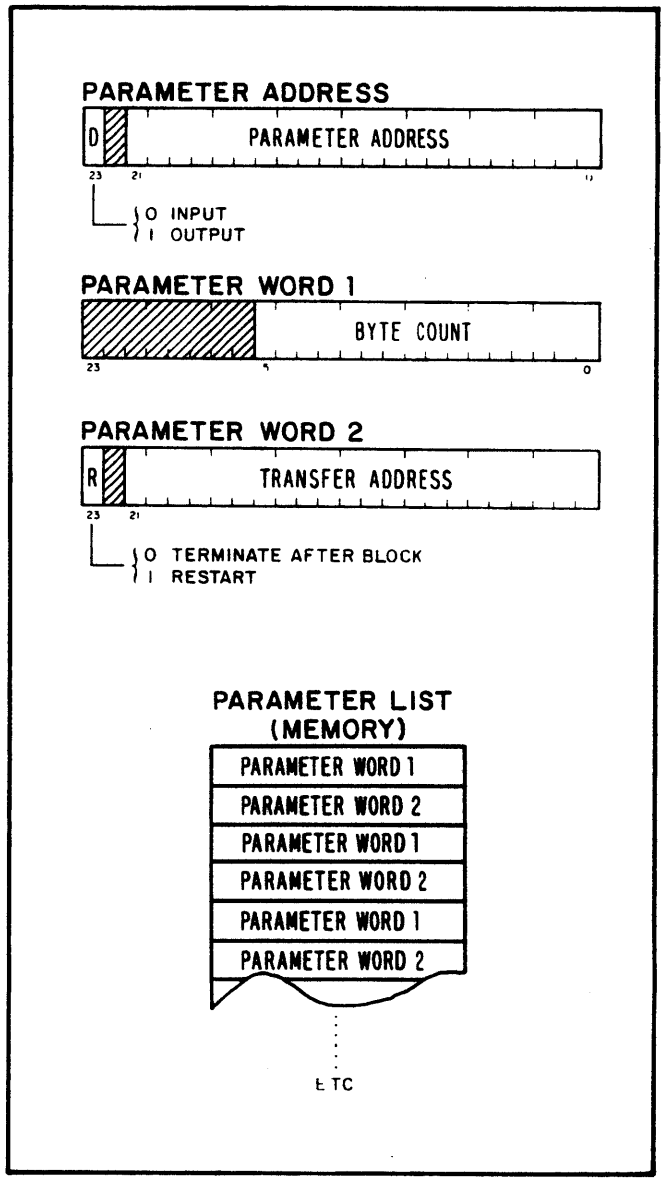


Figure 4-3. DMACP Parameter Word Formats

The first parameter applicable to UBC operations contains a 16-bit word count and the most-significant 8 bits contain a "Skip Count". The skip count is significant only in block transfers designated for input and is loaded into the channel's skip count register (SCTR). This parameter controls the actual transfer operations in which data is loaded into memory. When a count is set into the SCTR, the channel provides load sequences to transfer the data from the unit to the channel but does not request memory cycles to load the data into memory. The SCTR is decremented with each word transferred and, when the counter has decremented to zero, the channel begins data transfers to memory based on the word count parameter.

The second parameter word in UBC applications contains a 20-bit TA. The two most-significant bits of PW2 are stored in the channel and specify four termination sequences that may be entered when the block transfer has been completed; these are:

a. Normal termination – the channel goes to a "not busy" state when the last data word has been transferred.

b. Data restart – the channel goes into a re-initiate sequence to bring in two new parameters. The subsequent block transfer is as specified in the OCW initiating the previous block transfer.

c. Chain command restart – the channel goes into a re-initiate sequence in which a new command (from memory) is sent to the unit to change the transfer direction. As with the OCW initiating block mode operations, bit 23 of the command word must be set to command the initiate sequence. This is followed by bringing in PW1 and PW2 to set channel control action for the block of data to follow.

d. Chain command terminate – the channel goes into a re-initiate sequence in which a new command (from memory) is sent to the unit. If bit 23 of the command word is not set, the channel goes to a "not busy" state when the transfer sequence is completed.

XBC Channel Parameter Words

The XBC channel does not contain circuits to store and control parameters. Likewise, the channel does not provide any restart actions. The parameters are controlled by the external device, but the device may require that the parameters be initially furnished from memory. In the latter case, the channel is sequenced to execute an OAW instruction which transfers the TA to the unit. This is followed by an ODW instruction which sends the word count to the unit. After being initiated by the OCW command, each data transfer is sequenced and the unit itself furnishes the transfer address. The unit controls the word count and generates any operational interrupts.

IBC Channel Parameter Words

The IBC channel is initiated to the block-transfer mode via the conventional OCW with bit 23 of the command word set. The IBC channel then enters the initiate sequence to load two parameter words (Figure 4-2). The first parameter word contains the word count of the subsequent block transfer. The second parameter word contains an 18-bit TA and the "restart" condition. The IBC channel does not provide chain command functions in a restart operation. But, since the IBC channel contains storage for parameter addresses, the channel may access PW1 and PW2 from a "list".

DMACP Channel Parameter Words

Each port has assigned to it a Parameter Address Register, a Byte Count Register, and a Transfer Address Register. These registers are located in the Parameter Stack located on the DMACP board. Refer to Figure 4-3.

The Parameter Address Register contains the starting address in main memory of the next parameter list. The parameter list specifies the byte count to be placed in the Byte Count Register, and the transfer address to be placed in the Transfer Address Register. Along with the parameter address, a transfer direction bit (23) specifies whether the transfer is to be an output from main memory to the DMACP (ONE), or an input from the DMACP to main memory (ZERO).

Parameter Word 1, loaded into the Byte Count Register, contains in binary format the number of bytes of data to be transferred between main memory and the selected port. Maximum byte count per DMA sequence is 65,536.

Parameter Word 2, the transfer address, is loaded into the Transfer Address Register. The transfer address represents the location in main memory where the next data word (three bytes) is to be transferred. Each time a word is transferred, the TAR is incremented to point to the next memory address. An automatic restart function is provided to enable successive blocks of data to be transferred without CPU intervention. This is accomplished with bit 23 of the transfer address. If this bit is a ONE, the microprocessor will fetch a new byte count and transfer address from main memory as specified by the Parameter Address Register. A restart occurs when the existing count in the Byte Count Register reaches zero. When bit 23 is a ZERO, the restart function is disabled.

INPUT/OUTPUT INSTRUCTIONS

Execution of I/O instructions consists of the transfer or command (OCW), data (ODW and IDW), status (ISW), or address (OAW, IAW, IPW) words between the A Register and the specified channel/unit combination. The channel/unit codes in each I/O instruction (excluding OAW, IAW, and IPW instructions in applicable block-transfer channels except the IBC) allow one channel to be selected and one of up-to-16 units to be connected to the channel. When an instruction to the same channel carries a different unit code, the previously-specified unit is disconnected and the new unit is connected automatically. During this disconnect/connect sequence, the channel is busy and does not respond to I/O instructions until the sequence is completed. If a channel is in the process of transferring commands or data to a unit, an ISW or IDW instruction addressed to a different unit on the same channel receives a busy indication.

Command and data words from the CPU are transferred to the channel output buffer and subsequently to the connected peripheral unit. Data and status words are retained in the input buffer of the selected unit and transferred to the A Register upon request (instruction) from the CPU. Address words are applicable only to those channels employing the block-control capability. (Refer to I/O instruction formats in Section VII for the following discussions.)

I/O Commands

The OCW instruction transfers a command word to the specified channel/unit combination. The command word bits specify the unit control function(s) to be performed and/or the I/O condition to be established. Following the execution of an OCW instruction, the channel remains busy until the command has been accepted by the addressed unit. Figure 4-4 shows the format for a typical OCW instruction.

If the channel is busy or not ready when addressed by the OCW instruction, the Condition Register is set to "Not Zero" to allow a programmed delay. The override function causes the channel to automatically perform a unit disconnect/connect sequence. This clears the channel of any other activity and allows the current instruction to assume control of the channel unconditionally upon termination of the disconnect/connect sequence.

All of the I/O channels execute the OCW instruction, but channel capabilities may require setting of the instruction contents as follows:

a. Unit Addresses — The IBC channel contains interface capability for up-to-two devices. The unit address must therefore be set in Unit Code bits 0 and 1. The unit addressing requirements for the XBC channel is contained in Unit Code bits 0-2. Unit code 10_8 is the only valid code for the DMACP channel. All of the remaining channels, having the capability to interface with up-to-16 units, utilize all of the Unit Code bits for addressing purposes.

b. Channel Command Mode — Bits 4 and 5 provide command control to set an I/O channel to one of four modes: Normal, Offline, Multiplex, and Reset. The Normal mode specifies "normal" command functions. The Offline mode removes the units from the I/O channel interface, permitting a second computer and I/O channel to assume control over the units. The Multiplex mode allows a "Master" unit to communicate with a "Slave" unit and the CPU cannot intervene except via a Master Clear or an OCW instruction with "Override" specified or Reset mode commanded. The Reset mode allows a return to Normal mode operations from either the Offline or Multiplex modes.

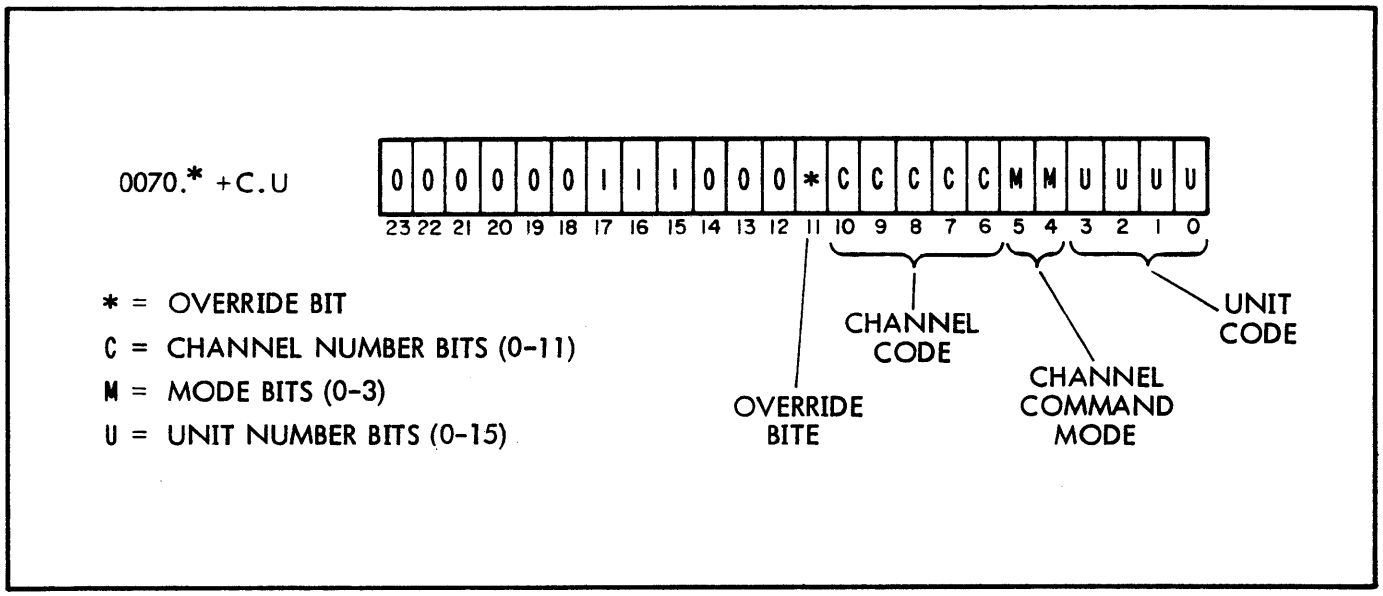


Figure 4-4. OCW Instruction Format

The IBC and DMACP channels do not respond to the mode control specifications of an OCW instruction (they thus always operate in the Normal mode).

The XBC, DMACP, and IBC channels cannot be commanded to the Multiplex mode of operation. The remaining channels may be commanded to any of the modes described above.

c. Override Control – This OCW instruction control function is exercised in all I/O channels except the XBC. An OCW instruction with this bit set assumes immediate control of the channel/unit by forcing a disconnect/connect sequence.

I/O Status Word

The ISW instruction is used to test the operational status of the channel/unit. When a channel is addressed by the ISW instruction, a 24-bit status word is transferred to the A Register in the CPU. The quantity and significance of the status bits depends on the type of peripheral unit involved. Units controlled by 8-bit interface channels (e.g., PIOC) furnish up to six unit-defined status bits which the channel sets into the least-significant bits of the input word. Channels with 24-bit unit interfaces (e.g., block controllers) may receive as many as 8 unit-defined status bits which are set into the 8 least-significant bits of the input word.

Channel status may be set into the three most-significant bits of the input word and reflect the channel's current mode or "busy" status as follows:

IBC and DMACP	None
PIOC	Bit 21 – Multiplex Bit 22 – Offline
UBC	Bit 21 – Multiplex Bit 22 – Offline Bit 23 – Busy
XBC	Bit 22 – Offline

Single Word Data Transfers

Input Data Word

The IDW instruction is a request from the CPU to a specific channel/unit combination for a data word. If data is available, the data word is transferred immediately to the A Register. If data is not available, the Condition Register is set to "Not Zero" to allow a programmed delay.

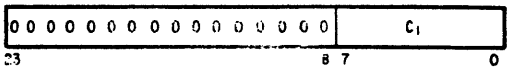
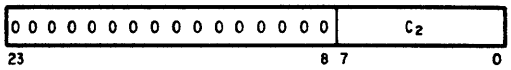
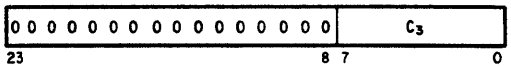
Normally, the 24-bit input data word contains a single data character. The actual number of data bits per character depends on the channel and unit involved in the transfer. For example, the console typewriter generates an 8-bit character and a card reader may generate a 12-bit character. In any case, the character is right-justified in the A Register with the unused bit positions set to ZEROs.

Assuming the data character contains no more than 12 bits, more than one character may be packed in the A Register through the use of the Merge feature. When a character Merge is employed, a logical OR is performed between the previous contents of the A Register and the new input data word. Without the Merge, the previous contents of A are destroyed upon transfer of a new character to A. An illustration of the character Merge technique, as compared to a normal IDW instruction, is shown in Figure 4-5.

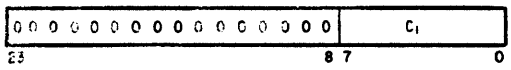
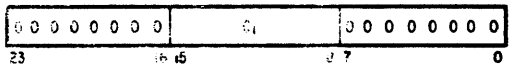
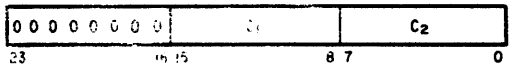
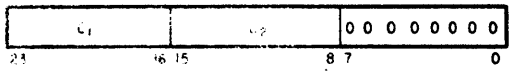
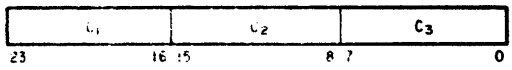
The IDW instruction is executed by all I/O channels except the XBC and IBC.

EXAMPLE: THREE 8-BIT DATA CHARACTERS ARE TO BE PACKED IN THE A REGISTER.

(a) NORMAL (WITHOUT MERGE)

CODING	COMMENTS	REGISTER A
IDW CU	BRING IN FIRST DATA CHARACTER	
. . .		
IDW CU	BRING IN SECOND CHARACTER	
. . .		
IDW CU	BRING IN THIRD CHARACTER	
. . .		

(b) MERGE

CODING	COMMENTS	REGISTER A
IDW CU	BRING IN FIRST DATA CHARACTER	
LLA 8	SHIFT LEFT 8 PLACES	
IDW* CU	BRING IN SECOND CHARACTER AND MERGE	
LLA 8	SHIFT LEFT 8 PLACES	
IDW* CU	BRING IN THIRD CHARACTER AND MERGE	
. . .		

MI60-005-770

Figure 4-5. IDW Instruction; Data Character Formatting

Output Data Word

When an ODW instruction is executed, an 8- or 24-bit data word is transferred from the A Register to the specified channel. The data word is subsequently transferred from the channel to the unit that is currently connected. If the channel is busy or not ready to accept the data word, the Condition Register is set to "Not Zero" to allow a programmed delay. If the unit is not ready to accept the data from the channel, the data remains in the channel buffer.

As soon as the peripheral unit is able to accept the data from the channel, the channel-to-unit transfer is made, thereby freeing the channel buffer for another data (or command) word from the CPU.

The number of data bits accepted by the peripheral unit varies according to the type of unit involved. Some peripheral units are word-oriented and accept the entire 24-bit word. Others are character-oriented and accept only a specific number of bits per character.

The ODW instruction function in XBC I/O channels serves the purpose of sending a word count parameter from the CPU A Register to the addressed unit, if required by the unit. In subsequent block-transfer operations the unit controls the WC parameter. The IBC channel does not execute the ODW instruction.

Address Transfers

Three address-transfer instructions are executed by block-transfer channels for the purpose of channel or unit set-up for subsequent transfers (OAW) or for CPU checks of transfer progress (IAW and IPW). However, the PIOC board may execute the OAW instruction. The following discussions cover applicability and qualifications for the address-transfer instructions.

Output Address Word

The OAW instruction is executed by the DMACP, UBC and IBC to set the starting address of parameters for block-transfer control. The XBC also executes the OAW instruction if a unit on its interface requires a TA starting address.

The DMACP, IBC and UBC channels load their respective PAR during execution of the OAW instruction. The instruction is executed in a single machine cycle.

NOTE

In UBC execution of the OAW instruction, the block-transfer logic is cleared. Therefore, this instruction should not be programmed for execution until all block transfer operations are completed.

The XBC channel will not execute the OAW instruction if the channel is busy executing an output command or a data instruction. The instruction word must be addressed to the unit to which the TA parameter is intended. Therefore, a "programmed delay" should be programmed to facilitate instruction execution.

In IBC channel OAW execution the instruction word must be addressed to a unit controller contained on the channel board. The channel executes the instruction in a single machine cycle, writing the PA into a register reserved for the addressed unit.

Available for software interrupt purposes, an Interrupt Generator is located on the PIOC board to allow generating one-of-four possible interrupt pulses in response to an OAW instruction. The instruction is executed automatically by the addressed channel to provide one microsecond interrupt pulses which may be routed for use as interrupts in another CPU or in any peripheral unit.

The Interrupt Generator responds to the particular OAW instruction with the proper channel code. The four least-significant bits (3-0) of the A Register, during the OAW instruction, will trigger the pulse from the generator. The pulse remains at the "true" level for the 1 microsecond cycle and then is restored to the "false" state. There is no interaction between the generation of different numbered interrupts, but the generation of the same numbered interrupt is limited to not more than one per microsecond. There is no response to the mainframe C (condition) Register during the execution of the OAW, i.e., if the C Register were tested, it would indicate "not zero".

In summary, if an interrupt pulse is to be generated, the following coding could be applied:

TOA	B0B1B2B3	(Unitary bits; one for each interrupt pulse line.)
OAW	CU	

Input Address Word

The IAW instruction may be addressed to any of the block-controller channels except the XBC channel. For IBC channel purposes the instruction word must be addressed to the channel and unit; otherwise the instruction is addressed only to the desired channel. In all applicable channels except the IBC the instruction is automatically executed during the current instruction cycle. The IBC channel executes the instruction only if it is not busy executing another instruction or transferring data. In all cases, the channel sets its "Ready" line to the CPU to clear the C Register. The address word is sent to the A Register and may be used as a check on transfer progress. The word represents the TA of the current transfer and is always 18 bits wide.

Input Parameter Word

This instruction is very similar to the IAW. The instruction is addressed only to those block I/O channels capable of PA storage: DMACP, UBC and IBC channels. The execution of the IPW instruction is identical to the IAW instruction.

INTERRUPT CONTROL

The OCW instruction may be used to selectively enable and disable two peripheral unit interrupts in PIOC board operations. The two interrupts are defined as Input and Output and are controlled by bits 2-0 of the command word. Table 4-1 illustrates the various bit configurations.

Table 4-1. Peripheral Unit Interrupt Control

Command Word Bit Configuration	Action
2 1 0*	
0 0 0	No Action.
0 0 1	No Action.
0 1 0	Disable Input (or Alternate) Interrupt
0 1 1	Enable Input (or Alternate) Interrupt
1 0 0	Disable Output (or Alternate) Interrupt
1 0 1	Enable Output (or Alternate) Interrupt
1 1 0	Disable Both Interrupts
1 1 1	Enable Both Interrupts

*No significance to some units, i.e., the interrupts are unconditionally enabled by CW Bits 1 and/or 2.

The terms "input interrupt" and "output interrupt" are applicable only to peripheral units that are equipped with both input and output data handling facilities. Input-only devices may make use of the input interrupt and an alternate interrupt at the normal output level. Output only devices may make use of the output interrupt plus an alternate at the normal input level.

When the unit input interrupt has been previously enabled, an input interrupt signal will be generated when the input buffer in the unit is loaded (i.e., the same time the "Data Available" signal is generated). An I/O channel has no control over an input interrupt.

When the unit "output interrupt" has been previously enabled, an output interrupt signal may be generated by the channel for two sets of conditions based on a device-defined signal, "Enable Channel Buffer Empty Interrupt" (ECBEI). If the unit raises ECBEI to the channel, the output interrupt will be generated for a minimum of 325 nanoseconds if:

A. PIOC board;

1. the channel has not been commanded to the Offline or Multiplex mode, and,
2. the channel is not performing a disconnect/connect sequence, and,
3. the channel's output buffer is not holding a command/data word for unit transfer purposes.

B. XBC, UBC and IBC channels;

These channels contain no output interrupt capability.

If the unit holds the ECBEI signal to the channel low, the output interrupt will be generated by the channel but the channel's output buffer condition (3, above) is ignored. Instead, the device-defined state of Status Bit 2 from the unit is allowed to set the output interrupt. The mode and manual conditions described for each type of channel above remain in effect.

The UBC channel contains the capability to generate a "word count complete" interrupt when the channel has loaded the final word of a block-transfer operation. (The IBC channel generates a "word count complete" signal to the unit when the channel has loaded the final word, if no "Restart" is specified. This signal, however, is under the control of the unit for interrupt purposes.) The approximate duration of the interrupt is 475 nanoseconds.

C. DMACP Channel;

Two interrupts are generated by the DMACP channel: 1) whenever a parity error is generated within the RAM located on the DMACP board, and 2) whenever one of the ports requires service. The particular event causing the interrupt can be determined by executing an ISW instruction to fetch the status word.

I/O CHANNEL SWITCH/PATCH CONTROLS

The various I/O channels contain switch and patching provisions to perform a number of operational functions. The PIOC board's patching capabilities is restricted to channel address selection. The block-transfer channels are also patched, or switches set, to encode a unique channel address, but those channels also contain a variety of other manually-activated functions. These functions are listed in Table 4-2 with I/O channel applicability specified.

I/O CHANNEL OPERATIONAL SUMMARIES

The following paragraphs summarize single-word and block-mode transfer capabilities of the various I/O channels interfacing with the computer. Included are program lists and suggestions. Refer to the paragraph describing input/output instructions for application to specific I/O channels.

Table 4-2. I/O Channels Manual Control Capabilities

Function	PIOC	IBC	UBC	XBC	DMACP
Permanent Offline/Multiplex mode selection	Switch		Switch		
Channel code selection	Switch	Switch	Switch	Switch	Switch
Memory cycle priority		Switch	Patch	Switch	Switch
Unit selection		Patch			

Single-Word Instruction Execution

OCW/ODW

The channel, if not busy, loads a command or data word from the CPU A Register into its output buffer. The channel sets itself "busy" to inhibit any further instruction executions until it has completed the transfer to the addressed unit. In the event of a disconnect/connect sequence, the channel withholds the handshake until the addressed unit is "connected" to its interface. A BNZ instruction should be programmed to verify channel execution of the OCW instruction.

IDW

The channel executes this instruction in one machine cycle if the channel is not busy executing an output transfer, is not involved in a disconnect/connect sequence, and the connected unit has signalled that data is available for transfer via its "Data Available" line. The BNZ execution performed by the CPU provides verification of transfer. The channel shakes hands with the connected unit and is ready for further instructions.

ISW

An I/O channel executes this instruction if the channel is not busy executing an output transfer and is "connected" to the addressed unit.

OAW

This instruction is addressed to block I/O channels (unit in XBC/IBC applications) for the purpose of transferring the address of the first word involved in control of a subsequent block data transfer. Channel loading of the output word from the CPU's A Register into the channel's PAR (UBC and DMACP applications) is automatic. In XBC applications the instruction involves transferring a TA to the unit for subsequent control by the unit. The XBC channel must have gone to "not busy" prior to instruction time for execution. A programmed delay must therefore be executed by the CPU for verification of transfer. A handshake with the unit is performed in this instruction and the channel sets itself busy until the transfer-to-unit is completed. In IBC applications the addressed channel executes the instruction unless previously set busy via an instruction or data transfer sequence.

IAW/IPW

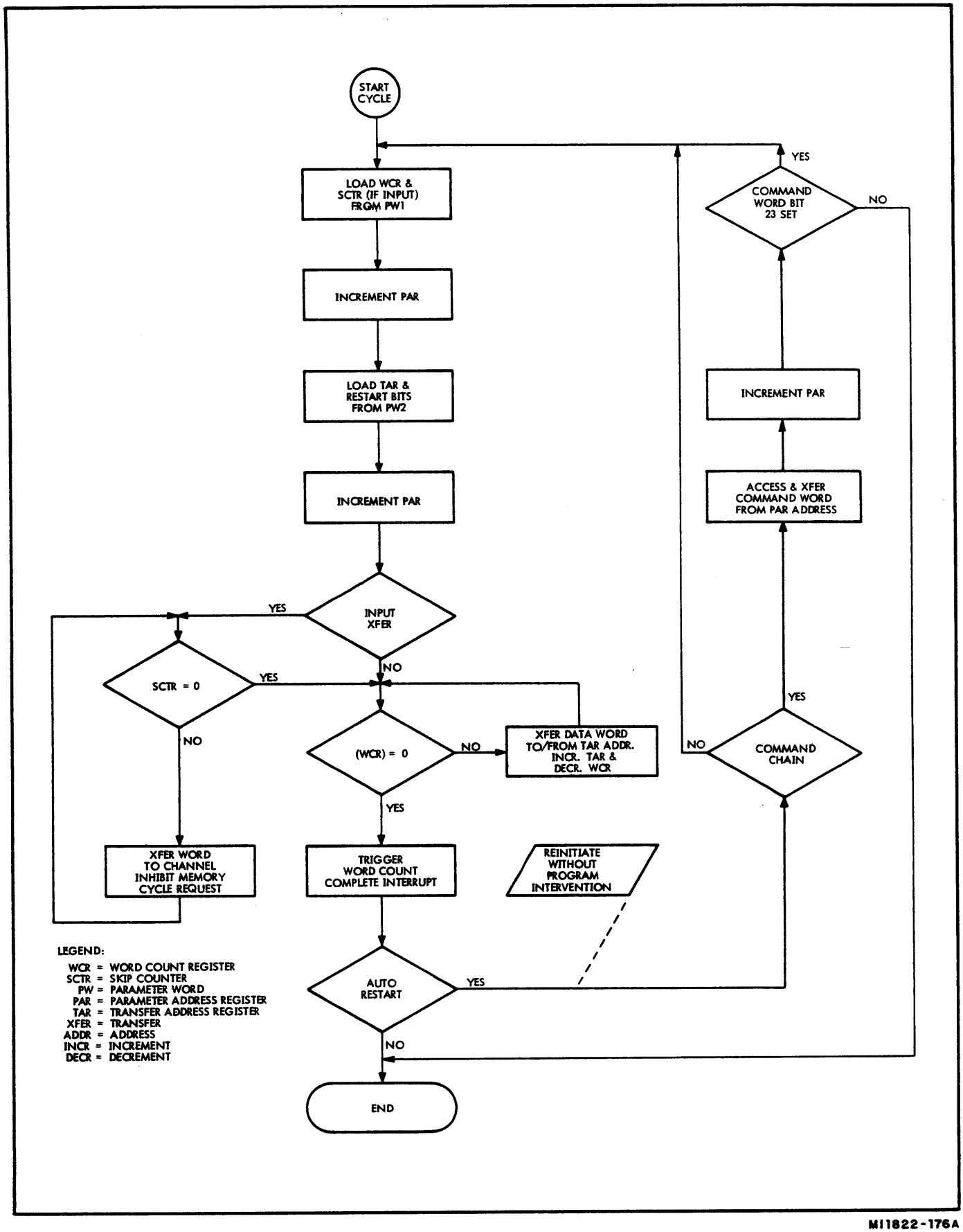
The IAW/IPW instructions are executed to transfer the contents of a block I/O channel's TAR/PAR to the CPU's A Register. The IPW and IAW instructions are not executed by XBC channels. The instructions, when applicable, are executed automatically in UBC channels. The IBC channel is inhibited from executing the instruction if currently busy in an instruction or data transfer operation.

Block-Transfer Operations

All block I/O channels are initialized by computer control for block-transfer operations and proceed under self control or unit control to perform the transfer operations. The following paragraphs describe general performance of block transfers applicable to each channel. Refer to Figures 4-6 through 4-9 for simplified flow diagram of block-transfer operations.

UBC Channel Block Transfers

The UBC channel is "set-up" via an OAW instruction and initiated via an OCW instruction with bit 23 of the unit command specifying the block transfer and bit 22 specifying the direction of transfer. During the OCW sequence the channel sets itself "busy" to all ODW, IDW, and OCW instructions (except an OCW specifying "Override"). The channel remains busy for the duration of transfers initiated by the OCW instruction. The channel automatically loads two parameter words (see Figure 4-2). If an output transfer has been specified, the channel sequences a memory request and specifies the location via its TAR. The channel increments the TAR, decrements its WCR, and loads the data word in its output buffer when the memory cycle is granted. The channel then "shakes hands" with the unit to complete the transfer. The channel then fetches another word for transfer. When the WCR has decremented to ZERO, the channel examines its "Restart" parameter (bit 23 of PW2) and either re-initiates itself for another block transfer or returns to an "idle" state, resetting its "busy" condition.



MI1822-176A

Figure 4-6. UBC Block Transfer Sequence; Simplified Flow Diagram

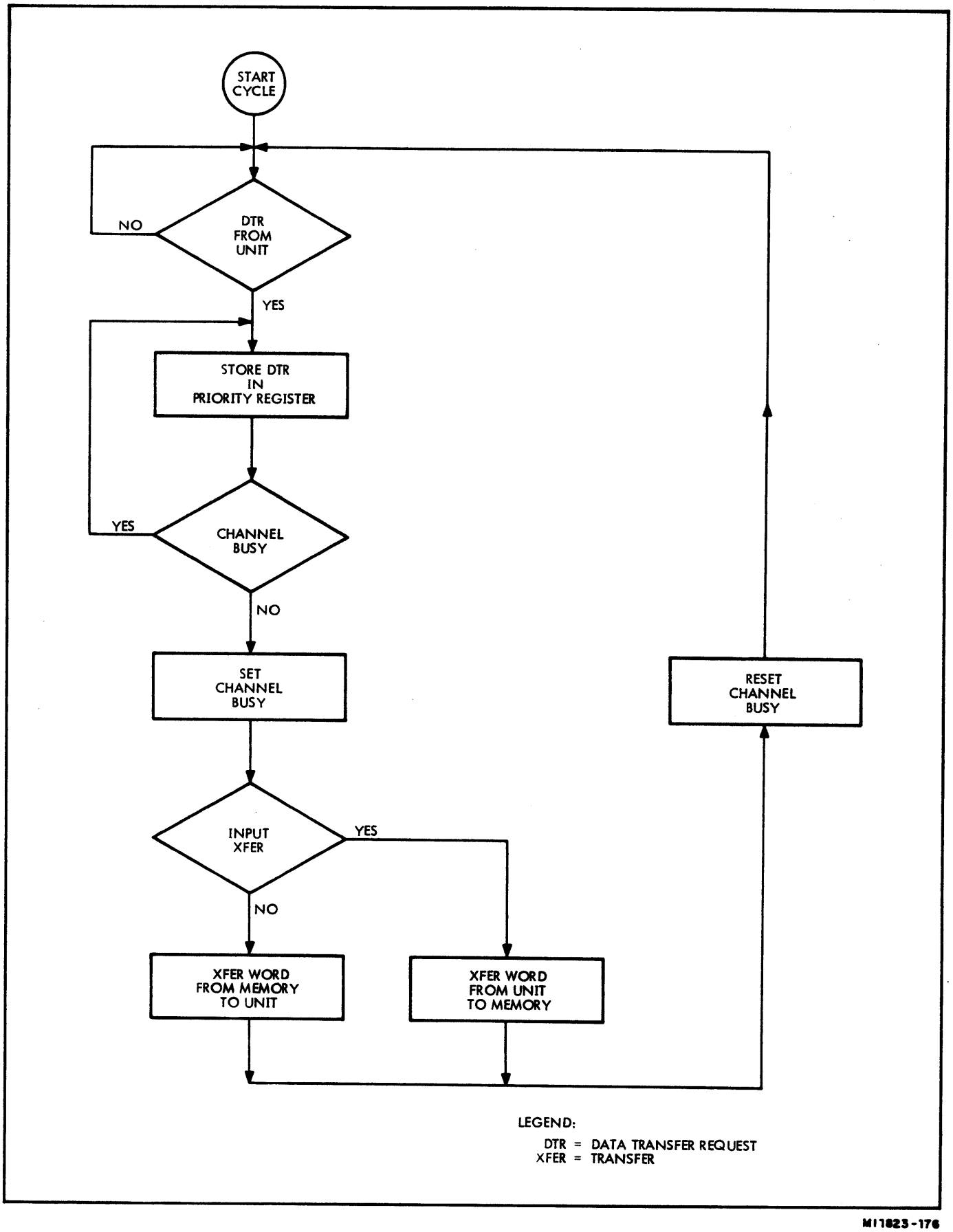
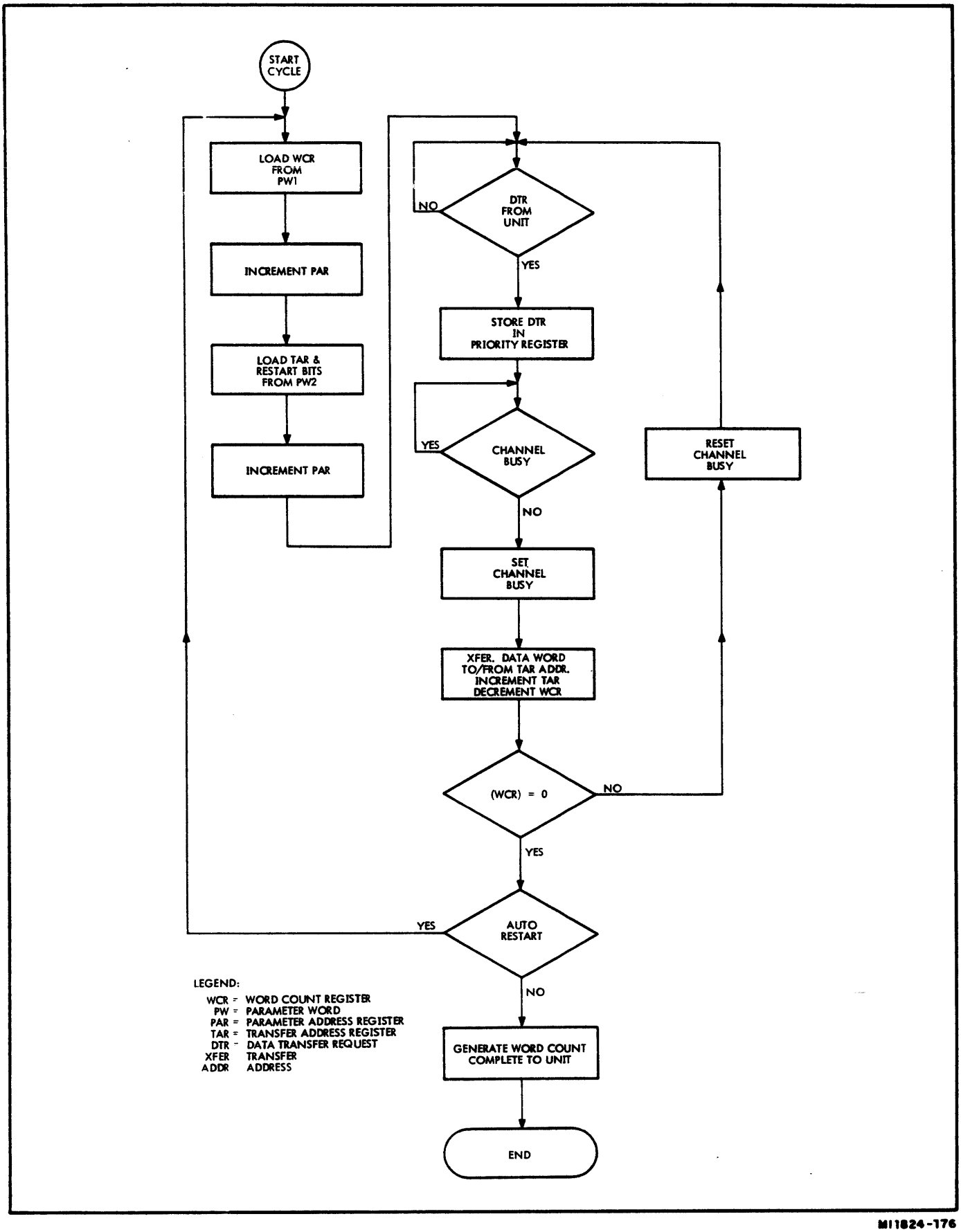
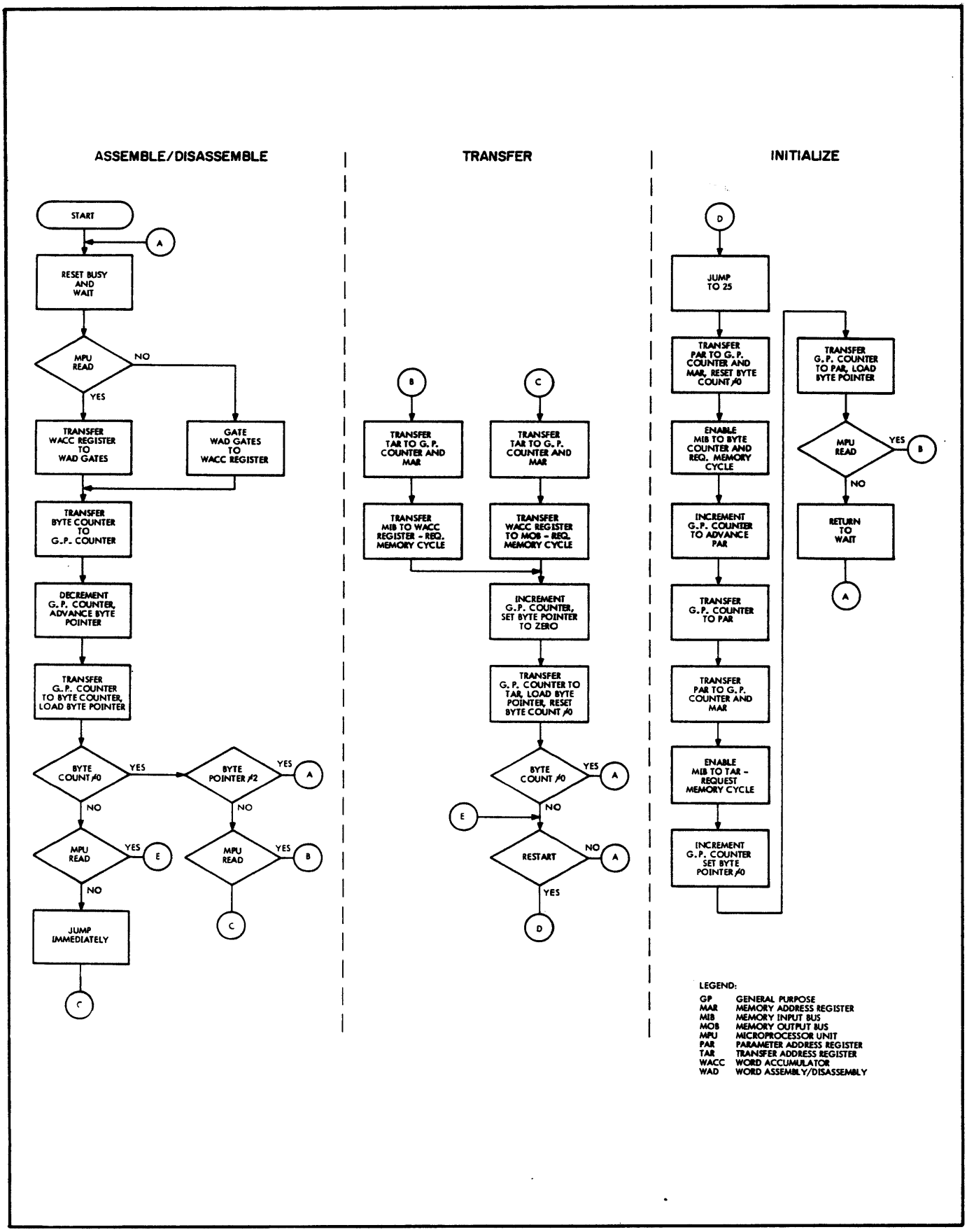


Figure 4-7. XBC Block Transfer Sequence; Simplified Flow Diagram



MI1824-176

Figure 4-8. IBC Block Transfer Sequence; Simplified Flow Diagram



MI 2057A-577

Figure 4-9. DMACP Channel Block Transfer Sequence; Simplified Block Diagram

If an input transfer was specified via the OCW, the channel waits for the unit to signal data availability. The channel then loads the input data into it's input buffer and signals "accepted" to the unit to free it for the next word. The channel then requests a memory cycle, and, when granted, places the TA and data on line to memory. The channel increments the TAR, decrements the WCR, and returns to sense the unit's "Data Available" line. This sequence continues until the WCR forces the restart sequence as described above.

The UBC channel contains a Skip Count Register for added parameter control in input transfer operations and may enter an alternate "Restart" after a block of data has been transferred.

The output data transfers are sequenced in an identical fashion. The channel's capability to "Restart and Chain Command" allows the re-initiate sequence to access an additional parameter (in this application, a new command to the unit) to change transfer direction without program intervention. In this situation, the new command word initiates the channel in the same manner as did the original OCW instruction.

The Skip Counter affects only those transfers slated for memory. The skip count allows the channel to pass over unwanted data (sync codes, etc.) before actual data loading is sequenced. When the skip count parameter specifies a count, the channel sequences handshakes with the unit to unload the unit, but the channel does not request the memory cycles from the CPU to load the data into memory. The SCTR is decremented with each transfer, but the TAR and WCR remain unchanged. When the SCTR has decremented to ZERO, the channel begins loading data words into memory.

XBC Channel Block Transfers

The XBC channel is normally initiated to block-transfer operations via an OCW instruction in which a command is transferred to the unit. If required, the OCW may have been preceded by OAW and/or ODW instructions to transfer TA and WC parameters to the unit. Once initiated, the channel is under the control of the unit for transfer purposes. When the unit signals a "Data Transfer Request" (DTR), the channel, if not previously set busy, sets itself busy and stores the TA from the unit. The unit specifies the transfer direction and, if an input transfer is specified, the channel "accepts" the data from the unit. The channel then requests a memory cycle and, when granted, transfers the data to memory, based on the TA furnished by the unit.

If the unit specifies an output transfer, the channel requests a memory cycle. When the cycle is granted, the channel places the TA on line to memory, loads the data from memory and performs a "Data Here"/"Accepted" handshake with the unit in which the data is transferred to memory.

The XBC channel's "busy" condition is reset after each instruction or data transfer is accomplished. The unit controls the TA and WC parameters and generates any required interrupts. With no mainframe contention, the maximum block transfer rates are 800,000 (input) and 666,666 (output) words per second. With mainframe contention, the transfer rates are 476,000 (input) and 428,000 (output) words per second.

IBC Channel Block Transfers

The IBC is set-up and initiated for block transfers via the OAW and OCW instructions, but the channel sets itself "not busy" after each instruction or data transfer. The channel may thus store the two parameter words for up-to-two self-contained unit controllers (Figure 4-2) and interleave data transfer.

Data transfers are sequenced by the channel based on "Data Transfer Request" signals from the units. The DTR lines are priority-structured and the unit indicates the direction of transfer. Data transfers then proceed as described for XBC channel operations except as follows:

A. the unit allows/inhibits TA and WC incrementing and decrementing by setting its "Block Mode" control line true/false (see External Addressing mode below).

B. the unit does not furnish the TA parameter (except in the External Addressing mode).

C. the channel/unit does not "shake hands" in output transfers.

D. the channel generates "Word Count Complete" to the unit only, which then controls the interrupt to the CPU.

The IBC channel may enter an External Addressing mode by the unit presenting its DTR, "Address Here," and "Input" lines set to the channel. The address is then loaded into the TAR for the specified unit. The data presented with the next DTR is transferred into or out, as set, of the memory address of the unit's TAR. If the "Address Here" signal is not presented again to change the TAR, any further data transfers will use the same TAR address. This allows the use of a specified memory address as a register.

The maximum transfer rates for the IBC channel block-transfer operations are determined by the card reader and floppy disc connected to the channel.

DMACP Channel Block Transfers

DMA transfers between the DMACP and memory are under control of the microprocessor and associated logic located on the DMACP board. After a parameter address is sent with an OAW instruction, the CPU can command the microprocessor to perform a block transfer with an OCW instruction.

Data transfers are controlled by a sequencer and transfer control logic contained on the DMACP board. Three main functions are performed by the transfer control logic; initialization for a DMA transfer, word assembly/disassembly, and the actual transfer. These functions are performed by three subroutines comprising a program which is stored in the sequencer PROM located on the DMACP.

The microprocessor starts a DMA initialize operation by accessing a special location in a RAM contained on the DMACP board. Eight special locations in RAM are provided, one for each port. The DMA logic in the DMACP fetches the byte count and transfer address from the RAM locations specified by the parameter address. If an output operation is specified, the first 24-bit data word is transferred to a Word Accumulator Register. The microprocessor then transfers data bytes between the word accumulator and a communications port until the byte count equals zero. A terminate interrupt is sent to the microprocessor at the completion of the operation. The microprocessor then generates an interrupt to the CPU to indicate that service is required.

Program Lists

The following program lists specify various software control functions for block-transfer I/O channels. Note the functional identity of the applicable channels.

IBC Channel Applications

The following examples illustrate two different IBC applications.

Example 1: Simple, single buffer input.

```
        TOA     PA      Parameter Address
        OAW     C       Initialize TAR
        TMA     CW      Command Word
        OCW     CU      Initiate transfer
        BNZ     *-1     Delay if channel is busy
        ....
CW      DATA            Bit 23 and others as required by the I/O device
PA      DAC     n       Absolute Word Count
        DAC     BUFF    Address of Input Buffer
BUFF    BLOK    n       Reserve n words. Word n+1 is of no significance since the AR bit is not set.
```

Example 2: Multi-buffered output with automatic restart and buffer switching.

```
        TOA     PA1     Parameter Address 1
        OAW     C       Initialize TAR
        TMA     CW      Command Word
        OCW     CU      Initiate first transfer
        BNZ     *-1     Delay if channel is busy
        ....
CW      DATA            Bits 23, 22, and others as required by the I/O device.
PA1     DAC     n       Word Count
        DAC*    BUF1    Address of buffer 1 and the ARF (*)
PA2     DAC     n       Word Count
        DAC     BUF2    Address of buffer 2
BUF1    BLOK    n       Reserve n words
        DAC     PA2     Automatic Reinitialization address for TAR, to switch buffers
BUF2    BLOK    n       Reserve n words
        DAC     PA1     Automatic reinitialization address for TAR, to switch buffers
```

NOTE

Once this cycle is initiated it will continue, without program intervention, until a new command is received.

UBC Channel Applications

The following examples illustrate four different UBC applications.

Example 1: Simple, single buffer input.

```
        TOA     PA      Parameter Address
        OAW     CU      Initialize PAR and TAR
        TMA     CW      Command Word
        OCW     CU      Initiate Transfer
        BNZ     *-1     Delay if channel busy
        ....
CW      DATA            B23 and others as required by the I/O device
PA      DAC     n       Word Count
        DAC     BUFF    Address of Input Buffer
BUFF    BLOK    n
```

Example 2: Use Skip Count to read a single word from within a record.

```
        TOA     PA      Parameter Address
        OAW     CU      Initialize PAR or TAR
        TMA     CW      Command Word
        OCW     CU      Initiate Transfer
        BNZ     *-1     Delay if channel is busy
        ....
CW      DATA            B23 and others as required by the I/O device
        FORM    8, 16
PA      DATA    /111,112/  Word count. Input 112 words from device, skipping first 111.
        DAC     BUFF    Address of Input Buffer
BUFF    BLOK    1       Input Buffer
```

Example 3: Use Automatic Restart to Read a single record into discontiguous buffers.

```
        TOA     PA      Parameter Address
        OAW     CU      Initialize PAR and TAR
        TMA     CW      Command Word
        OCW     CU      Initiate Transfer
        BNZ     *-1     Delay if channel is busy
        ....
CW      DATA            B23 and others are required by the I/O device
PA      DAC     n       Word count of input into first buffer
        DAC*    BUF1    Address of first buffer (*) = ARF
        DAC     m       Word count of input into second buffer
        DAC     BUF2    Address of second buffer
        ....
BUF1    BLOK    n       Reserve n words
BUF2    BLOK    m       Reserve m words
```

Example 4: Use Command Chaining to read two records into a single buffer on same UBC transfer.

```
            TOA     PA          Parameter Address
            OAW     CU          Initialize PAR and TAR
            TMA     CW          Command Word
            OCW     CU          Initiate Transfer
            BNZ     *-1         Delay if channel is busy
            ....
CW          DATA                B23 and others as required by device to read first record
            ....
PA          DAC     n           Word count of first record
            DAC*    BUFF,J      Address of buffer for first record. (*) = ARF, and
                                (,J) = B22 for command and restart
            DATA                B23 and others as required by I/O device to read second record
            DAC     m           Word count of second record
            DAC     BUFF+n      Address of buffer for second record
BUFF        BLOK    n+m         Reserve n+m words
```

XBC Channel Applications

The following example illustrates an XBC application.

```
            TOA     INPAD       Set-up Input Buffer Address Start
            OAW     CU          Output the Address to Channel/Unit
            BNZ     *-1         Delay if Channel busy
            TOA     OUTAD       Set-up Output Buffer Address Start
            OAW     CU          Output the Address to Channel/Unit
            BNZ     *-1         Delay if Channel busy
Only if     TMA     WC          Set-up the required Word Count
Required    ODW     CU          Output the WC to Channel/Unit
            BNZ     *-1         Delay if Channel busy
-------------------------------------------------------------------------
INPAD       BLOK    100         Is the Starting Address of the Input Buffer that device may load
                                data into.*
OUTAD       BLOK    100         Is the Output Buffer that the device may read data from.*
WC          DATA    100         Number of Words to Transfer
```

*The external device controls the addressing and interrupt requests to the XBC channel. The external device also controls the word count.

SECTION V

PRIORITY INTERRUPT SYSTEM

GENERAL DESCRIPTION

The priority interrupt system provides added control over internal CPU operations and I/O functions, and immediate recognition of special external conditions on the basis of predetermined priority. Receipt and recognition of internal or external triggers allows the normal program flow to be diverted to interrupt service subroutines.

Two separate interrupt groups (0 and 1) are provided. Group 0 is reserved for internal CPU functions and is composed of up to eight executive trap levels. Group 1 is reserved for external interrupts. A maximum of 24 external interrupts are available.

INTERRUPT ORGANIZATION

Priority Conventions

All interrupt levels (both executive traps and external interrupts) are assigned a unique priority number. This assigned priority determines the order in which interrupts will be recognized and serviced. Interrupt levels descend in order of priority from Group 0, Level 0, to Group 1, Level 23. Group 0 has priority over Group 1; Level 0 has priority over Level 23.

Executive Traps (Group 0)

Each executive trap level is associated with a specific computer feature and is, therefore, permanently assigned. Each executive trap includes the associated internal interrupt level. Interrupt level assignments for the executive traps (Group 0) are listed below:

Level		
0	Power Down	Power Failure Shutdown/Restart
1	Power Up	
2	Program Restrict	See note following
3	Instruction Trap	
	Or	
2	Demand Page	
3	Instruction Trap	
4	Stall Alarm	
5	Interval Timer	
6	SAU Overflow/Underflow	
7	Address Trap	

NOTE

The Program Restrict and Instruction Trap is available as an option. If enabled by the key switch, executive trap Levels 2 and 3 are assigned to the Program Restrict and Instruction Trap, respectively. When virtual memory is enabled by the key switch, these levels are assigned to the Demand Page and Instruction Trap.

External Interrupts (Group 1)

A standard computer system includes interrupt logic and sixteen individual external interrupt levels. Eight of these levels are located on the mainframe CPU board and represent Group 1, Levels 16 through 23. Eight additional external levels are located on the option board. An additional eight levels can be added to the option board to provide a maximum of 24 external interrupts. The priority interrupt levels installed on the option board represent Group 1, Levels 0 through 15. With the exception of Levels 0 and 1, priority assignements of the interrupt levels are determined by system requirements and are made to meet user's requirements. Levels 0 and 1 are permanently assigned to report hard and soft parity errors, respectively. See Section III for a description of this function.

Dedicated Memory Locations

Each interrupt level has a memory location dedicated for its exclusive use. This applies to both the executive traps (Group 0) and external interrupts (Group 1). Dedicated memory locations for the interrupt system are as follows:

Addresses (Octal)	Assignments (Respective)
60-67	Executive Traps, Levels 0-7
70-117	Group 1 Interrupts, Levels 0-23

OPERATION AND CONTROL

Basic Operation

Figure 5-1 is a functional block diagram of the priority interrupt system. Both the executive traps and external interrupts are initiated by a trigger from their assigned functions. The primary operational difference between the two interrupt types is the method of control; executive traps are hardwired in an armed and enabled state, while external interrupts must be previously armed and enabled under program control before an interrupt trigger can be recognized and processed.

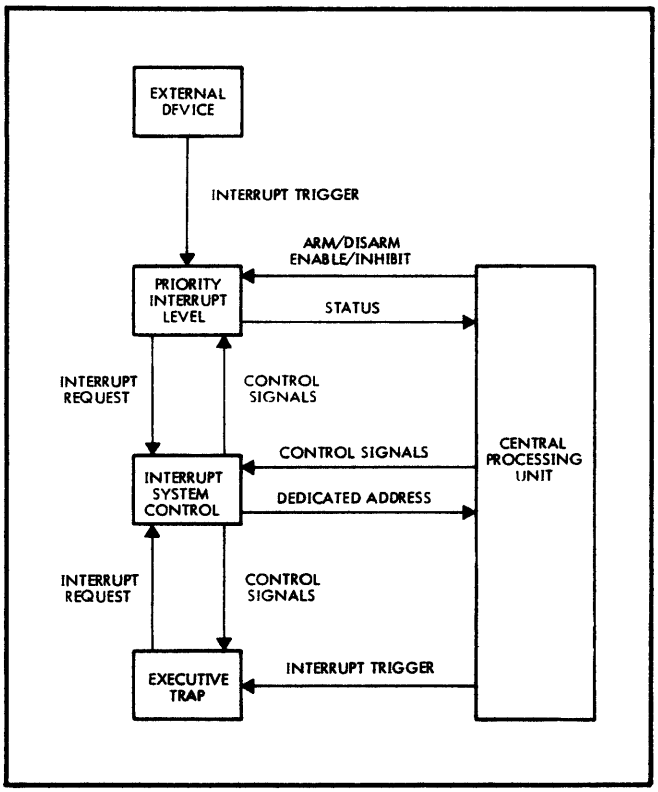


Figure 5-1. Functional Block Diagram, Priority Interrupt System

Executive Traps (Group 0)

Each executive trap interrupt is designed so as to become active immediately upon receipt of its associated internal trigger, provided no higher-priority level is active. Executive trap interrupt levels are physically integrated with their associated CPU functions, so that installation of the interrupt level is performed simultaneously with installation of the functional logic. Since executive traps are constantly armed and enabled, no program control over the activation of these interrupts is provided.

External Interrupts (Group 1)

External interrupts are program-controlled and, with the exception of Levels 0 and 1, are not permanently assigned. Program control is afforded by several instructions. Individual levels can be selectively (unitarily) armed, disarmed, enabled, or inhibited under program control. The entire group of interrupts can be simultaneously controlled. For a detailed description of all priority interrupt instructions, refer to the appropriate portion of Section VII in this manual.

Four registers are associated with the external interrupt group. These registers may each be 8, 16, or 24 bits wide, depending on the number of interrupt levels within the group. As interrupt levels are added to the system, bits are added to each of the four registers in the group. The register bit positions correspond to the priority level assignments, i.e., bit 0 represents Level 0, bit 1 represents Level 1, etc. Control of the interrupt registers is accomplished by the following group of instructions.

Transfer Double to group 1 (TD1)

Transfer group 1 to Double (T1D)

Transfer Double to group 1 (TD4 – software-triggered interrupt)

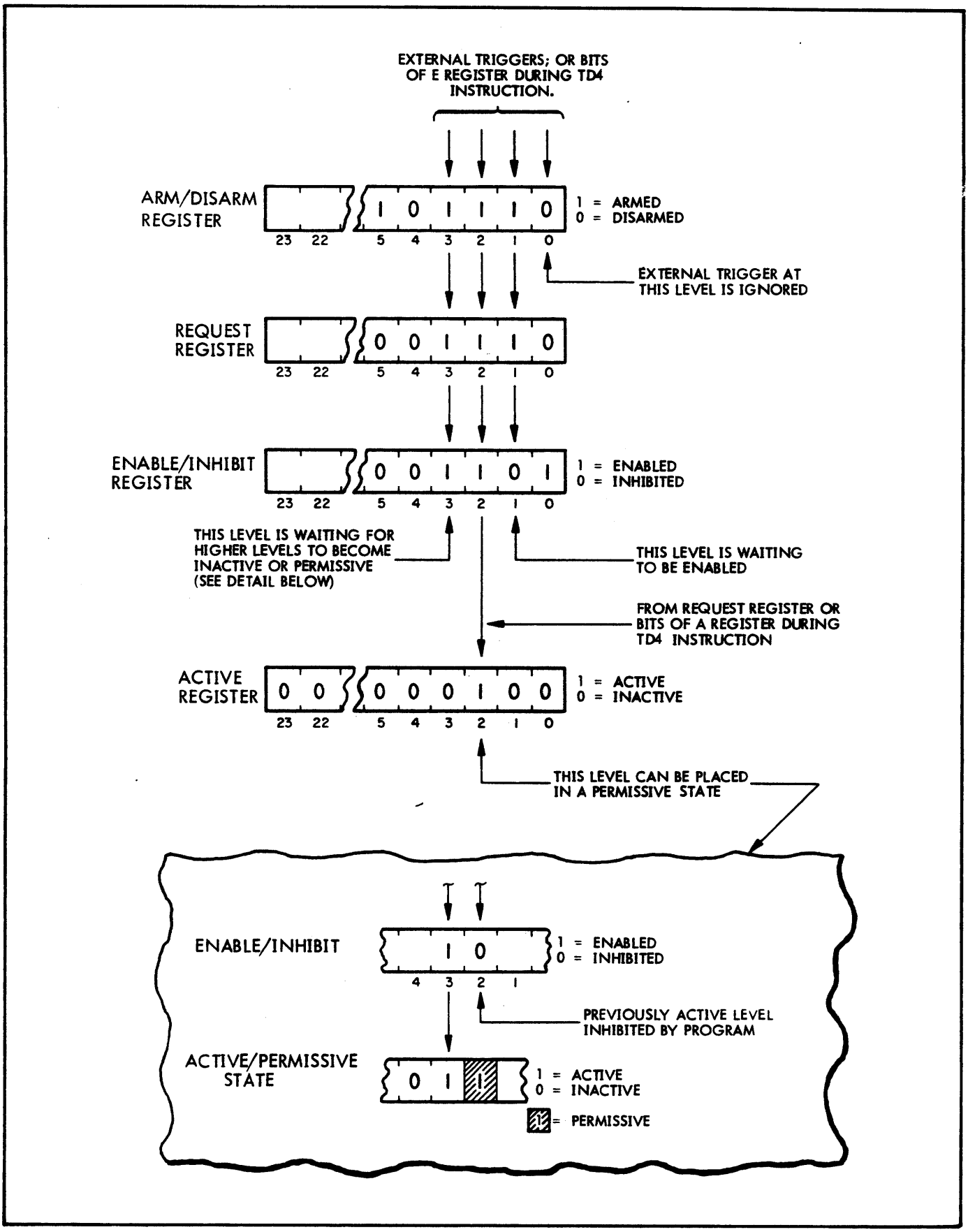
Transfer group 1 to Double (T4D – software interrupt status)

The armed/disarmed and enabled/inhibited states of each interrupt level are retained in the Arm/Disarm (A/D) and Enable/Inhibit (E/I) Registers, respectively. A TD1 instruction is used to selectively arm, disarm, enable, or inhibit individual interrupt levels within the group. Upon execution of a TD1 instruction, the contents of the E and A Registers are transferred, respectively, to the A/D and E/I Registers in Group 1. Transfers are performed in a bit-for-bit pattern. A ONE in a given bit position of the A/D Register and will cause the corresponding interrupt level to be armed; a ZERO will disarm the level. An interrupt will be enabled or inhibited by a ONE or ZERO, respectively, in the corresponding bit position of the E/I Register.

The interrupt group's armed/disarmed and enabled/inhibited status may be determined under program control by the execution of a T1D instruction. The contents of the A/D and E/I Registers are transferred to the E and A Registers, respectively. A/D and E/I Register contents are not affected by the transfer.

External interrupt triggers normally occur asynchronously with respect to CPU operation. However, interrupt triggers can be generated under program control by a TD4 instruction. The TD4 instruction performs a logical OR between the contents of the E and A Registers and the interrupt Request and Active Registers, respectively. Loading the Request Register with a ONE has the same effect as an external trigger at the corresponding interrupt level. When the Active Register is loaded with a ONE, the corresponding level will become active as long as no higher-level interrupt is active. The T4D instruction transfers the contents of the Request and Active Registers to the E and A Registers, respectively. The Request and Active Registers are not affected.

Figure 5-2 illustrates the control system for external interrupts. Each external interrupt operates in three distinct states: inactive, waiting, and active. In the inactive state, the level has not received an interrupt trigger. When a trigger is received, the armed/disarmed status determines whether the triggered interrupt will be placed in a waiting state or ignored. If the triggered interrupt is armed, it will be placed in the waiting state; if disarmed, it will be ignored.



M160-100-976B

Figure 5-2. External Interrupt Control

If an interrupt is armed but inhibited (i.e., not enabled), it is held in the waiting state until such time as it is enabled under program control. Once enabled, the interrupt will become active as soon as the current instruction is completed, assuming that no higher level is active and that external interupts are not being held (HXI instruction).

Once an interrupt becomes active, it can be inhibited under program control (TD1 instruction). This places the active level in an off-line mode or permissive state. The permissive state does not affect execution of the interrupt subroutine but enables lower priority armed and enabled interrupts to become active when triggered. For example, if active level two is inhibited by the program, waiting level three becomes active immediately. After level three is serviced, the processing of the level two subroutine is resumed until it is completed or another interrupt becomes active. Should another interrupt trigger be received by an interrupt that is in the permissive state, it will be saved and recognized when that level is returned to the on-line mode.

Hold and Release eXternal Interrupts (HXI and RXI) instructions are employed to prohibit and restore the activation of any external interrupt (other than currently-active levels) regardless of that interrupt's armed/disarmed and enabled/inhibited states. Such a prohibition would ensure that another, lower-level, interrupt could complete its processing routine without interruption. This hold condition can only be released by an RXI instruction.

Several instructions are privileged. Should an interrupt occur during the execution of one of these instructions, it will not be allowed to become active until the completion of the instruction following the privileged instruction. The privileged instructions are:

Branch and Save return — Long (BSL)
Hold interrupts and Transfer I to memory (HTI)
Hold interrupts and Transfer J to memory (HTJ)
Hold interrupts and transfer K to memory (HTK)
Release eXternal Interrupts (RXI)
EXecute Memory (EXM)
Transfer Memory to Registers (TMR)
Transfer Registers to Memory (TRM)
Update Stack Pointer (USP)
Transfer Double to group 1 (TD1)
Transfer Double to group 1 (TD4)
Unitarily Arm group 1 interrupts (UA1)
Unitarily Disarm group 1 interrupts (UD1)
Unitarily Enable group 1 interrupts (UE1)
Unitarily Inhibit group 1 interrupts (UI1)

If the virtual memory system is enabled, the following instructions are also privileged.

Transfer Double to Source and destination registers (TDS)
Transfer Source and destination registers to Double (TSD)
Transfer A to 1 virtual address Register (TAR)
Transfer Double to 2 virtual address Registers (TDR)
Transfer 2 virtual address Registers to Double (TRD)
Transfer Double to Paging Limit register (TDP)
Transfer Paging limit registers to Double (TPD)
Transfer Usage base register and demand page register to Double (TUD)
Transfer E to Usage base register (TEU)
Query virtual Usage Register (QUR)
Query Not-modified Register (QNR)
Release Operand Mode (ROM)
Release User Mode (RUM)

INTERRUPT PROCESSING CONSIDERATIONS

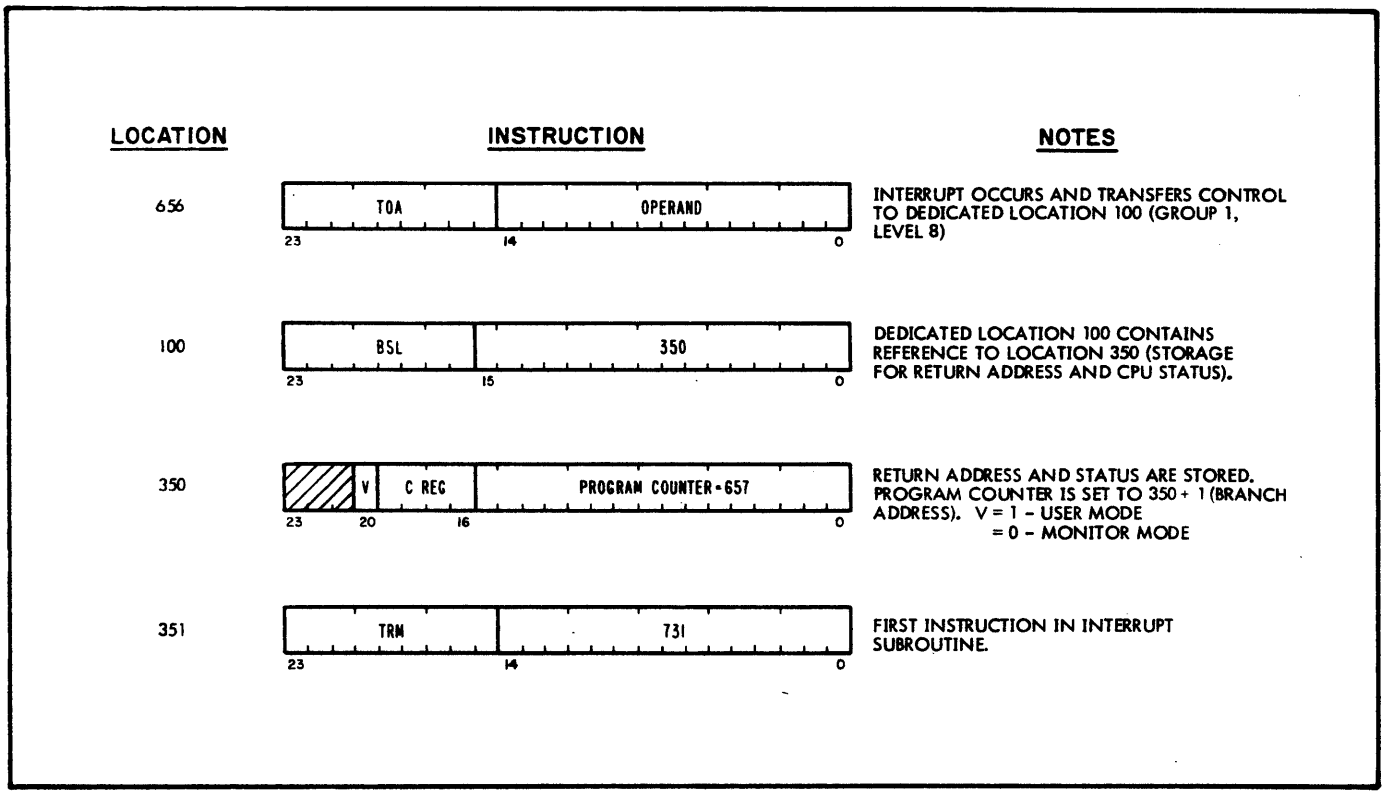
Each external interrupt and executive trap level is assigned a unique memory location as previously explained. An interrupt, when activated, generates an address and an instruction operation code. The address specifies the dedicated location and the operation code defines an EXecute Memory (EXM) instruction. The address and EXM instruction are placed in the Instruction Register, decoded, and executed as a normal operation. This causes the instruction in the dedicated location to be executed as if it were the next instruction in the main program.

Although any instruction may be stored in an interrupt's dedicated memory location, the operation designed for subroutine entry is the Branch and Save return — Long (BSL) instruction. The BSL instruction is used to enter an interrupt subroutine because it provides a means of saving machine status and returning to the program location following that being executed at the time of the interrupt. When an interrupt is generated, the current instruction is allowed to continue so the program counter can be advanced before interrupt processing begins. Figure 5-3 illustrates the sequence of events.

If virtual memory is enabled, the BSL instruction will record the paging mode (User or Monitor) in bit 20 of the effective memory address. Bit 20 will be set to ONE if the CPU was in the User Mode when the interrupt occurred, or to ZERO if the Monitor Mode was active.

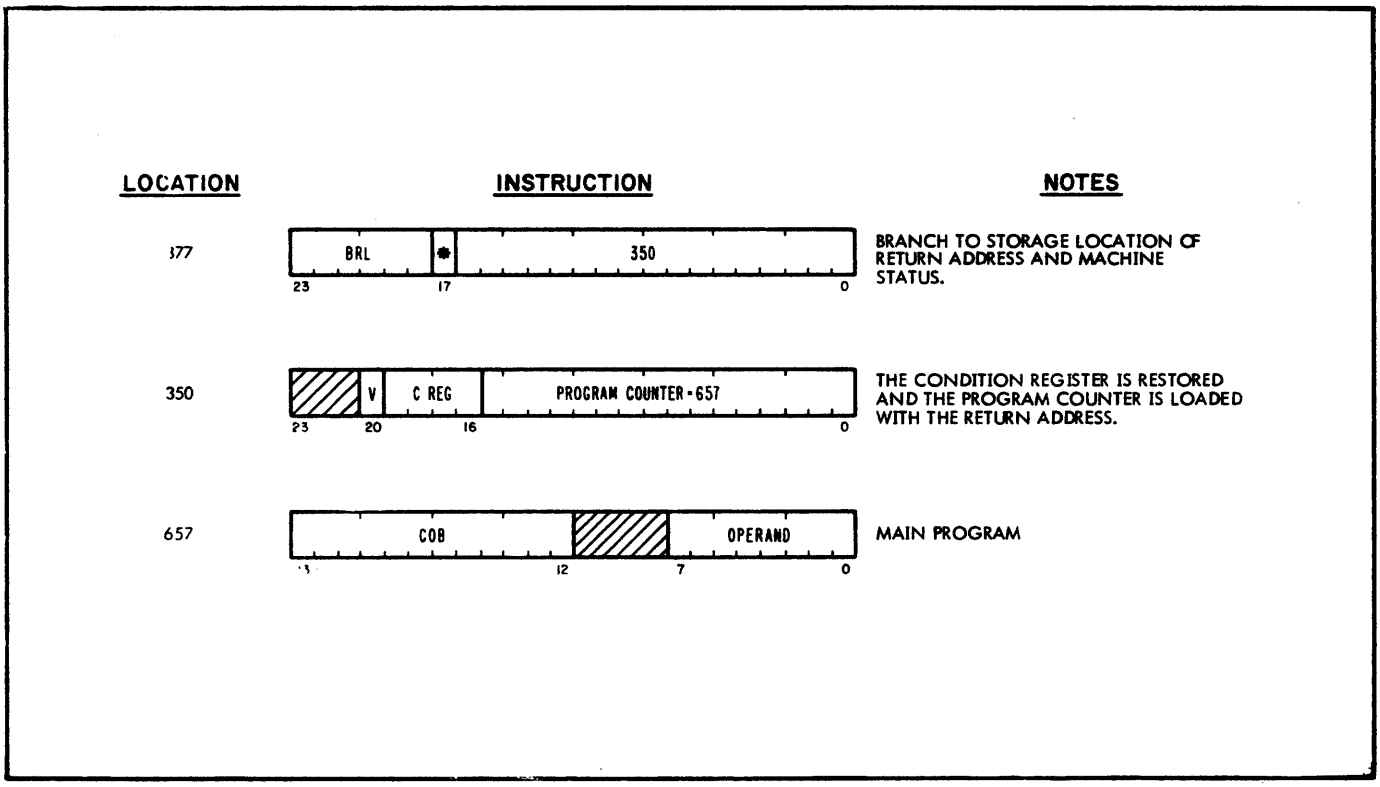
A means of exit from the interrupt routine is the Branch and Reset interrupt — Long (BRL) instruction. Normally, the BRL instruction would make use of an indirect reference (*) to the address previously referenced by the BSL instruction upon entering the routine. If this is done, the Condition Register is restored to its original contents (at the time the interrupt occurred). The state of bit 20 (in the return address) will be tested by the BRL and the appropriate virtual memory mode will be reestablished when the subsequent instruction is fetched. Figure 5-4 illustrates the subroutine exit sequence.

The BRL instruction resets the highest active (not in permissive state) trap or external interrupt level provided that external interrupts are not being "held" (HXI instruction). Active traps can only be reset by the BRL instruction. Active interrupts can only be reset by the BRL instruction, a TD1 instruction, or by master clearing the CPU. A BRL instruction will not reset an interrupt that is in the permissive state.



M160-060-9768

Figure 5-3. Interrupt Subroutine Entry



M160-157-1738

Figure 5-4 Interrupt Subroutine Exit

SECTION VI
SCIENTIFIC ARITHMETIC UNIT (SAU)

GENERAL DESCRIPTION

The optional Scientific Arithmetic Unit (SAU) provides concurrent double-precision, floating-point capability for the computer. When used with the computer, the SAU implements the execution of 47 additional instructions, or operation codes. Of these instructions, 27 permit concurrent computer/SAU operations. SAU data and condition information are displayed on the Programmer's Control Panel as a function of selectable shared indicators.

FLOATING-POINT DATA FORMAT

All arithmetic operations are carried out in double-precision format to yield a 39-bit mantissa and an 8-bit exponent. Figure 6-1 illustrates the floating-point data formats employed by the CPU's Double (D) Register, memory, and the SAU's X and XW Registers.

Data transfers to the SAU from the CPU are either single-precision integers or double-precision, floating-point, normalized numbers. All arithmetic operations performed within the SAU are executed in the double-precision, floating-point format as illustrated in Figure 6-1. Therefore, any integer number transferred to the SAU for arithmetic operations is first normalized and converted to floating-point format within the SAU. All double-precision transfers to the SAU, whether from the D Register or memory, are assumed to be normalized, floating-point quantities. Bit 23 of the least-significant half (LSH) of the double word is truncated.

SAU REGISTERS

Three SAU registers are available to the programmer. These are:

a. X Register (signed mantissa – Figure 6-1);

b. XW Register (signed exponent – Figure 6-1); and

c. Y Register (SAU condition – Figure 6-2).

The XW Register can be independently modified via the SAU instruction set. Figure 6-2 illustrates the Y (condition) Register bit configuration and their significance in reflecting the results of SAU operations.

OPERATION AND CONTROL

Data Transfers

A simplified block diagram of the SAU in relation to the CPU is shown in Figure 6-3. All data transfers between the CPU and SAU are, effectively, confined to the X, XW, and Y Registers. CPU-SAU data transfers may involve the E and A Registers or memory. The transfer source and destination are selected as a function of the instruction being executed. In all double-precision transfers to and from the SAU, the least-significant half (LSH) is transferred first. When memory is involved in the double-precision transfer, memory location N+1 (refer to Figure 6-1) must be addressed before location N in order to maintain the proper format. The CPU controls this addressing sequence as a normal instruction execution function.

SAU INSTRUCTIONS

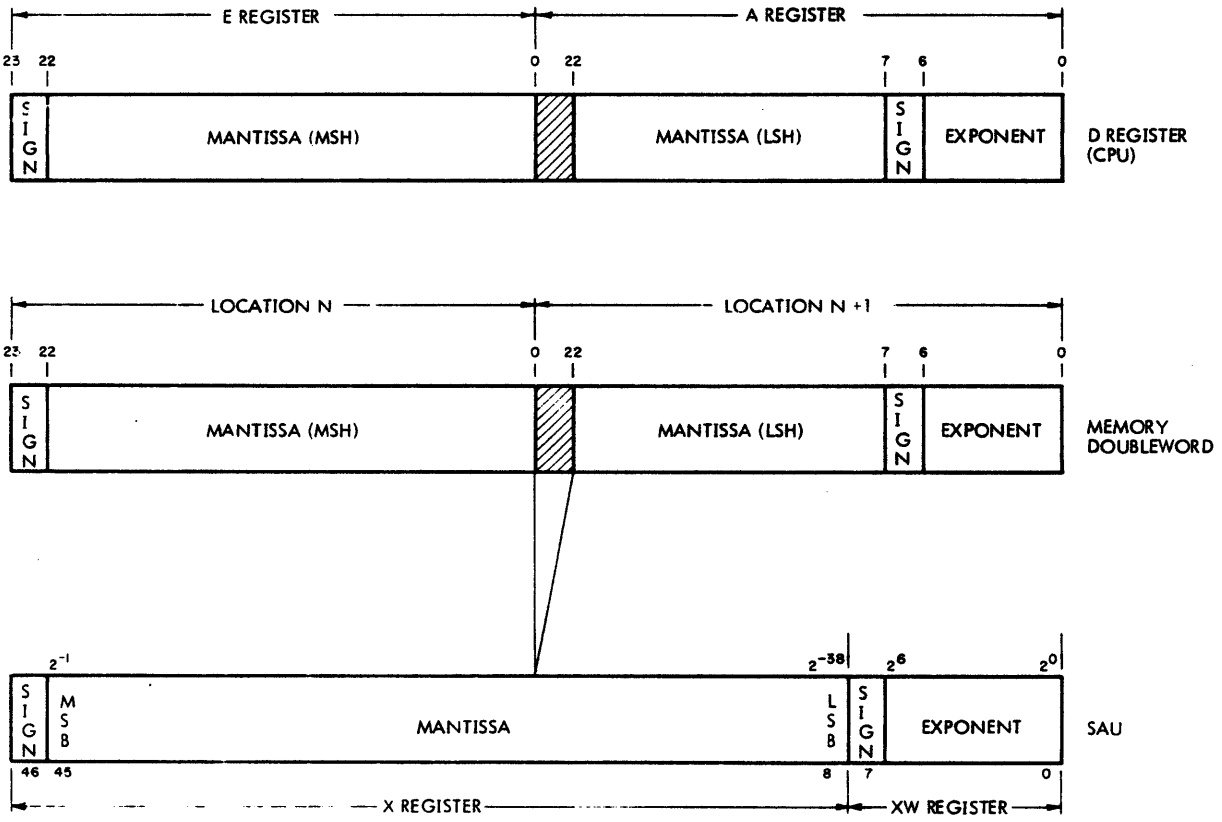
For a detailed description of SAU instructions, refer to Section VII of this manual. Appendix A shows instruction execution times and also lists the concurrent times available for processing non-SAU instructions during SAU "busy" periods.

PROGRAMMING CONSIDERATIONS

The SAU and CPU will operate concurrently for one or more microcycles, depending on the SAU instruction being executed. In order to take advantage of the available concurrent time, CPU and SAU instructions must be intermixed.

If the instruction sequence contains several consecutive SAU instructions, the CPU will wait for the SAU, i.e., if an SAU instruction is in progress and another SAU instruction follows it, the CPU must wait until the second instruction has started (or completed, if there is no time-sharing) before executing any non-SAU instruction. For example, the sequence

```
TMX   A
MMX   B    (4.7 + RS available for concurrent operation)
DMX   C    (9.6 + RS available for concurrent operation)
TXM   D
```



MI 1226-9768

Figure 6 1. Floating Point Data Formats

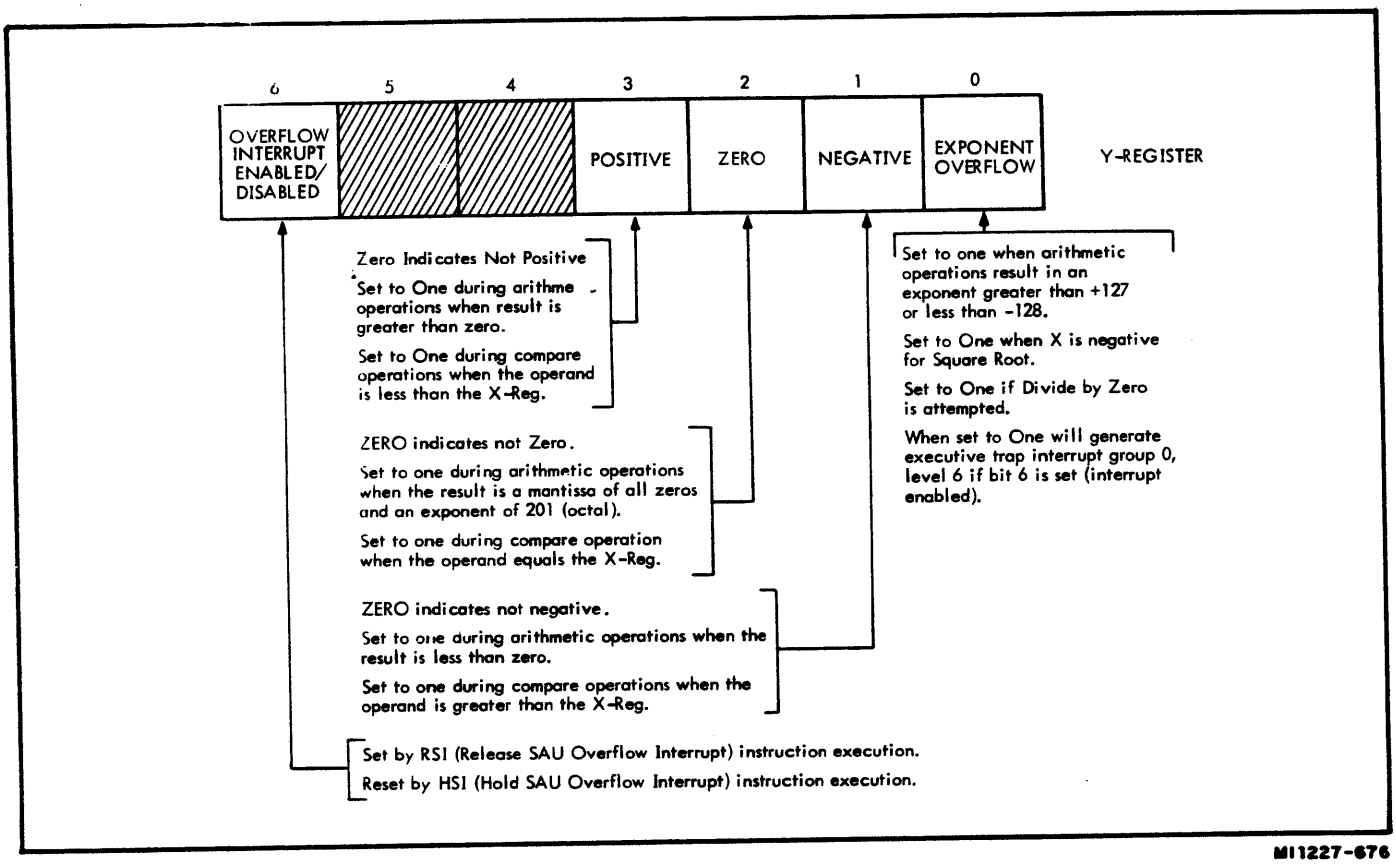


Figure 6-2. SAU Y (Condition) Register

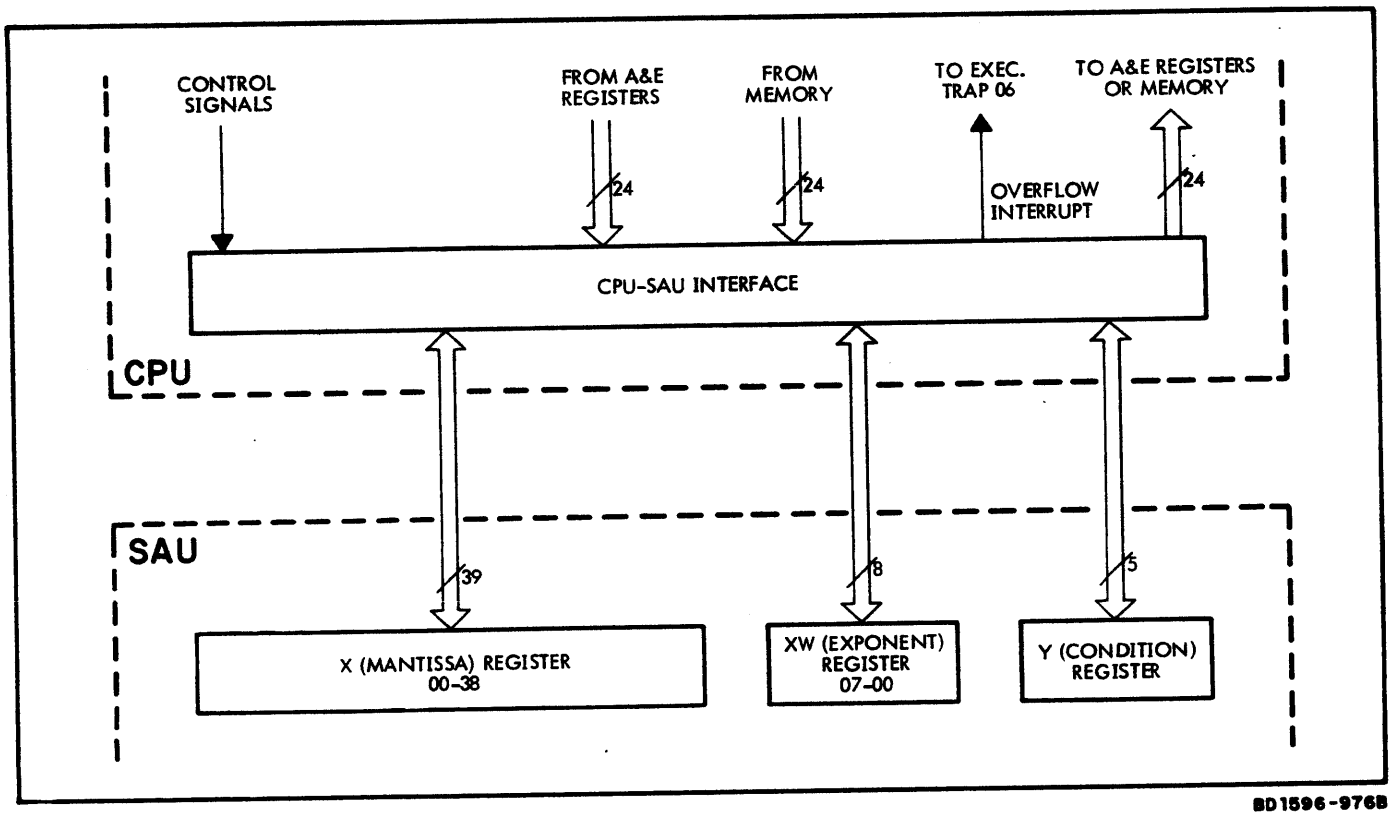


Figure 6-3. CPU - SAU Transfer Paths; Simplified Block Diagram

does not make use of the available concurrent time. Note, however, that time equal to 14.3 + 2RS is available in the sequence for executing non-SAU instructions. The following sequence makes use of the available concurrent time.

TMX	A		
MMX	B		
TMD	X	3A + 2.1	4A + 2.4 concurrent time used
TOI	30	A + 0.3	
DMX	C		
AMD	Y	3A + 2.4	10A + 6.3 + 2W concurrent time used
TDM	Z	A + 1.8 + W	
AMI	J	2A + 0.6	
TIA		A + 0.3	
NII		A + 0.3	
AAM	K	2A + 0.9 + W	
TXM	D		

SAU INTERRUPT

The executive trap (Group 0, Level 6) provided with the SAU is used to detect overflow/underflow conditions resulting from the execution of SAU instructions. The trap is controlled by two SAU instructions and the hold/release external interrupt instructions of the CPU.

The SAU instructions which control the trap are:

Hold SAU overflow Interrupt (HSI)
Release SAU overflow Interrupt (RSI)

The trap, when enabled, is triggered by the overflow bit (bit 0) of the SAU condition register (Y Register). In order to start SAU operation and enable the trap the following sequence may be used.

```
TOY     0                TMX     OPERAND
                  or
RSI                      RSI
```

Either sequence clears the overflow bit and prevents an extraneous interrupt.

When the SAU trap is enabled and an overflow occurs, the SAU is set to a busy condition, preventing the execution of any other SAU instruction except an HSI. This allows the program to determine the location of the SAU instruction which caused the overflow. The SAU interrupt processing routine must execute an HSI as its first SAU instruction. Prior to exiting the service routine, bit 0 of the Y Register must be cleared and an RSI instruction performed to rearm the SAU trap. A typical entry/exit sequence is:

```
SAUPI   ***
        HSI
        .
        .
        .
        TOY     0
        RSI
        BRL*    SAUPI
```

Note that an overflow can be caused by program control with the sequence:

```
        HSI
        RSI
        TOY     1
```

It should be noted that the contents of the Program Counter at the time of the interrupt does not necessarily have a direct relation to the location of the SAU instruction which caused the overflow. This is due to the concurrent processing capability, the occurrence of other interrupts, the execution of the HXI/RXI instructions and the way in which the SAU and CPU instructions are intermixed.

When it is a requirement to know exactly where the instruction causing the overflow is located, careful coding is mandatory if the concurrent operation capability is to be used. It is recommended that in cases where overflow is likely, the SAU instructions be written consecutively to simplify the procedure for finding which SAU instruction caused the overflow.

SECTION VII
INSTRUCTION SET

INTRODUCTION

The instruction set consists of several functional groups or families of instructions. Among these are: arithmetic; branch; compare; input/output; logical; shift; transfer; etc. Each group, in turn, is composed of individual instructions that perform specific functions.

Through the application of the instruction set, the programmer has access to each memory location and major register in the CPU. In addition, the instruction set provides for the alteration and control of program flow, manipulation and modification (arithmetic and logical) of data, servicing of priority interrupts and control of I/O operations.

INSTRUCTION FORMATS

Each instruction is decoded from a 24-bit memory word. The instruction word bits define the operation to be performed and the manner in which it is to be performed. All instruction formats contain an operation code (Op Code) that defines the general process that is to be undertaken (add, subtract, interchange, etc.). The Op Code usually contains either six or 12 bits; a few instructions require expansion of the Op Code beyond 12 bits.

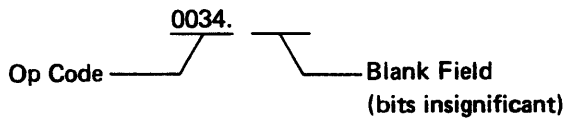
Additional bits in the instruction word specify how the general operation is to be performed. For example, when adding the contents of one register to the contents of another, the additional bits indicate which registers are involved.

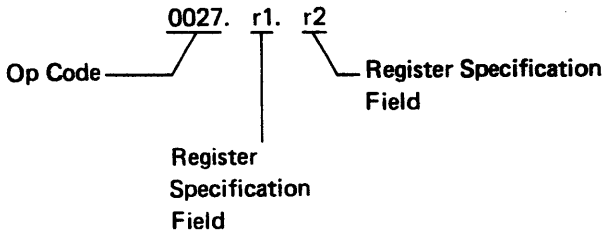
Some instructions access memory and use formats that specify an address. The address bits are sometimes supplemented by special bits (indirect, index) in the instruction word. In other cases, the additional bits are not used for address modification, but are used to define a condition under which the specified memory location will be accessed or to indicate which of the CPU registers will be used in the operation. The appropriate formats are provided with the individual instruction descriptions.

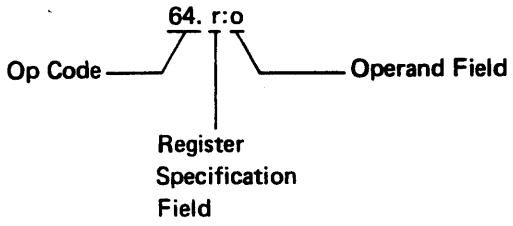
INSTRUCTION FORMULA

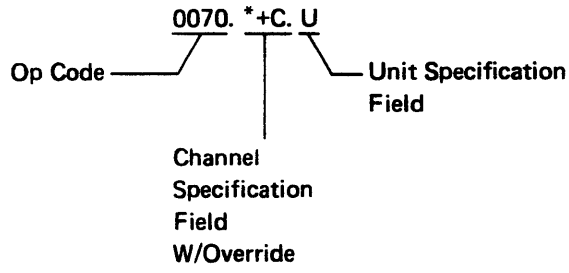
The instruction formula, presented with each instruction description, provides a graphic representation of a 24-bit instruction word. The formula expresses an instruction word as a concatenation of its various fields where each field is represented by one or more octal digits. For example, the formula 21.*+X:a expresses a memory reference branch where "21" represents a 6-bit (2 octal digits) Op Code, * and X are additive quantities defining the indirect (*) and index (X) field, and "a" is a memory reference in a 15-bit address field.

The period (.) and colon (:) provide field separation in the formula, with the colon indicating right/left justification. All digits or references to the left of the colon are left-justified, and those to the right are right-justified in their respective fields. The absence of a colon indicates that all digits or references are left-justified in their fields. Examples of instruction formulas are as follows:









INSTRUCTION DESCRIPTIONS

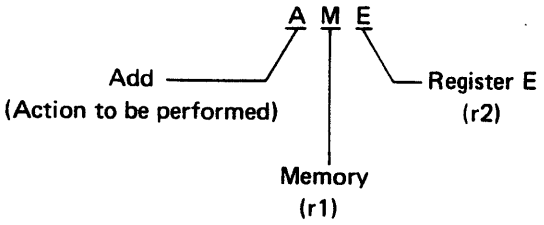
The following paragraphs describe, in detail, the various instructions. The instructions are arranged by functional groups (arithmetic, branch, compare, etc.). General information pertaining to each group is presented in the introductory paragraphs.

Each instruction description includes the three-letter mnemonic identifier, instruction name, instruction formula, and lists the registers affected. Bit assignments for each instruction are shown by means of the binary word format illustration, and a brief explanation of the instruction operation is provided. Special notes are given, where required, to complete the instruction description.

ARITHMETIC INSTRUCTIONS

The arithmetic instruction group includes the standard arithmetic operations – addition, subtraction, multiplication and division – as well as square root, normalization and sign extension instructions. Also included are several register-to-register operations which compute the absolute value, negate or round off the contents, or negate the sign of one register and subsequently transfer its contents to a second register.

The arithmetic instruction mnemonics provide a brief definition of specific operations to be performed. The first letter of the mnemonic signifies the action or type of operation to be performed, the second letter identifies the first quantity or reference (r1) to be used in the operation, and the third letter identifies the second reference (r2). For example:



In the majority of arithmetic instructions, the result of the operation remains in r2 leaving r1 unchanged (except where r1 and r2 are the same). Certain instructions – notably, those performing multiplication, division, sign extension and square root computation – do not comply with the r1 and r2 conventions stated above. These instructions are described thoroughly in the individual instruction descriptions.

Unless noted otherwise, each arithmetic operation causes the Condition (C) Register to be set reflecting the status of the result. The various arithmetic conditions are defined as follows:

a. Positive – Result is arithmetically greater than zero, indicated by a ONE in bit position 3 of the C Register. A ZERO in bit position 3 indicates "Not Positive".

b. Zero – All bits of the quantity under consideration are ZEROs, indicated by a ONE in bit position 2 of the C Register. A ZERO in bit position 2 indicates "Not Zero".

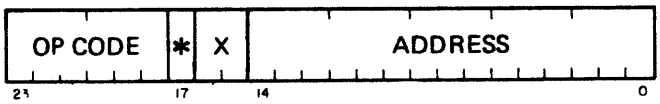
c. Negative – Result is arithmetically less than zero, indicated by a ONE in bit position 1 of the C Register. A ZERO in bit position 1 indicates "Not Negative".

d. Overflow – An Overflow results from an operation instead of displaying the status of an operand. As a general rule, an arithmetic Overflow will occur when a bit is carried into the designated sign bit position and not carried out or vice versa. An Overflow condition is indicated by a ONE in bit position 0 of the C Register. A ZERO in bit position 0 indicates "No Overflow".

The following instructions are included in the arithmetic group.

AUM Add Unity to Memory

Formula 30.*+X:a **Affected** M,C



Operation

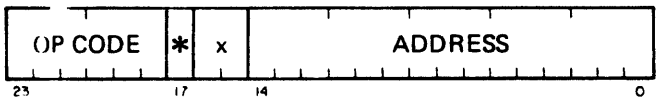
The contents of the effective memory address are incremented by one.

Note

The Condition Register is set to Positive, Negative or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out of the sign bit (23) without a carry in.

AMx Add Memory to Register

Formula 41.*+x:a **Affected** x,C



Operation

The contents of the effective memory address are algebraically added to the contents of register I, J or K.

Notes

AMx is not a computer instruction mnemonic but represents a family of instruction mnemonics. x is coded as follows to select one of the index registers.

x = 1 (I)
2 (J)
3 (K)

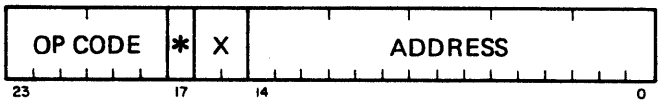
A code of 41.*+1:a, for example, implements the Add Memory to I (AMI) instruction.

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the Operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out of the sign bit (23) without a carry in.

AMA Add Memory to A

Formula 43.*+X:a **Affected** A,C



Operation

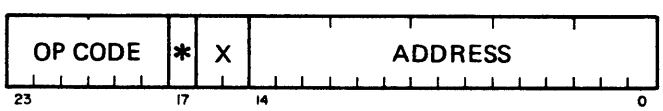
The contents of the effective memory address are algebraically added to the contents of the A Register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out of the sign bit (23) without a carry in.

AME Add Memory to E

Formula 42.*+X:a **Affected** E,C



Operation

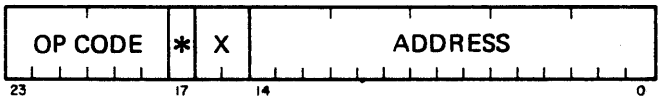
The contents of the effective memory address are algebraically added to the contents of the E Register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out of the sign bit (23) without a carry in.

AMD Add Memory to Double

Formula 44.*+X:a **Affected** E,A,C



Operation

The contents of the effective memory address (EMA) and the next sequential memory address (EMA+1) are algebraically added to the contents of the D Register according to the double integer format defined in Section II.

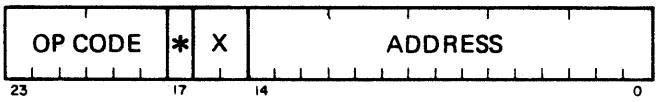
Notes

Bit A23 must be ZERO. The state of A23, after the addition of the LSH of the double words, is used to determine a carry into the MSH of the addition. If A23 is set and/or bit 23 of the LSH of the double word in memory is set prior to the addition, the carry forward will be in error.

The Condition Register is set to Positive, Negative, or Zero, based on the result in the D Register after the operation. Overflow is set if one occurs during the addition.

AMB Add Memory to Byte

Formula 45.*+X:a **Affected** A,C



Operation

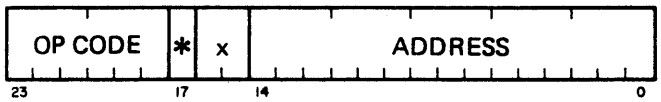
Bits 7-0 of the contents of the effective memory address are algebraically added to the contents of the B Register (A7-A0). Bits 23-8 of the A Register are unchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

AxM Add Register to Memory

Formula 46.*+x:a **Affected** M,C



Operation

The 24-bit contents of the I, J or K Register are algebraically added to the contents of the effective memory address.

Notes

AxM is not a computer instruction mnemonic but represents a family of instruction mnemonics. x is coded as follows to select one of the index registers.

x = 1 (I)
2 (J)
3 (K)

A code of 46.*+2:a, for example, implements the add J to Memory (AJM) instruction.

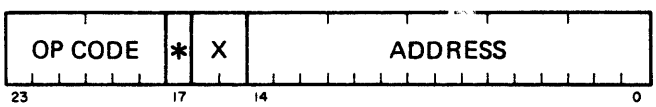
The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
      AJM*    X
       .
       .
       .
X     DAC     Y,K
```

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

AAM Add A to Memory

Formula 50.*+X:a **Affected** M,C



Operation

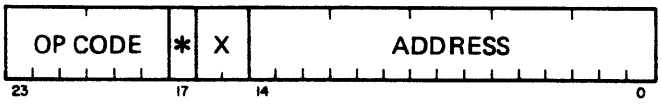
The contents of the A Register are algebraically added to the contents of the effective memory address.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

AEM Add E to Memory

Formula 47.*+X:a **Affected** M,C



Operation

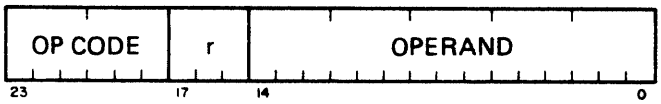
The contents of the E Register are algebraically added to the contents of the effective memory address.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

AOr Add Operand to Register

Formula 64.r:o **Affected** r,C



Operation

The 15-bit unsigned operand is algebraically added to the contents of the specified register.

Notes

AOr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r is coded as follows to select any of the general purpose registers.

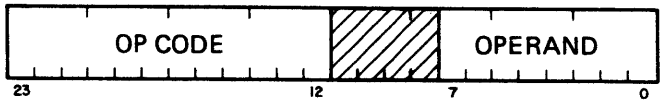
r = 1 (I)
2 (J)
3 (K)
4 (E)
5 (A)
6 (T)

A code of 64.3:o, for example, implements the Add Operand to K (AOK) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry out of the sign bit without a carry in, or a carry out without a carry in.

AOB Add Operand to Byte

Formula 0012:o **Affected** A,C



Operation

The 8-bit signed operand is algebraically added to the contents of the B Register (A7-A0). Bits 23-8 of the A Register are unchanged.

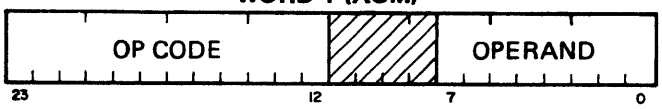
Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

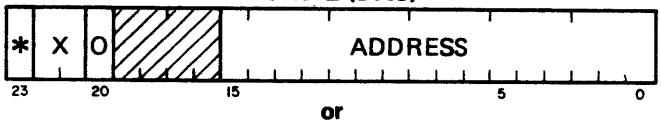
AOM(n) DAC(m) or AOM(n) LAC(m) Add Operand to Memory

Formula 0074:o (word 1) **Affected** M,C
*+X.O:A or *+X.1:A (word 2)

WORD 1 (AOM)

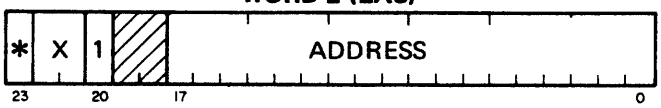


WORD 2 (DAC)



or

WORD 2 (LAC)



Operation

The 8-bit signed operand (n) is algebraically added to the contents of the effective memory address (m).

Notes

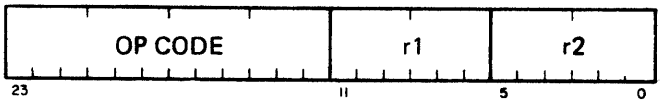
If a demand page, restrict mode violation, or limit violation occurs when attempting to access the effective memory address while in the virtual memory User mode, the Program Counter will be decremented by one. If the violation occurs during the fetch of the second word, the Program Counter will be decremented by one.

An AOM instruction may not be used after a ROM instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

Arr Add Register to Register

Formula 0020.r1.r2 **Affected** r,C



Operation

The contents of r1 are algebraically added to the contents of r2.

Notes

Arr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select one of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

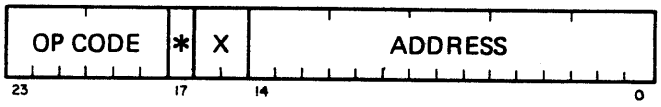
A code of 0020.10.40, for example, implements the Add E to T (AET) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1 or r2, they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 registers. Affected registers are only those selected in group r2.

DVM Divide by Memory

Formula 57.*+X:a **Affected** E,A,C



Operation

A23 is cleared and the double-precision contents of the D Register (E and A) are algebraically divided by the single-precision contents of the effective memory address. The signed, single-precision, quotient is left in A and the remainder is left in E. The remainder will have the same sign as the original dividend and the Condition Register will be set according to the status of the quotient.

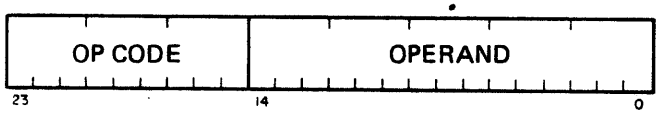
Notes

If it is desired to divide a single–precision number in A by memory, an Extend Sign of A (ESA) instruction should be executed prior to the DVM. This will establish the proper format for the dividend.

If the contents of E are equal to, or greater than, the contents of memory, an Overflow condition will result and the Condition Register will be set accordingly.

DVO Divide by Operand

Formula 610:o **Affected** E,A,C



Operation

A23 is cleared and the double-precision contents of the D Register (E and A) are algebraically divided by the 15-bit unsigned operand. The signed, single-precision, quotient is left in A and the remainder is left in E. The remainder will have the same sign as the original dividend and the Condition Register will be set according to the status of the quotient.

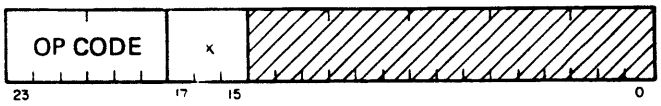
Notes

If it is desired to divide a single-precision number in A by the operand, an Extend Sign of A (ESA) instruction should be executed prior to the DVO. This will establish the proper format for the dividend.

If the contents of E are equal to, or greater than, the operand, an Overflow condition will result and the Condition Register will be set accordingly.

DVx Divide by Register

Formula 61.x **Affected** E,A,C



Operation

A23 is cleared and the double-precision contents of the D Register (E and A) are algebraically divided by the specified register. The signed, single-precision, quotient is left in A and the remainder is left in E. The remainder will have the same sign as the original dividend and the Condition Register will be set according to the status of the quotient.

Notes

DVx is not a computer instruction mnemonic but represents a family of instruction mnemonics. x is coded as follows to select one of the index registers.

x = 1 (I)
2 (J)
3 (K)

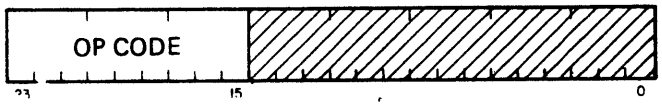
A code of 61.1, for example, implements the Divide by I (DVI) instruction.

If it is desired to divide a single-precision number in A by the contents of the specified register, and Extend Sign of A (ESA) instruction should be executed prior to the divide instruction. This will establish the proper format for the dividend.

If the contents of E are equal to, or greater than, the contents of the specified register, an Overflow condition will result and the Condition Register will be set accordingly.

DVT Divide by T

Formula 616. **Affected** E,A,C



Operation

A23 is cleared and the double-precision contents of the D Register (E and A) are algebraically divided by the T Register. The signed, single-precision, quotient is left in A and the remainder is left in E. The remainder will have the same sign as the original dividend and the Condition Register will be set according to the status of the quotient.

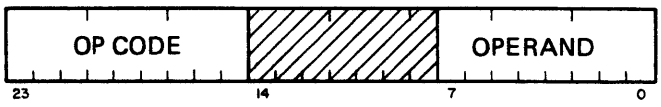
Notes

If it is desired to divide a single-precision number in A by the contents of the T Register, an Extend Sign of A (ESA) instruction should be executed prior to the divide instruction. This will establish the proper format for the dividend.

If the contents of E are equal to, or greater than, the contents of the specified register, an Overflow condition will result and the Condition Register will be set accordingly.

DV2 Divide by 2

Formula 615:o **Affected** E



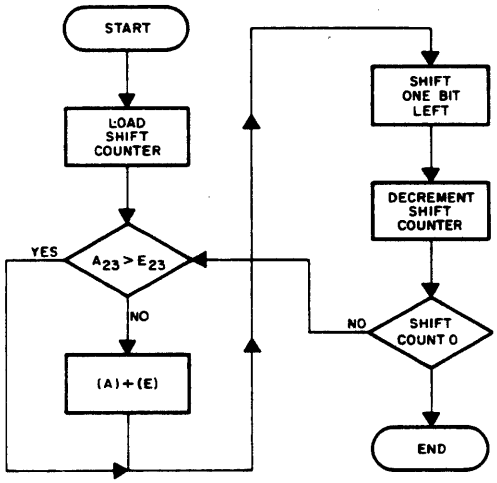
Operation

The DV2 instruction divides the contents of the E Register by the contents of the A Register, except that the arithmetic operation will be Modulo 2 (exclusive OR) instead of 2's complement arithmetic. The 8-bit operand contained in the instruction specifies the number of shifts.

Notes

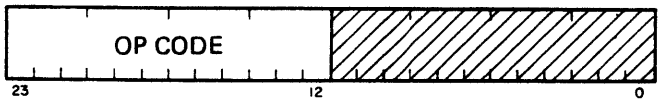
The specified number of shifts must be an even number and cannot be zero. If zero shifts are specified, the operation is the same as when a shift of one (1) is specified.

This instruction is used for generating and checking error codes based on polynomial coding techniques. The polynomial and the operand to be implemented must be left-justified in the A and E Registers. The result will be placed in the E Register while the polynomial will remain in the A Register.



ESA Extend Sign of A

Formula 0037. **Affected** E,C,A

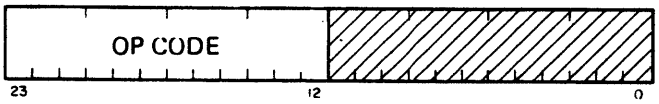


Operation

The state of the sign bit (A23) of the A Register is copied into all 24 positions of the E Register and bit A23 is then set to zero. This forms a double-precision number in E and A.

ESB Extend Sign of Byte

Formula 0010. **Affected** A,C



Operation

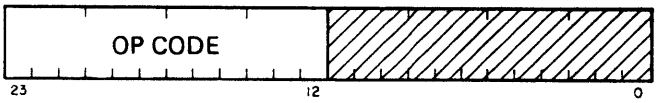
The state of the register B sign bit (A7) is copied into bit positions A8-A23, forming a sign extension of the byte.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in A at the completion of the operation.

FNO Floating Normalize

Formula 0054. **Affected** E,A,I,C



Operation

The contents of the D Register (E and A) are shifted left arithmetically until bit E22 differs from E23. The negative shift count (i.e., the number of shifts performed) replaces the contents of the I Register.

Notes

Example: Convert a double-precision integer in D to double-precision floating point format.

```
TOC     0       Clear Overflow
FNO             Normalize
TIB             Position exponent in byte
                (A7-A0).
BOZ     *+2     If result is zero, no exponent
                adjustment is necessary.
AOB     46      Adjust shift count
```

There are four special cases where the shifting process differs from that described above.

If the binary pattern 11000...0 is detected in register D, normalization is terminated to avoid creating the invalid pattern 10000...0.

If the invalid binary pattern 10000...0 is detected, it is shifted right one position producing the pattern 11000...0. The shift count is adjusted accordingly.

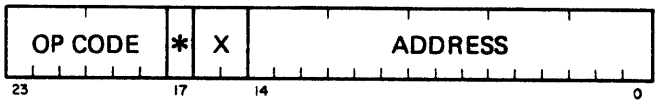
If the pattern 00000...0 is detected, the shift count is set to -177$_8$, making a zero less significant than any other value.

If an Overflow condition is present when beginning the operation, the contents of the D Register are arithmetically shifted right one position. The shift count is set to ONE and the sign of D is complemented.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

MYM Multiply by Memory

Formula 56.*+X:a **Affected** E,A,C



Operation

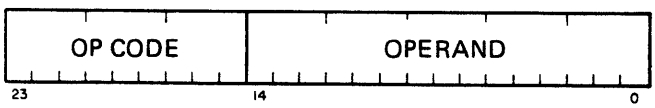
The contents of the A Register are algebraically multiplied by the contents of the effective memory address. The double-precision product replaces the previous contents of the D Register (E and A).

Note

An Overflow will result if the full-scale negative number (1000....00) is used as both the multiplier and multiplicand.

MYO Multiply by Operand

Formula 600:o **Affected** E,A,C

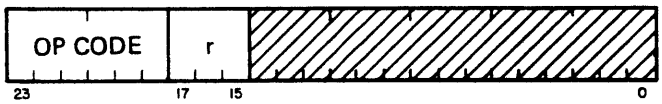


Operation

The contents of the A Register are algebraically multiplied by the 15-bit unsigned operand in the instruction word. The double-precision product replaces the previous contents of the D Register (E and A).

MYr Multiply by Register

Formula 60.r **Affected** E,A,C



Operation

The contents of the A Register are algebraically multiplied by the contents of the specified register. The double-precision product replaces the previous contents of the D Register (E and A).

Notes

MYr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r is coded as follows to select one of the general purpose registers.

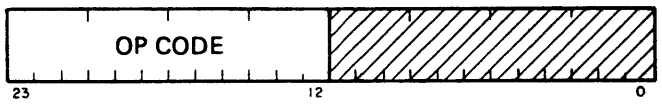
r = 1 (I)
2 (J)
3 (K)
4 (E)
5 (A)
6 (T)

A code of 60.4, for example, implements the Multiply by E (MYE) instruction.

An Overflow will result if the full-scale negative number (1000....00) is used as both the multiplier and multiplicand.

NBB Negate of Byte to Byte

Formula 0005. **Affected** A,C



Operation

The contents of the B Register (A7-A0) are two's complemented. Bit positions A23-A8 are unchanged.

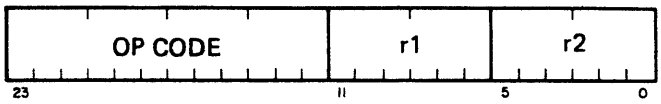
Notes

An Overflow will result when negating 2^7 (full-scale negative byte).

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

Nrr Negate of Register to Register

Formula 0022.r1.r2 **Affected** r,C



Operation

The two's complement of the contents of r1 replace the previous contents of r2.

Notes

Nrr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select any of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

A code of 0022.40.01, for example, implements the Negate of T to I (NTI) instruction.

An Overflow will result when negating 2^{23} (full-scale negative number).

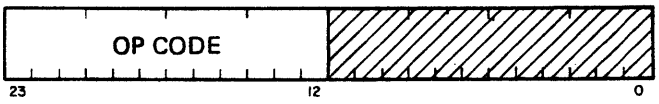
The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1 or r2, they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 registers. Affected registers are only those selected in group r2.

If the Timer (T) Register is selected as source or destination, the instruction is treated as a multiple register instruction for timing.

NDD Negate of Double to Double

Formula 0033. **Affected** E,A,C



Operation

The contents of the D Register (E and A), in double-precision format, are two's complemented.

Notes

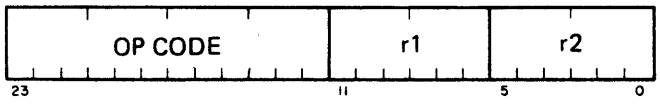
An Overflow will result when negating 2^{46} (full-scale negative double integer).

Bit A23 is copied into the carry flip-flop after the first half of the double word is added. If A23 or bit 23 of the LSH of the double word is set, a carry may be lost or added.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

NSr Negate Sign of Register

Formula 0032.r1.r2 **Affected** r,C



Operation

The sign bit of the specified register is complemented.

Notes

NSr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select one of the general purpose registers.

r1 and r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

A code of 0032.01.01, for example, implements the Negate Sign of I (NSI) instruction.

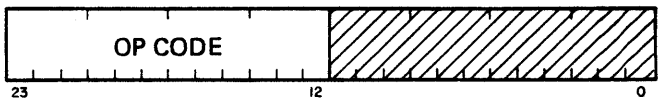
An Overflow will result when negating zero to create a full-scale negative.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1 or r2, they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 registers. Affected registers are those selected in group r2 and the Condition Register.

PBB Positive of Byte to Byte

Formula 0006. **Affected** A,C



Operation

The absolute value of the contents of the B Register (A7-A0) is placed in the B Register.

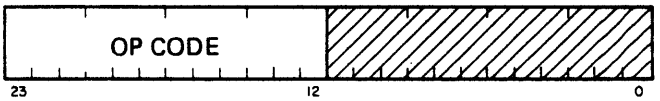
Notes

An Overflow will result when negating a full scale negative byte.

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

PDD Positive of Double to Double

Formula 0034. **Affected** E,A,C



Operation

The absolute value of the contents of the D Register is placed in the D Register according to the double integer format defined in Section II.

Notes

An Overflow will result when negating a full scale negative.

According to the double integer format, A is cleared by this instruction execution.

Bit A23 is copied into the carry flip-flop after the first half of the double word is added. If A23 or bit 23 of the LSH of the double word is set, a carry may be lost or added.

A code of 51.*+1:a, for example, implements the Subtract Memory from I (SMI) instruction.

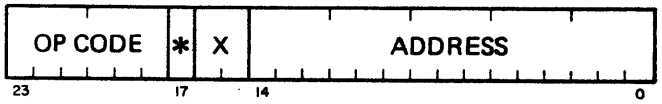
The immediate memory reference cannot be indexed; however, indexing of indirect reference is permitted, e.g.,

```
        SMI*    X
         .
         .
         .
X       DAC     Y,J
```

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

SMA Subtract Memory from A

Formula 53.*+X:a **Affected** A,C



Operation

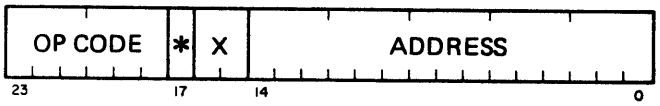
The contents of the effective memory address are algebraically subtracted from the contents of the A Register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

SME Subtract Memory from E

Formula 52.*+X:a **Affected** E,C



Operation

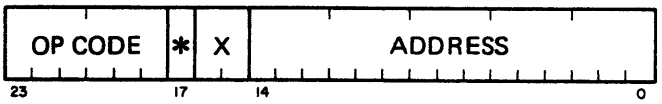
The contents of the effective memory address are algebraically subtracted from the contents of the E Register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

SMD Subtract Memory from Double

Formula 54.*+X:a **Affected** E,A,C



Operation

The contents of the effective memory address (EMA) and the next sequential address (EMA+1) are algebraically subtracted from the contents of the D Register (E and A), according to the double integer format defined in Section II.

Notes

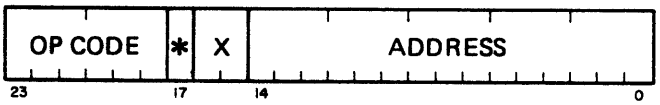
Failure to adhere to the double integer format will provide incorrect results. Bits A23 must be ZERO. (A carry or borrow may be lost between the E and A Registers.)

Bit A23 is copied into the carry flip-flop after the first half of the double word is added. If A23 or bit 23 of LSH of the double word in memory is set, a carry may be lost or added.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

SMB Subtract Memory from Byte

Formula 55.*+X:a **Affected** A,C



Operation

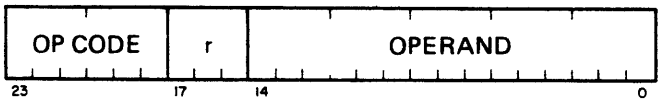
The contents of bits 7-0 of the effective memory address are algebraically subtracted from the B Register (A7-A0). Bits A23-A8 are unaffected.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

SOr Subtract Operand from Register

Formula 65.r:o **Affected** r,C



Operation

The 15-bit unsigned operand is algebraically subtracted from the contents of the specified register.

Notes

SOr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r is coded as follows to select one of the general purpose registers.

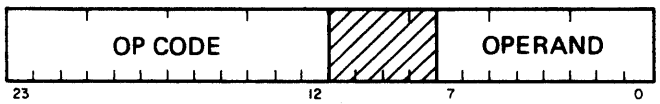
r = 1 (I)
2 (J)
3 (K)
4 (E)
5 (A)
6 (T)

A code of 65.1:o, for example, implements the Subtract Operand from I (SOI) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the result of the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

SOB Subtract Operand from Byte

Formula 0013:o **Affected** A,C



Operation

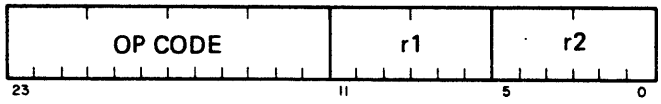
The 8-bit signed operand is algebraically subtracted from the contents of the B Register (A7-A0). Bits A23-A8 are unaffected.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

Srr Subtract Register from Register

Formula 0021.r1.r2 **Affected** r2,C



Operation

The contents of r1 are algebraically subtracted from r2.

Notes

Srr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select any of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

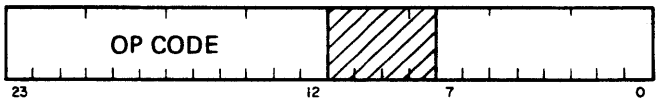
A code of 0020.01.02, for example, implements the Subtract I from J (SIJ) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1 or r2, they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 registers.

SRT Square Root

Formula 0076:014 **Affected** E,A,C



Operation

The contents of the A Register are treated as a 23-bit positive integer. The square root of this quantity is placed in the A Register, right justified, and the remainder is placed in the E Register so that:

$root^2$ + remainder = original integer.

Notes

If the sign bit (23) of the A Register is set, the Condition Register will be set to Overflow.

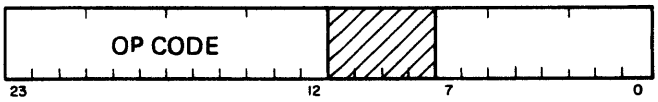
SRT generates a root of 12 significant bits; i.e., the true integer root of any positive integer in the A Register.

Consider the following examples where An implies a binary point to the right of bit n.

Positive Integer	Root (Octal)
2 at A0	1 at A0
2 at A20	1.3240 at A10

SRE Square Root Extended

Formula 0076:027 **Affected** E,A,C



Operation

The contents of the A Register are treated as a 23-bit positive integer. The square root of this quantity is placed in the A Register, right justified, and the remainder is placed in the E Register so that:

$$root^2 + remainder = original\ integer.$$

Notes

If the sign bit (23) of the A Register is set, the Condition Register will be set to Overflow.

SRE generates a root of 23 significant bits. This extended significance is obtained by assuming 22 zeros to the right of bit A0; effectively multiplying the contents of A by 2^{22} and, consequently, the root by 2^{11}

Consider the following examples where: An implies a binary point in the right of bit n.

Positive Integer	Root (Octal)
2 at A0	1.3240 at A11
2 at A20	1.3240474 at A21

BRANCH INSTRUCTIONS

The branch group of instructions can be divided into two basic types; conditional and unconditional branches. Conditional branches cause control to be transferred to a specified address upon detection of a certain machine condition as indicated by the contents of the Condition Register. Unconditional branches cause control to be transferred unconditionally to a specified address.

When virtual memory is disabled, or enabled and in the Monitor Mode of operation, only long branch (BJL, BLL, BRL, BSL, BUL) instructions or the Branch and Link – Unrestricted (BLU) instruction should be used in the last location of the lower or upper 32K sections of memory. Use of any other branch instruction will cause control to be automatically transferred to the opposite memory section.

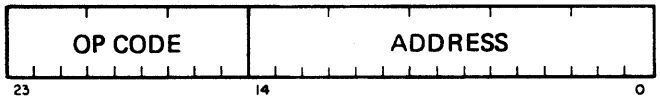
Caution should be observed when employing branch instructions in conjunction with the virtual memory system. When a Release Operand Mode (ROM) instruction is executed, any branch instruction following the ROM will cause the User Mode to be established. If the instruction is a conditional branch, the User Mode will be established regardless of the outcome of the conditional test. (SAU conditional branches are exceptions to this rule.) A BLU instruction automatically establishes the Monitor Mode.

The following instructions are included in the branch group.

BBI	Branch When Byte Address +1 in I ≠ 0	7-14
BBJ	Branch When Byte Address +1 in J ≠ 0	7-15
BJL	Branch Indexed by J Long	7-16
BLL	Branch and Link (J) Long	7-17
BLU	Branch and Link Unrestricted	7-18
BLx	Branch and Link Register	7-17
BOc	Branch on Condition Code	7-16
BRL	Branch and Reset Interrupt Long	7-18
BSL	Branch and Save Return Long	7-17
BUC	Branch Unconditionally	7-16
BUL	Branch Unconditionally Long	7-16
BWx	Branch When Register +1 ≠ 0	7-16

BBI Branch when Byte Address +1 in I ≠0

Formula 607:a **Affected** I



Operation

The contents of bits 22 and 23 of the I Register are incremented by one. If the result of this addition (in bits 22 and 23) is not 00_2 then the contents of the P Register (current program address) are replaced by the 15-bit effective memory address. If the result of the addition to bits 22 and 23 is 00_2 then bits 22 and 23 are set to 01_2 and bits 21-0 are incremented by one. If the resultant sum in bits 21-0 is zero, then the P Register advances to the next sequential program location and the index register is set to

20000000_8 Otherwise, the contents of the P Register are replaced by the 15-bit effective memory address.

Notes

In general, the BBI and BBJ instructions are used as special index register increments in order to sequentially reference consecutive bytes in memory via the EMB and RBM instructions. Consider the following examp' which will move 11 consecutive bytes starting from the third byte at location '200 to the first byte at location '300.

```
TMJ     = '60000200
I MI    = '20000300
TNK     11
EMB     0
RBM     0
BBI     *+1
BBJ     *+1
BWK     *-4
```

Occasionally, it is possible to use the address of a portion of the I Register as a byte counter as well as a word pointer. This may be illustrated by the following example which will set the buffer to blanks, starting at byte 3 of location '100 through byte 3 of location '102.

```
TOB     "ø"
TMI     = '77777775     bits 22 and 23 = 3,
                        bits 21-0 = -3
RBM     '100+3
BBI     *-1
```

However, it should be noted this technique of using the index register as both a byte counter and word pointer may be used only in certain instances. Specifically, when the following relationship is true.

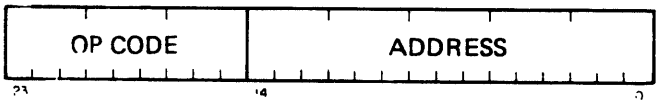
$$R\left(\frac{4\text{-B.n.}}{3}\right) = R\left(\frac{CT}{3}\right)$$

Where:

R ( ) = remainder
b.n. = the starting byte number (1,2, or 3)
CT = The number of bytes to be referenced.

BBJ Branch when Byte Address +1 in J ≠ 0

Formula 617:a **Affected** J



Operation

The contents of bits 22 and 23 of the J Register are incremented by one. If the result of this addition (in bits 22 and 23) is not 00_2 then the contents of the P Register (current program address) are replaced by the 15-bit effective memory address. If the result of the addition to bits 22 and 23 is 00_2 then bits 22 and 23 are set to 01_2 and bits 21-0 are incremented by one. If the resultant sum in bits 21-0 is zero, then the P Register advances to the next sequential program location and the index register is set to 20000000_8 Otherwise, the contents of the P Register are replaced by the 15-bit effective memory address.

Notes

In general, the BBI and BBJ instructions are used as special index register increments in order to sequentially reference consecutive bytes in memory via the EMB and RBM instructions. Consider the following example which will move 11 consecutive bytes starting from the third byte at location '200 to the first byte at location '300.

```
TMJ     = '60000200
TMI     = '20000300
TNK     11
EMB     0
RBM     0
BBI     *+1
BBJ     *+1
BWK     *-4
```

Occasionally, it is possible to use the address of a portion of the J Register as a byte counter as well as a word pointer. This may be illustrated by the following example which will set the buffer to blanks, starting at byte 3 of location '100 through byte 3 of location '102.

```
TOB     "ø"
TMI     = '77777775     bits 22 and 23 = 3,
                        bits 21-0 = -3
RBM     '100+3
BBJ     *-1
```

However, it should be noted this technique of using the index register as both a byte counter and word pointer may be used only in certain instances. Specifically, when the following relationship is true.

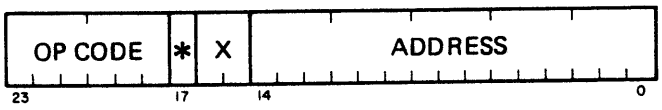
$$R\left(\frac{4\text{-B.n.}}{3}\right) = R\left(\frac{CT}{3}\right)$$

Where:

R ( ) = remainder
b.n. = the starting byte number (1,2 or 3)
CT = The number of bytes to be referenced.

BUC Branch Unconditionally

Formula 21.*+X:a **Affected** P

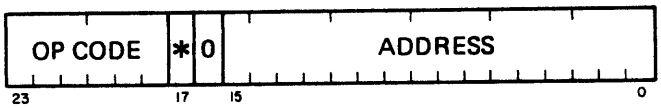


Operation

The contents of the P Register (current program address) are replaced by the 15-bit effective memory address.

BUL Branch Unconditionally Long

Formula 26.*+0:A **Affected** P



Operation

The contents of the P Register (current program address) are replaced by the 16-bit effective memory address.

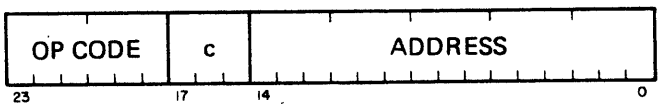
Note

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
      BUL*   X
       .
       .
       .
X     DAC    Y,I
```

BOc Branch on Condition Code

Formula 22.c:a **Affected** P



Operation

The contents of the Condition Register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the 16-bit effective memory address. If the specified condition is not present, the program advances to the next sequential location (program address +1).

Note

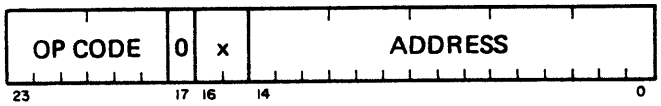
BOc is not a computer instruction mnemonic but represents a family of instruction mnemonics. c is coded as follows to select the branch on condition.

c = 0 (Overflow)
1 (Negative)
2 (Zero)
3 (Positive)
4 (No Overflow)
5 (Not Negative)
6 (Not Zero)
7 (Not Positive)

A code of 22.1:a, for example, implements the Branch on Negative (BON) instruction.

BWx Branch When Register +1 ≠0

Formula 23.x:a **Affected** x,P



Operation

The contents of the specified register are incremented by one and then tested for zero. If the contents are not zero, the contents of the P Register (current program address) are replaced by the 16-bit effective memory address. If the contents are zero, the program advances to the next sequential location (program address +1).

Note

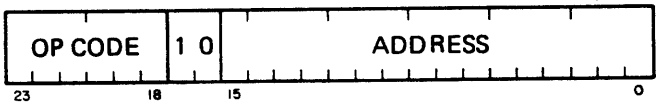
BWx is not a computer instruction mnemonic but represents a family of instruction mnemonics. x is coded as follows to select one of the index registers.

x = 1 (I)
2 (J)
3 (K)

A code of 23.1:a, for example, implements the Branch When I+1≠0 (BWI) instruction.

BJL Branch Indexed by J Long

Formula 23.4:A **Affected** P



Operation

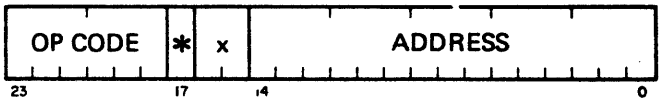
The contents of the P Register (current program address) are replaced by the 16-bit effective memory address.

Note

The immediate memory reference is automatically indexed by J.

BLx Branch and Link Register

Formula 24.*+x:a **Affected** x,P



Operation

The contents of the I, J or K Register are replaced by the next sequential address (program address +1) and the contents of the P Register (current program address) are replaced by the 15-bit effective memory address.

Notes

BLx is not a computer instruction mnemonic but represents a family of instruction mnemonics. x is coded as follows to select one of the index registers.

x = 1 (I)
2 (J)
3 (K)

A code of 24.*+1:a, for example, implements the Branch and Link I (BLI) instruction.

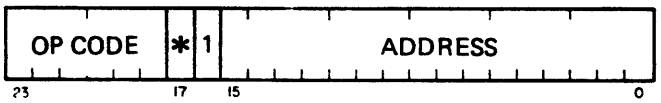
On an indirect or index operation, the specified register is loaded with the contents of the P Register (address of next sequential instruction) before indexing or indirection takes place.

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
BLI*    X
 .
 .
 .
X       DAC     Y,J
```

BLL Branch and Link (J) Long

Formula 26.*+2:A **Affected** J,P



Operation

The contents of the J Register are replaced by the next sequential address (program address +1) and the contents of the P Register (current program address) are replaced by the 16-bit effective memory address.

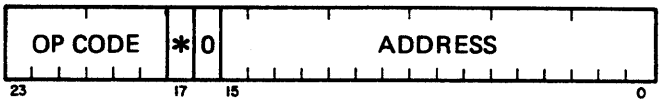
Note

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
BLL*
 .
 .
 .
X       DAC
```

BSL Branch and Save Return Long

Formula 25.*+0:A **Affected** P



Operation

The next sequential address (program address +1), along with the contents of the Condition Register are stored in the 16-bit effective memory address (EMA). The contents of the P Register (current program address) are then replaced by the address following the effective memory address (EMA +1).

Notes

This instruction is used to enter an interrupt subroutine because it provides a means of returning to the main program at the point of interrupt and saves the machine status (condition) at the time of the interrupt.

The contents of the Condition Register are stored in bit positions 19-16 of the EMA and the return address (program address +1) is stored in bits 15-0. The remaining bits are set to ZEROs; however, refer to the following for variation on bit 20.

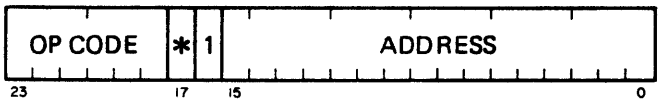
The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

When an interrupt occurs, the status of the virtual memory system is recorded. Bit 20 is set to ONE if the system is in the User Mode at the time of interrupt; bit 20 is set to ZERO if the Monitor Mode is active.

BRL Branch and Reset Interrupt Long

Formula 25.*+2:A **Affected** C,P



Operation

The highest-level active interrupt is reset (i.e., returned to the inactive state) and the contents of the P Register (current program address) are replaced by the 16-bit effective memory address.

Notes

BRL is normally used to exit an interrupt subroutine. If BRL contains an indirect reference, the last word in the indirect address chain contains the previous status of the virtual memory system in bit M20, the previous machine status (i.e., C Register contents at the time of the interrupt) in bit positions M19-M16, and the return address in bit positions M15-M0 as a result of the BSL instruction. The C Register is restored and the program branches to the return address (restarting the machine to the pre-interrupt status).

Example:

```
                .
                .
        TMA
L       AMA
        SMA     Interrupt occurs (EXM K).
                .
                .
K       BSL  M  Dedicated interrupt location.
M       ***     M becomes L+1 as a result of
                .  BSL at K. The C Register con-
                .  tents are stored in M19-M16.
                .
                .
        BRL* M  Restore C Register and return
                   to L+1.
```

The BRL will not reset the interrupt if external interrupts have been held by an HXI instruction. Control will be returned to the effective memory address.

Those executive traps, which are not affected by the HXI instruction, will be reset by the BRL.

The immediate memory reference cannot be indexed; however, indexing indirect references is permitted, e.g.,

```
        BRL*   X
                .
                .
X       DAC    Y,K
```

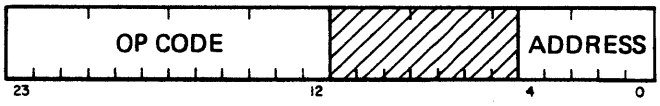
If the BRL instruction is not indirected, the Condition Register is not affected.

External interrupts are prohibited for the period of one instruction following this instruction.

In virtual memory systems, if an indirect BRL is executed in Monitor Mode, bit 20 of the effective memory address determines mode of operation to which machine returns. If bit 20 is set, User Mode is established; if reset, the Monitor Mode is established.

BLU Branch and Link Unrestricted

Formula 0067:a **Affected** J,P



Operation

The next sequential address (program address +1) replaces the contents of the J Register and the contents of the P Register (current program address) are replaced by the 5-bit immediate memory address.

Notes

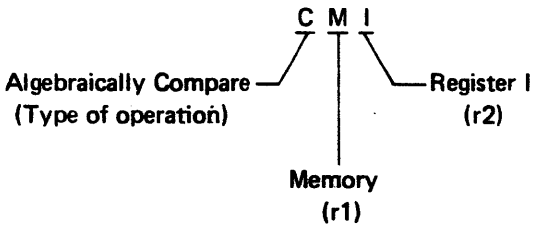
If Program Restrict is enabled, execution of the BLU instruction will turn OFF the Program Restricted Flag (PRF). If the computer is in a HALT condition and the PRF is ON, the BLU instruction will be treated as a NOP instruction.

If virtual memory is enabled, execution of the BLU instruction will automatically establish the Monitor Mode. The 5-bit immediate memory address will not be mapped. Bit 20 of the J Register will be set (ONE) if the system was in the User Mode, and reset (ZERO) if the Monitor Mode was active when the BLU was executed.

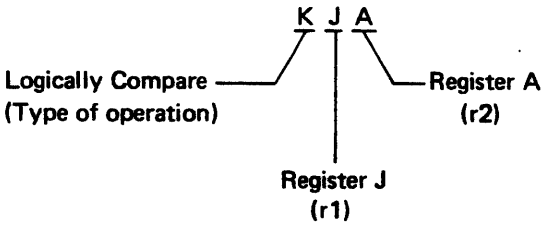
COMPARE INSTRUCTIONS

The compare group of instructions is composed of two basic types of operations; algebraic and logical comparisons. Both types of instructions compare two referenced quantities and set the Condition Register according to the result. Algebraic comparisons treat the references as signed (+ or -) quantities, while logical comparisons assume the references are unsigned quantities.

Algebraic comparisons are identified by the letter "C" as the first letter in the instruction mnemonic (e.g., CAI). Logical comparisons use a mnemonic code beginning with the letter "K" (KAI). The second letter of the mnemonic code designates the first of the compared quantities (r1) and the last letter designates the second quantity. For example:



or



Both algebraic and logical comparisons are performed according to the formula:

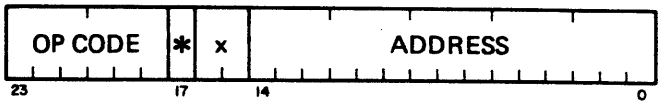
r2 - r1 = C (positive, zero or negative)

Therefore, r2 > r1, r2 < r1 and r2 = r1 will set the Condition Register (C) to positive (+), negative (-) and zero (0), respectively.

The following instructions are included in the compare group.

CMx Compare Memory and Register

Formula 31.*+x:a **Affected** C



Operation

The contents of the effective memory address and the contents of the I, J, or K Register are algebraically compared.

Notes

CMx is not a computer instruction mnemonic but represents a family of instruction mnemonics. x is coded as follows to select one of the index registers.

x = 1 (I)
2 (J)
3 (K)

A code of 31.*+1:a, for example, implements the Compare Memory and I (CMI) instruction.

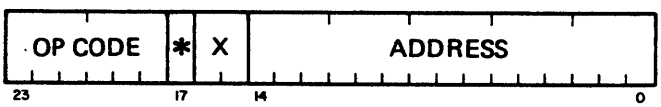
The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
      CMI*    X
       .
       .
X     DAC     Y,K
```

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

CMA Compare Memory and A

Formula 33*+X:a **Affected** C



Operation

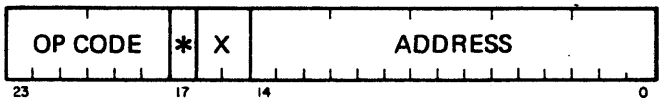
The contents of the effective memory address and the contents of the A Register are algebraically compared.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

CME Compare Memory and E

Formula 32.*+X:a **Affected** - C



Operation

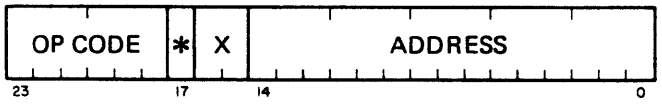
The contents of the effective memory address and the contents of the E Register are algebraically compared.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

CMB Compare Memory and Byte

Formula 34.*+X:a **Affected** C



Operation

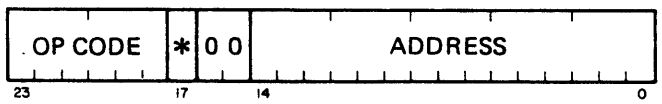
The contents of the B Register (A7-A0) and the contents of the effective memory address (M7-M0) are algebraically compared.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

CZM Compare Zero and Memory

Formula 41.*+0:a **Affected** C



Operation

The contents of the effective memory address and zero are algebraically compared.

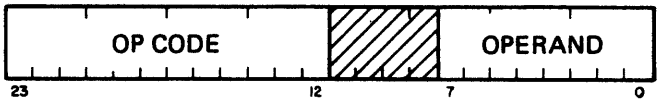
Notes

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

COB Compare Operand and Byte

Formula 0014:o **Affected** C



Operation

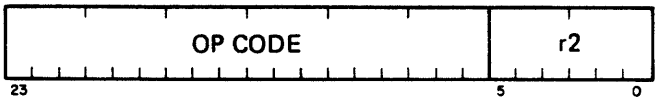
The 8-bit signed operand and the contents of the B Register (A7-A0) are algebraically compared.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

CZr Compare Zero and Register

Formula 002400.r2 **Affected** C



Operation

The contents of the specified register and zero are algebraically compared.

Notes

CZr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r2 is coded as follows to select any of the general purpose registers.

r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

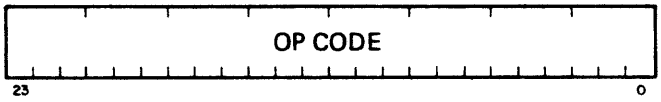
A code of 002400.01, for example, implements the Compare Zero and I (CZI) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

r2 is selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r2, they are logically ORed prior to the specified operation.

CZD Compare Zero and Double

Formula 00240030 **Affected** C



Operation

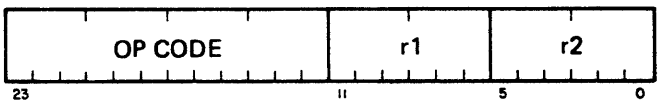
The contents of the E Register are logically ORed with the contents of the A Register, and the result in the D Register and zero are algebraically compared.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

Crr Compare Register and Register

Formula 0024.r1.r2 **Affected** C



Operation

The contents of r1 and the contents of r2 are algebraically compared.

Notes

Crr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select any of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

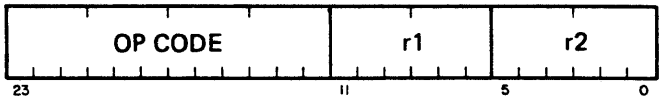
A code of 0024.01.02, for example, implements the Compare I and J (CIJ) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1, or r2, they are logically ORed prior to the specified operation.

Krr Kompare Register and Register

Formula 0025.r1.r2 **Affected** C



Operation

The contents of r1 and r2 are logically compared.

Notes

Krr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select any of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

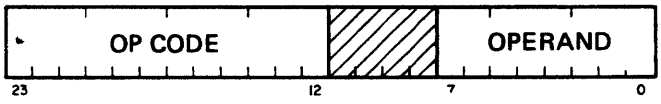
A code of 0025.01.02, for example, implements the Kompare I to J (KIJ) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1 and r2, they are logically ORed prior to the specified operation.

KOB Kompare Operand and Byte

Formula 0015:o **Affected** C



Operation

The 8-bit operand and the contents of the B Register (A7-A0) are logically compared.

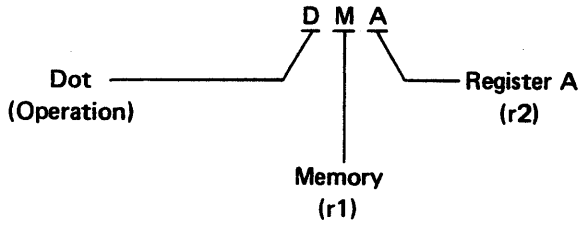
Note

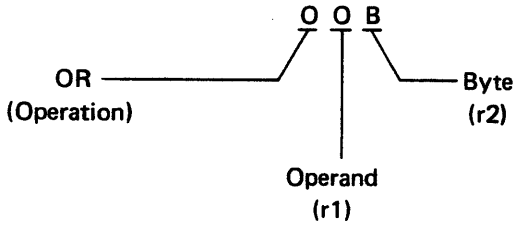
The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

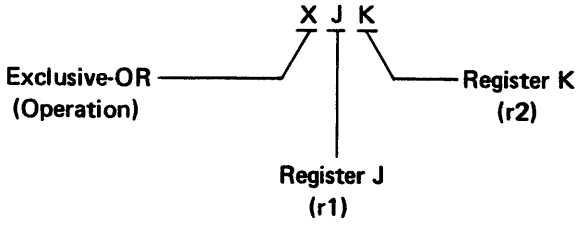
LOGICAL INSTRUCTIONS

The logical group of instructions includes AND (Dot product), OR and exclusive-OR operations. All three types use two quantities to produce a logical result. The AND instructions use a mnemonic code beginning with the letter "D" for "Dot". The OR instructions use a mnemonic beginning with the letter "O", while exclusive-OR instructions are distinguished by the letter "X".

The second letter of the mnemonic code identifies the first of the two quantities (r1). The third letter signifies the second quantity (r2). Some examples are listed below.







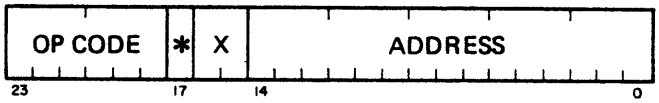
Unless specifically noted otherwise in the individual descriptions, the result of the logical operation replaces the previous contents of r2 while r1 is unchanged. The Condition Register is set to the status of the result (Positive, Negative, or Zero) after the operation. The various logical operations are illustrated in the following table.

r1	r2	r1 AND r2	r1 OR r2	r1 XOR r2
1	1	1	1	0
0	1	0	1	1
1	0	0	1	1
0	0	0	0	0

The following instructions are included in the logical group.

DMA Dot Memory with A

Formula 36.*+X:a **Affected** A,C



Operation

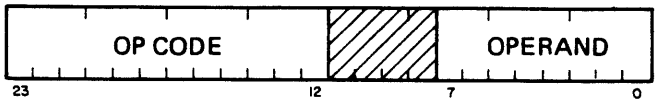
A logical AND is performed between the contents of the effective memory address and the contents of the A Register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

DOB Dot Operand with Byte

Formula 0016:o **Affected** A,C



Operation

A logical AND is performed between the 8-bit operand and the contents of the B Register (A7-A0). Bits A23-A8 are unchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

Drr Dot Register with Register

Formula 0026.r1.r2 **Affected** r2,C

Operation

A logical AND is performed between the contents of r1 and r2.

Notes

Drr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select one of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

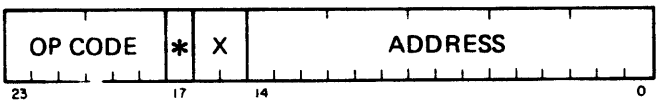
A code of 0026.01.02, for example, implements the Dot I with J (DIJ) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1 or r2, they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 registers.

OMA OR Memory with A

Formula 35.*+X:a **Affected** A,C



Operation

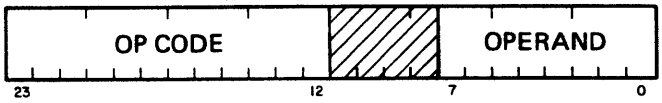
A logical OR is performed between the contents of the effective memory address and the contents of the A Register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

OOB OR Operand with Byte

Formula 0004:o **Affected** A,C



Operation

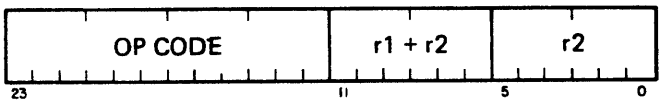
A logical OR is performed between the 8-bit operand and the contents of the B Register (A7-A0). Bits A23-A8 are unchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

Orr OR Register with Register

Formula 0030.r1+r2.r2 **Affected** r2,C



Operation

A logical OR is performed between the contents of r1 and r2.

Notes

Orr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select any of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

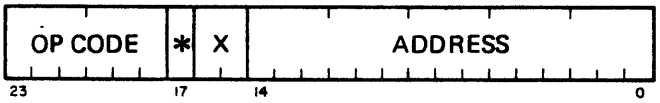
A code of 0030.03.02, for example, implements the OR I with J (OIJ) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1, they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 registers. Affected registers are the Condition Register and those selected in group r2.

XMA Exclusive-OR Memory with A

Formula 37.*+X:a **Affected** A,C



Operation

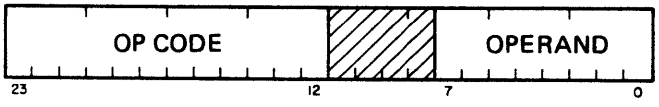
An exclusive-OR operation is performed between the contents of the effective memory address and the contents of the A Register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

XOB Exclusive-OR Operand with Byte

Formula 0017:o **Affected** A,C



Operation

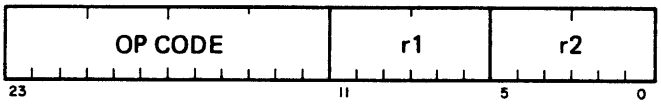
An exclusive-OR operation is performed between the 8-bit operand and the contents of the B Register (A7-A0). Bits A23-A8 are unchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

Xrr Exclusive-OR Register with Register

Formula 0027.r1.r2 **Affected** r2,C



Operation

An exclusive-OR function is performed between the contents of r1 and r2.

Notes

Xrr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select any of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

A code of 0027.01.02, for example, implements the Exclusive-OR I with J (XIJ) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1 or r2, they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 registers. Affected registers are the Condition Register and those selected in group r2.

SHIFT INSTRUCTIONS

The shift instruction group consists of arithmetic and logical shifts. The arithmetic shifts cause the contents of a register to be shifted left or right a specified number of times, while preserving the original sign. The logical shifts are similar to the arithmetic shifts, except that the sign bit is shifted along with the other bits.

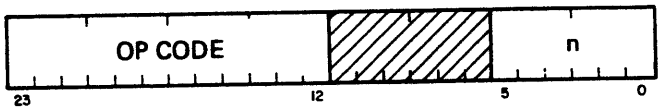
With both types of shift instructions, any number of shifts from 1 to 63 may be programmed without restriction. The number of shifts (n) are specified in bits 5-0 of the instruction word.

At the conclusion of any shift operation, the Condition Register is set to the status of the affected register's contents (Positive, Negative, Zero).

The following instructions are included in the shift group.

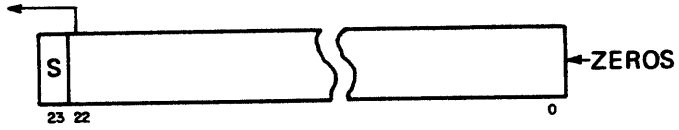
LAA Left Shift Arithmetic A

Formula 0040:n **Affected** A,C



Operation

Bits A22-A0 are shifted left n places, with the most significant n bits being lost and n ZEROs being shifted into the least significant bit positions. The sign bit (A23) is unchanged.



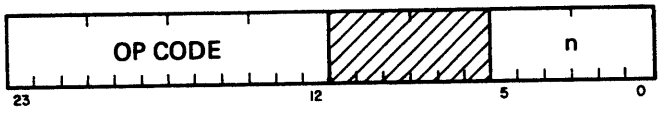
Notes

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

If a bit shifted off from A22 differs from the sign bit, the Condition Register will be set to Overflow. (This is in addition to the Positive/Negative/Zero status.)

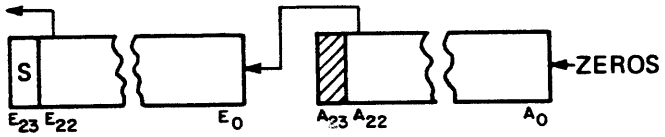
LAD Left Shift Arithmetic Double

Formula 0046:n **Affected** E,A,C



Operation

Bits E22-E0 and A22-A0 are shifted, as one register, left n places. The most significant n bits are lost and the least significant n bits are replaced with ZEROs. Bits E23 and A23 are bypassed. E23 is the D Register sign bit and A23 is not used in the double-precision format.



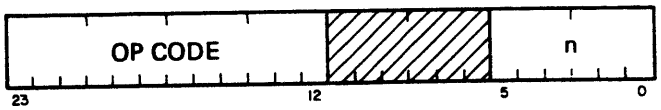
Notes

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

If a bit shifted off from E22 differs from the sign bit, the Condition Register will be set to Overflow. (This is in addition to the Positive/Negative/Zero status.)

LLA Left Shift Logical A

Formula 0042:n **Affected** A,C



Operation

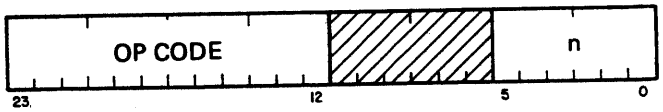
Bits A23-A0 are shifted left n places, with the most significant n bits being lost and the least significant n bits replaced by ZEROs.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

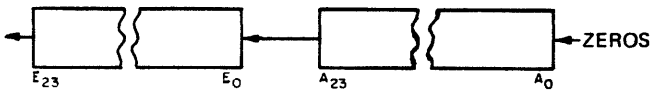
LLD Left Shift Logical Double

Formula 0050:n **Affected** E,A,C



Operation

Bits E23-E0 and A23-A0 are shifted, as one register, left n places. The most significant n bits are lost and the least significant n bits are replaced with ZEROs.

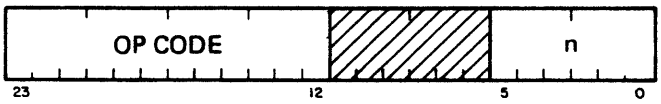


Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

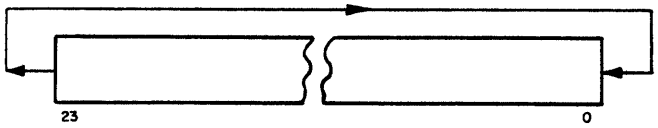
LRA Left Rotate A

Formula 0044:n **Affected** A,C



Operation

Bits A23-A0 are rotated left n places. No bits are lost.

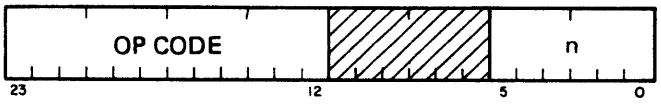


Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

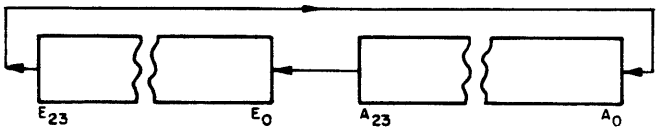
LRD Left Rotate Double

Formula 0052:n **Affected** E,A,C



Operation

Bits E23-E0 and A23-A0 are rotated, as one register, left n places, with E23 replacing A0 and A23 replacing E0 as each shift takes place. No bits are lost.

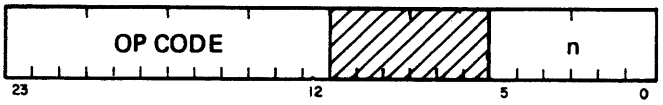


Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

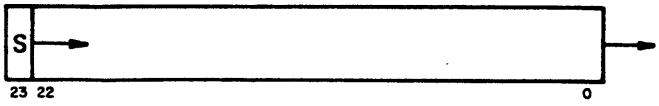
RAA Right Shift Arithmetic A

Formula 0041:n **Affected** A,C



Operation

Bits A22-A0 are shifted right n places. The least significant n bits are lost and the most significant n bits are replaced by an extension of the sign bit (A23). The sign bit is not changed.

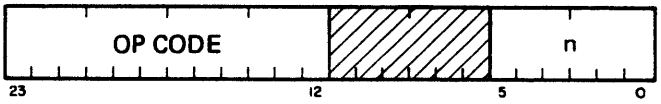


Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

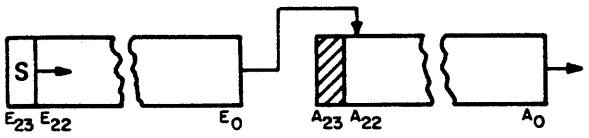
RAD Right Shift Arithmetic Double

Formula 0047:n **Affected** E,A,C



Operation

Bits E22-E0 and A22-A0 are shifted, as one register, right n places. The least significant n bits are lost and the most significant n bits are replaced by an extension of the sign bit (E23). Bit A23 is bypassed.

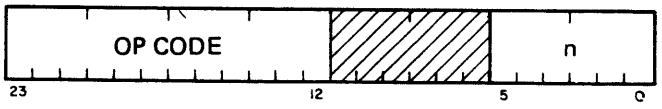


Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

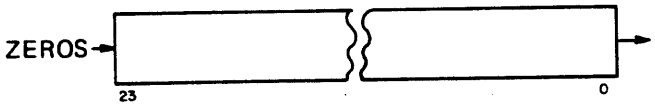
RLA Right Shift Logical A

Formula 0043:n **Affected** A,C



Operation

Bits A23-A0 are shifted right n places. The least significant n bits are lost and the most significant n bits are replaced by ZEROs.

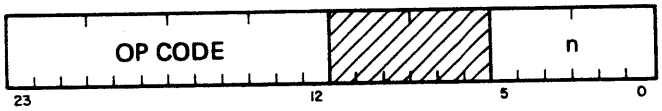


Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

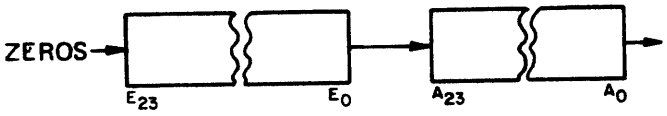
RLD Right Shift Logical Double

Formula 0051:n **Affected** E,A,C



Operation

Bits E23-E0 and A23-A0 are shifted, as one register, right n places. The least significant n bits are lost and the most significant n bits are replaced by ZEROs.

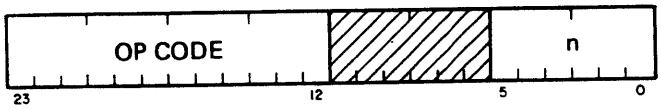


Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

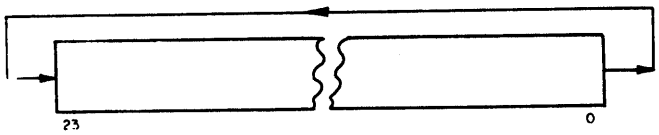
RRA Right Rotate A

Formula 0045:n **Affected** A,C



Operation

Bits A23-A0 are rotated right n places. No bits are lost.

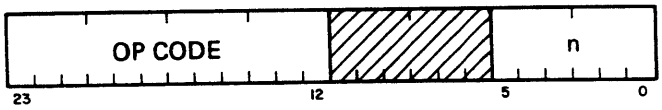


Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

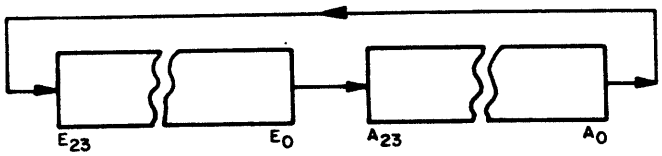
RRD Right Rotate Double

Formula 0053:n **Affected** E,A,C



Operation

Bits E23-E0 and A23-A0 are rotated, as one register, right n places, with E0 replacing A23 and A0 replacing E23 as each shift takes place. No bits are lost.



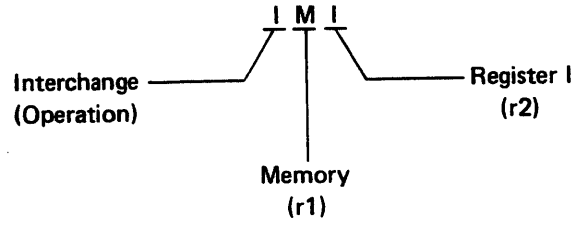
Note

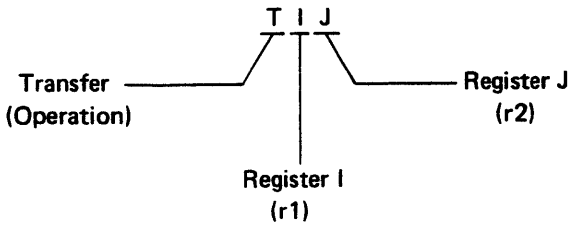
The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

TRANSFER INSTRUCTIONS

The transfer instruction group includes various types of operations. Among these are: interchanges between memory and a specified register, interchanges between registers, memory-to-register and register-to-memory transfers, and register-to-register transfers.

The mnemonic code for the transfer instruction describes the individual operation. The first letter of the mnemonic indicates what action is to be taken; "I" for interchange or "T" for transfer. The second and third letters specify the source (r1) and destination (r2), respectively. Some examples are listed below:





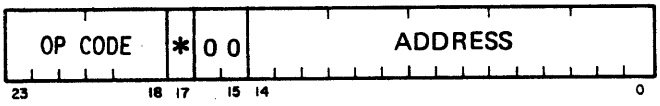
With the exception of the interchange instructions, the transfer group (r1) is not altered by the execution of any instructions in the transfer group.

The Condition Register is always set to reflect the status (Positive, Negative, or Zero) of the contents of r2, at the completion of the instruction.

The following instructions are included in the transfer group.

EMB Extract Memory Byte

Formula 31.*+0:a **Affected** B,C



Operation

The effective memory address is added to the contents of the J Register, producing the word address which contains the byte to be extracted. The selected byte, as determined by the contents of bits 23 and 22 of the index J Register, is then placed in the B Register.

Notes

The following table shows the correspondence between bits 23 and 22 of J and the byte to be extracted:

Bits 23 and 22 J Register	Byte Selection
01	Leftmost Byte (bits 23-16 of EMA+J)
10	Middle byte (bits 15-8 of EMA+J)
11	Rightmost byte (bits 7-0 of EMA+J)
00	Rightmost byte (bits 7-0 of EMA+J)

The final address of any indirect/index sequence should not be indexed since implied indexing on the J Register takes place. If indexing is specified on the final address, then the specified index register will be algebraically added to the EMA prior to the final addition of J with the EMA.

Examples:

If J = '40000030
and K = '00000010 when the following is executed:

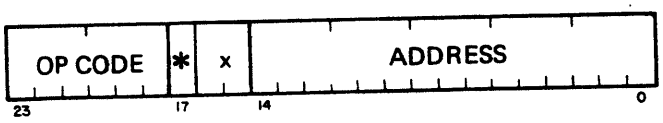
```
                EMB*    '40
'40             DAC*    '50,K
'42             DATA    "XYZ"
'60             DAC     '12
```

Then the character Y will be placed in the B Register. Note that the effective address of the indirect/index sequence is '12. However, '12 plus bits 15-0 of index J Register ('30) yields the final address of '42. Since a byte specification of 10_2 was made in bits 23-22 of index J Register, then the second byte (bits 15-8) of memory location '42 is placed in the B Register.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

IMx Interchange Memory and Register

Formula 66.*+x:a **Affected** M,x,C



Operation

The contents of the effective memory address and the I, J, or K Register are interchanged.

Notes

IMx is not a computer instruction mnemonic but represents a family of instruction mnemonics. x is coded as follows to select one of the index registers.

x = 1 (I)
2 (J)
3 (K)

A code of 66*+1:a, for example, implements the Interchange Memory and I (IMI) instruction.

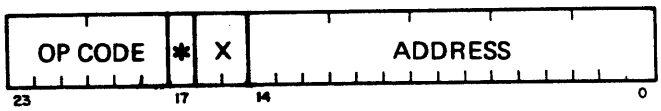
The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.

```
      IMK*    X
       .
       .
X     DAC     Y,J
```

The Condition Register is set to Positive, Negative, or Zero, based on the result in I, J, or K at the completion of the operation.

IMA Interchange Memory and A

Formula 70.*+X:a **Affected** M,A,C



Operation

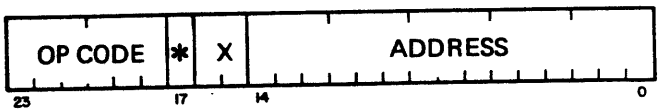
The contents of the effective memory address and the A Register are interchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in A at the completion of the operation.

IME Interchange Memory and E

Formula 67.*+X:a **Affected** M,E,C



Operation

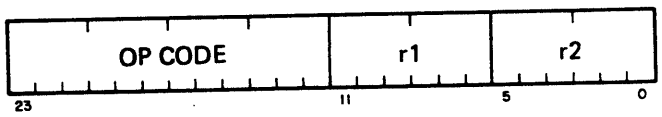
The contents of the effective memory address and the E Register are interchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in E at the completion of the operation.

Irr Interchange Register and Register

Formula 0035.r1.r2 **Affected** r1,r2,C



Operation

The contents of r1 and r2 are interchanged.

Notes

Irr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select any of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

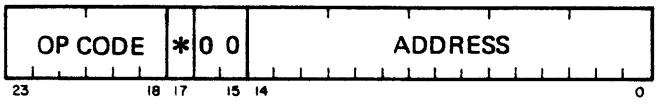
A code of 0035.01.02, for example, implements the Interchange I and J (IIJ) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result in r2 at the completion of the operation.

r1 and r2 are selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1 or r2 they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 and r1 registers. Affected registers are the Condition Register and those selected in group r1 or r2.

RBM Replace Byte in Memory

Formula 27.*+0:a **Affected** M



Operation

The effective memory address is added to the contents of the I Register producing the word address which contains the byte to be replaced. The selected byte, as determined by the contents of bits 22 and 23 of the index I Register, is then replaced by the contents of the B Register.

Notes

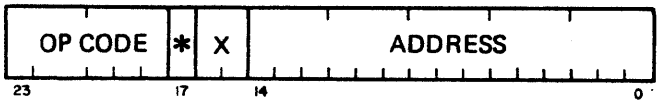
The following table shows the correspondence between bits 22 and 23 of I and the byte to be replaced.

Bits 23 and 22 I Register	Byte Selection
01	Leftmost byte (bits 23-16 of EMA+I)
10	Middle byte (bits 15-8 of EMA+I)
11	Rightmost byte (bits 7-0 of EMA+I)
00	Causes no operation

The final address of any indirect/index sequence should not be indexed since implied indexing of the I Register takes place. If indexing is specified on the final address, then the specified index register will be logically ORed with the I Register prior to the add function with the EMA.

TMB Transfer Memory to Byte

Formula 07.*+X:a **Affected** A,C



Operation

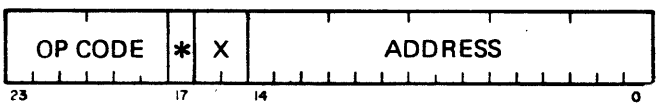
The 8 least significant bits (7-0) of the contents of the effective memory address replace the previous contents of the B Register (A7-A0). Bits A23-A8 are unaffected.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the B Regiter at the completion of the operation.

TMD Transfer Memory to Double

Formula 06.*+X:a **Affected** E,A,C



Operation

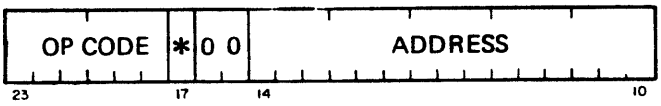
The contents of the effective memory address (EMA) and the next sequential address (EMA+1) replace the previous contents of the D Register (E and A). EMA and EMA+1 are transferred to E and A, respectively.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in D at the completion of the operation.

TMQ Transfer Memory to Query Register

Formula 51.*+0:a **Affected** Query



Operation

Bits 23, 22, 21 and 17-0 of the contents of the effective memory address replace the previous contents of the Query Register. These bits are loaded into the Query Register in bit positions 23, 22, 21, and 17-0, respectively.

Notes

Executing this instruction will cause the Program Halt and Address Trap to be enabled or disabled, depending on the state of bits 23, 22, and 21 of the effective memory address.

Bit 23 = ONE - Disable Address Trap
Bit 23 = ZERO = Enable Address Trap

Bit 22 = ONE = Trap on Write only
Bit 22 = ZERO = Trap each time selected address is referenced

Bit 21 = ONE = Trap or Halt during User mode only
Bit 21 = ZERO = Trap or Halt during Monitor mode only

Example:

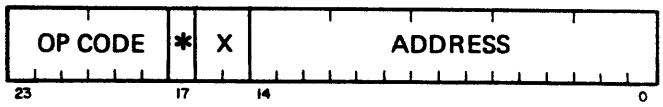
```
        TMQ   OA
        ...
OA      DAC   ADDR    Enable Address Trap
        or
OA      DAC*  0       Disable Address Trap
```

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
        TMQ*  X
        .
        ..
X       DAC   Y,I
```

TMA Transfer Memory to A

Formula 05.*+X:a **Affected** A,C



Operation

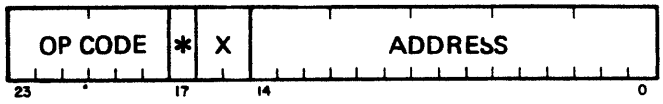
The contents of the effective memory address replace the previous contents of the specified register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in A at the completion of the operation.

TME Transfer Memory to E

Formula 04.*+X:a **Affected** E,C



Operation

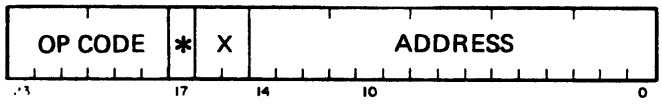
The contents of the effective memory address replace the previous contents of the specified register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in E at the completion of the operation.

TMI Transfer Memory to I

Formula 01.*+X:a **Affected** I,C



Operation

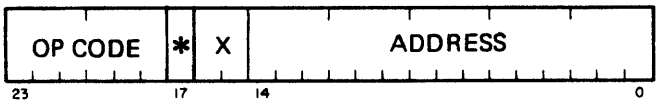
The contents of the effective memory address replace the previous contents of the specified register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in I at the completion of the operation.

TMJ Transfer Memory to J

Formula 02.*+X:a **Affected** J,C



Operation

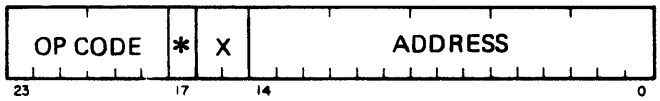
The contents of the effective memory address replace the previous contents of the specified register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in J at the completion of the operation.

TMK Transfer Memory to K

Formula 03.*+X:a **Affected** K,C



Operation

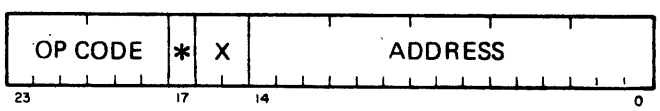
The contents of the effective memory address replace the previous contents of the specified register.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in K at the completion of the operation.

TMR Transfer Memory to Registers

Formula 10.*+X:a **Affected** I,J,K,E,A



Operation

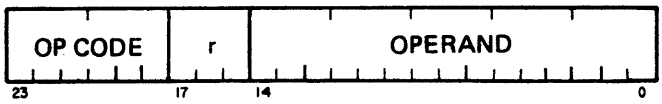
The I, J, K, E and A Registers are loaded from consecutive memory addresses beginning with the effective memory address.

Note

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

TNr Transfer Negative Operand to Register

Formula 63.r:o **Affected** r,C



Operation

The two's complement of the 15-bit unsigned operand replaces the previous contents of bits 23-0 of the specified register.

Notes

TNr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r is coded as follows to select one of the general purpose registers.

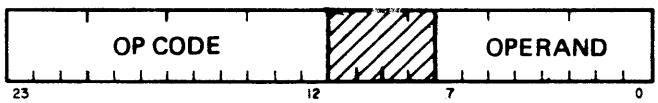
r = 1 (I)
2 (J)
3 (K)
4 (E)
5 (A)
6 (T)

A code of 63.1:o, for example, implements the Transfer Negative Operand to I (TNI) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result in the specified register at the completion of the operation.

TOB Transfer Operand to Byte

Formula 0003:o **Affected** A,C



Operation

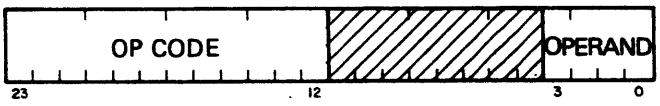
The 8-bit signed operand replaces the previous contents of the B Register (A7-A0). Bits A23-A8 are unaffected.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

TOC Transfer Operand to Condition Register

Formula 0036:o **Affected** C



Operation

The 4-bit operand replaces the previous contents of the Condition Register.

Note

Operand definition is as follows:

Bit 0 = ONE = Overflow
= ZERO = No Overflow

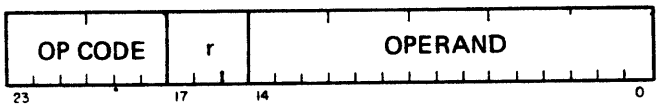
Bit 1 = ONE = Negative
= ZERO = Not Negative

Bit 2 = ONE = Zero
= ZERO = Not Zero

Bit 3 = ONE = Positive
= ZERO = Not Positive

TOr Transfer Operand to Register

Formula 62.r:o **Affected** r,C



Operation

The 15-bit unsigned operand replaces the previous contents of bits 23-0 of the specified register.

Notes

TOr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r is coded as follows to select one of the general purpose registers.

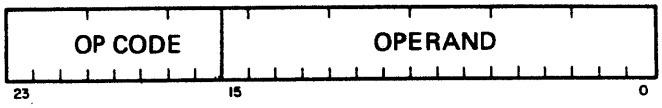
r = 1 (I)
2 (J)
3 (K)
4 (E)
5 (A)
6 (T)

A code of 62.1:o, for example, implements the Transfer Operand to I (TOI) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result in the specified register at the completion of the operation.

TLO Transfer Long Operand to K

Formula 236:A **Affected** K

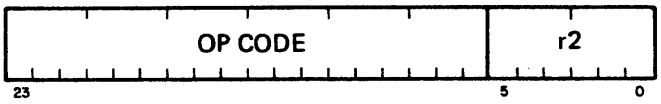


Operation

The 16-bit operand (or address) replaces the previous contents of bits 15-0 of the K Register. Zeros are copied into bit positions 23-16.

TSr Transfer Switches to Register

Formula 003100.r2 **Affected** r2,C



Operation

The states (set = ONE) of the console control switches (i.e., switch register) are transferred to the corresponding bit positions of the specified register.

Notes

TSr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r2 is coded as follows to select any of the general purpose registers.

r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

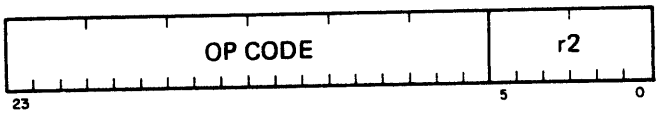
A code of 003100.01, for example, implements the Transfer Switches to I (TSI) instruction.

The Condition Register is set to Positive, Negative, or Zero, based on the result in the specified register at the completion of the operation.

r2 is selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r2, the switches are copied into all of the selected r2 registers. Affected registers are the Condition Register and those selected in group r2.

TZr Transfer Zero to Register

Formula 003000.r2 **Affected** r2,C



Operation

The previous contents of the specified register are replaced with ZEROs.

Notes

TZr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r2 is coded as follows to select any of the general purpose registers or the D register.

r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)
30 (D)

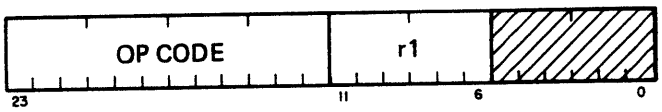
A code of 003000.01, for example, implements the Transfer Zero to I (TZI) instruction.

The Condition Register is set to Postive, Negative, or Zero, based on the result in the specified register at the completion of the operation.

r2 is selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r2, they are logically ORed prior to the specified operation. The result is copied into all of the selected r2 registers. Affected registers are the Condition Register and those selected in group r2.

TrB Transfer Register to Byte

Formula 0002.r1 **Affected** A



Operation

The least significant 8 bits (7-0) of the contents of the specified register replace the previous contents of the B Register (A7-A0). Bits A23-A8 are unchanged.

Notes

TrB is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 is coded as follows to select one-of-five general purpose registers.

r1 = 01 (I)
02 (J)
04 (K)
10 (E)
40 (T)

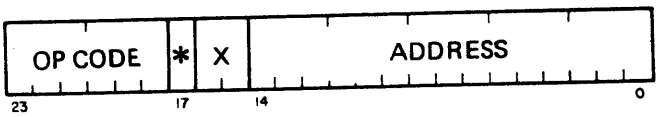
A code of 0002.01, for example, implements the Transfer I to Byte (TIB) instruction.

The Condition Register is not affected.

r1 is selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1, they are logically ORed prior to the specified operation.

TBM Transfer Byte to Memory

Formula 17.*+X:a **Affected** M

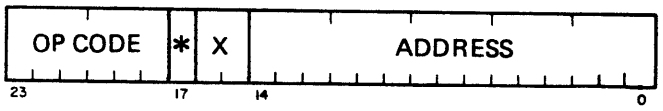


Operation

The contents of the B Register (A7-A0) replace the 8 least significant bits of the contents of the effective memory address. Bits 23-8 of the memory word are unaffected.

TDM Transfer Double to Memory

Formula 16.*+X:a **Affected** M

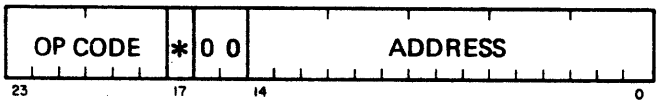


Operation

The contents of the D Register (E and A) replace the previous contents of the effective memory address (EMA) and the next sequential address (EMA+1). The contents of E and A are transferred to EMA and EMA+1, respectively.

TFM Transfer Flag to Memory

Formula 46.*+0:a **Affected** M,C



Operation

The previous contents of the effective memory address are replaced by ONEs.

Notes

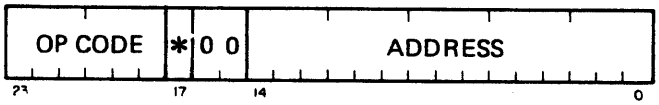
The Condition Register is set to the status of memory (Positive, Negative, or Zero) prior to the transfer.

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
      TFM*    X
       .
       .
X     DAC     Y,I
```

TZM Transfer Zero to Memory

Formula 66.*+0:a **Affected** M,C



Operation

The previous contents of the effective memory address are replaced by ZEROs

Notes

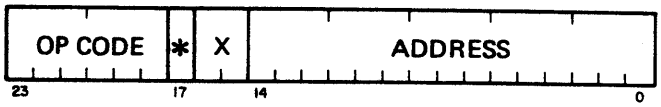
The Condition Register is set to the status of memory (Positive, Negative, or Zero) prior to the transfer.

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
      TZM*    X
       .
       .
X     DAC     Y,I
```

TAM Transfer A to Memory

Formula 15.*+X:a **Affected** M

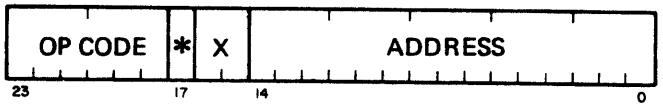


Operation

The contents of the A Register replace the previous contents of the effective memory address.

TEM Transfer E to Memory

Formula 14.*+X:a **Affected** M

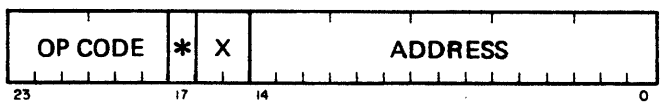


Operation

The contents of the E Register replace the previous contents of the effective memory address.

TIM Transfer I to Memory

Formula 11.*+X:a **Affected** M

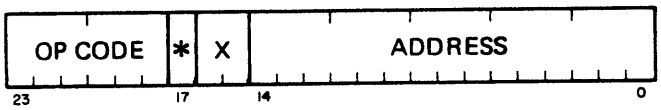


Operation

The contents of the I Register replace the previous contents of the effective memory address.

TJM Transfer J to Memory

Formula 12.*+X:a **Affected** M

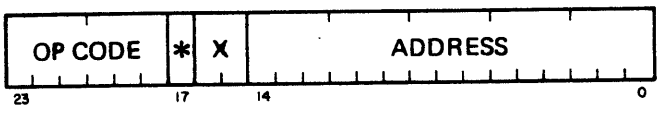


Operation

The contents of the J Register replace the previous contents of the effective memory address.

TKM Transfer K to Memory

Formula 13.*+X:a **Affected** M

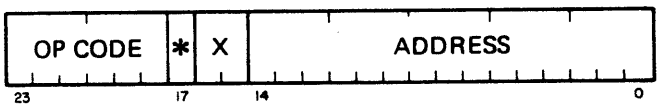


Operation

The contents of the K Register replace the previous contents of the effective memory address.

TRM Transfer Registers to Memory

Formula 20.*+X:a **Affected** M



Operation

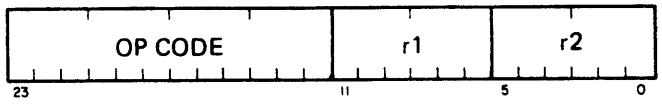
The contents of the I, J, K, E and A Registers are stored in consecutive memory locations beginning with the effective memory address.

Note

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

Trr Transfer Register to Register

Formula 0030.r1.r2 **Affected** r2,C



Operation

The contents of r1 replace the previous contents of r2.

Notes

Trr is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 and r2 are coded as follows to select any of the general purpose registers.

r1 or r2 = 01 (I)
02 (J)
04 (K)
10 (E)
20 (A)
40 (T)

A code of 0030.01.02, for example, implements the Transfer I to J (TIJ) instruction.

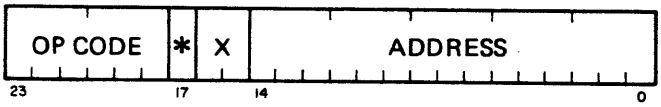
The Condition Register is set to Positive, Negative, or Zero, based on the result in r2 at the completion of the operation.

BYTE PROCESSING INSTRUCTIONS

The byte processing group of instructions permits program manipulation of all three bytes within the computer word (24 bits); e.g., extract, replace, etc. The following instructions are inclusive of byte processing operations.

AMB Add Memory to Byte

Formula 45.*+X:a **Affected** A,C



Operation

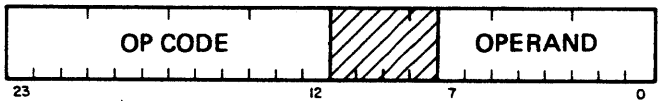
Bits 7-0 of the contents of the effective memory address are algebraically added to the contents of the B Register (A7-A0). Bits 23-8 of the A Register are unchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

AOB Add Operand to Byte

Formula 0012:o **Affected** A,C



Operation

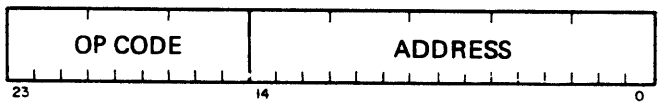
The 8-bit signed operand is algebraically added to the contents of the B Register (A7-A0). Bits 23-8 of the A Register are unchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

BBI Branch when Byte Address +1 in I ≠ 0

Formula 607:a **Affected** I



Operation

The contents of bits 22 and 23 of the I Register are incremented by one. If the result of this addition (in bits 22 and 23) is not 00_2, then the contents of the P Register (current program address) are replaced by the 15-bit effective memory address. If the result of the addition to bits 22 and 23 is 00_2, then bits 22 and 23 are set to 01_2 and bits 21-0 are incremented by one. If the resultant sum in bits 21-0 is zero, then the P Register advances to the next sequential program location and the index register is set to 20000000_8 Otherwise, the contents of the P Register are replaced by the 15-bit effective memory address.

Notes

In general, the BBI and BBJ instructions are used as special index register increments in order to sequentially reference consecutive bytes in memory via the EMB and RBM instructions. Consider the following example which will move 11 consecutive bytes starting from the third byte at location '200 to the first byte at location '300.

```
TMJ     = '60000200
TMI     = '20000300
TNK     11
EMB     0
RBM     0
BBI     *+1
BBJ     *+1
BWK     *-4
```

Occasionally, it is possible to use the address of a portion of the I Register as a byte counter as well as a word pointer. This may be illustrated by the following example which will set the buffer to blanks, starting at byte 3 of location '100 through byte 3 of location '102.

```
TOB     "b"
TMI     = '77777775     bits 22 and 23 = 3,
                        bits 21-0 =-3
RBM     '100+3
BBI     *-1
```

However, it should be noted this technique of using the index register as both a byte counter and word pointer may be used only in certain instances. Specifically, when the following relationship is true.

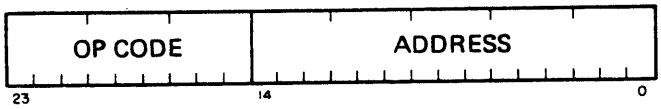
$$R\left(\frac{4\text{-B.n.}}{3}\right) = R\left(\frac{CT}{3}\right)$$

Where:

R ( ) = remainder
b.n. = the starting byte number (1, 2, or 3)
CT = The number of bytes to be referenced.

BBJ Branch when Byte Address +1 in J≠0

Formula 617:a **Affected** J



Operation

The contents of bits 22 and 23 of the J Register are incremented by one. If the result of this addition (in bits 22 and 23) is not 00_2, then the contents of the P Register (current program address) are replaced by the 15-bit effective memory address. If the result of the addition to bits 22 and 23 is 00_2, then bits 22 and 23 are set to 01_2 and bits 21-0 are incremented by one. If the resultant sum in bits 21-0 is zero, then the P Register advances to the next sequential program location and the index register is set to 20000000_8. Otherwise, the contents of the P Register are replaced by the 15-bit effective memory address.

Notes

In general, the BBI and BBJ instructions are used as special index register increments in order to sequentially reference consecutive bytes in memory via the EMB and RBM instructions. Consider the following example which will move 11 consecutive bytes starting from the third byte at location '200 to the first byte at location '300.

```
TMJ     = '60000200
TMI     = '20000300
TNK     11
EMB     0
RBM     0
BBI     *+1
BBJ     *+1
BWK     *-4
```

Occasionally, it is possible to use the address of a portion of the J Register as a byte counter as well as a word pointer. This may be illustrated by the following example which will set the buffer to blanks, starting at byte 3 of location '100 through byte 3 of location '102.

```
TOB     "ϕ"
TMI     = '77777775     bits 22 and 23 =3,
                        bits 21-0 =-3
RBM     '100+3
BBJ     *-1
```

However, it should be noted this technique of using the index register as both a byte counter and word pointer may be used only in certain instances. Specifically, when the following relationship is true.

$$R\left(\frac{4\text{-B.n.}}{3}\right) = R\left(\frac{CT}{3}\right)$$

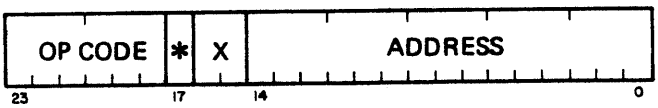
Where:

R ( ) = remainder
b.n. = the starting byte number (1, 2, or 3)

CT = The number of bytes to be referenced.

CMB Compare Memory and Byte

Formula 34.*+X:a **Affected** C



Operation

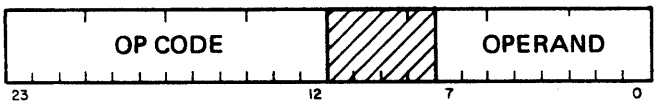
The contents of the B Register (A7-A0) and the contents of the effective memory address (M7-M0) are algebraically compared.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

COB Compare Operand and Byte

Formula 0014:o **Affected** C



Operation

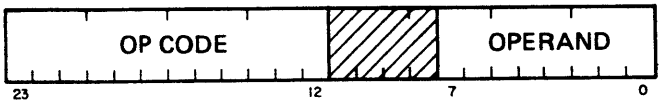
The 8-bit signed operand and the contents of the B Register (A7-A0) are algebraically compared.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

DOB Dot Operand with Byte

Formula 0016:o **Affected** A,C



Operation

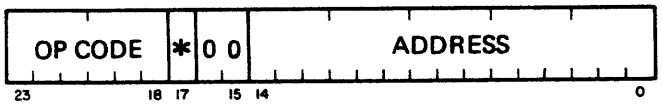
A logical AND is performed between the 8-bit operand and the contents of the B Register (A7-A0). Bits A23-A8 are unchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

EMB Extract Memory Byte

Formula 31.*+0:a **Affected** B,C



Operation

The effective memory address is added to the contents of the J Register, producing the word address which contains the byte to be extracted. The selected byte, as determined by the contents of bits 23 and 22 of the index J Register, is then placed in the B Register.

Notes

The following table shows the correspondence between bits 23 and 22 of J and the byte to be extracted:

Bits 23 and 22 J Register	Byte Selection
01	Leftmost byte (bits 23-16 of EMA+J)
10	Middle byte (bits 15-8 of EMA+J)
11	Rightmost byte (bits 7-0 of EMA+J)
00	Rightmost byte (bits 7-0 of EMA+J)

The final address of any indirect/index sequence should not be indexed since implied indexing on the J Register takes place. If indexing is specified on the final address, then the specified index register will be algebraically added to the EMA prior to the final addition of J with the EMA.

Examples:

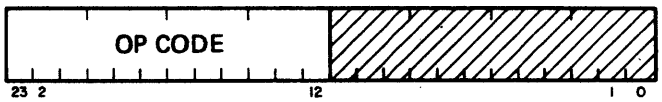
```
          If J =     '40000030
          and K =    '00000010 when the following
                     is executed:
          EMB*       '40
'40       DAC*       '50,K
'42       DATA       "XYZ"
'60       DAC        '12
```

Then the character Y will be placed in the B Register. Note that the effective address of the indirect/index sequence is '12. However, '12 plus bits 15-0 of index J Register ('30) yields the final address of '42. Since a byte specification of 10_2 was made in bits 23-22 of index J Register, then the second byte (bits 15-8) of memory location '42 is placed in the B Register.

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

ESB Extend Sign of Byte

Formula 0010. **Affected** A,C



Operation

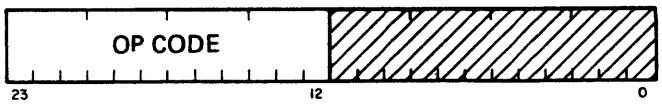
The state of the B Register sign bit (A7) is copied into bit positions A23-A8, forming a sign extension of the byte.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in A at the completion of the operation.

EZB Extend Zeros from Byte

Formula 0007. **Affected** A



Operation

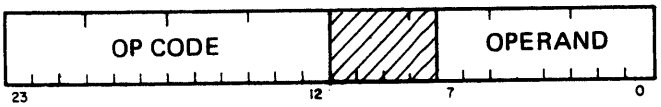
Bit positions A23-A8 are set to ZERO. The contents of the B Register (A7-A0) are not affected.

Note

The Condition Register is not affected.

KOB Kompare Operand and Byte

Formula 0015:o **Affected** C



Operation

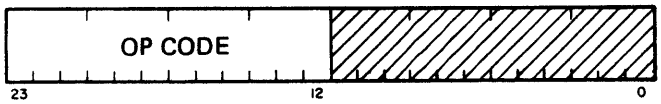
The 8-bit operand and the contents of the B Register (A7-A0) are logically compared.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation.

NBB Negate of Byte to Byte

Formula 0005. **Affected** A,C



Operation

The contents of the B Register (A7-A0) are two's complemented. Bit positions A23-A8 are unchanged.

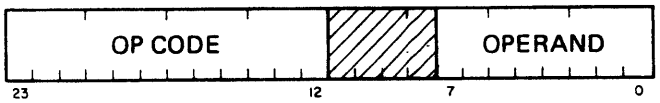
Notes

An Overflow will result when negating 2^7 (full-scale negative byte).

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

OOB OR Operand with Byte

Formula 0004:o **Affected** A,C



Operation

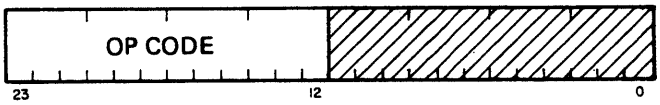
A logical OR is performed between the 8-bit operand and the contents of the B Register (A7-A0). Bits A23-A8 are unchanged.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

PBB Positive of Byte to Byte

Formula 0006. **Affected** A,C



Operation

The absolute value of the contents of the B Register (A7-A0) is placed in the B Register.

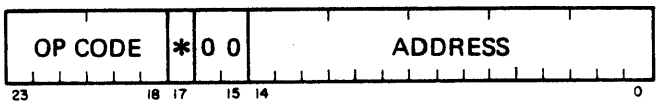
Notes

An Overflow will result when negating a full scale negative byte.

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

RBM Replace Byte in Memory

Formula 27.*+0:a **Affected** M



Operation

The effective memory address is added to the contents of the I Register producing the word address which contains the byte to be replaced. The selected byte, as determined by the contents of bits 22 and 23 of the Index I Register, is then replaced by the contents of the B Register.

Notes

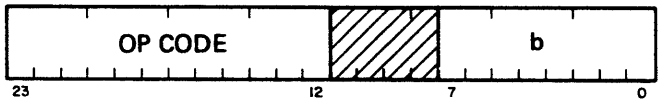
The following table shows the correspondence between bits 22 and 23 of I and the byte to be replaced.

Bits 23 and 22 I Register	Byte Selection
01	Leftmost byte (bits 23-16 of EMA+I)
10	Middle byte (bits 15-8 of EMA+I)
11	Rightmost byte (bits 7-0 of EMA+I)
00	Causes no operation

The final address of any indirect/index sequence should not be indexed since implied indexing on the I Register takes place. If indexing is specified on the final address, then the specified index register will be logically ORed with the I Register prior to the add function with the EMA.

QBB Query Bits of Byte

Formula 0011:b **Affected** C



Operation

A logical AND is performed between operand bits 7-0 and the contents of the B Register. The Condition Register is set according to the status of the result; i.e., Positive, Negative, or Zero.

Note

Examples:

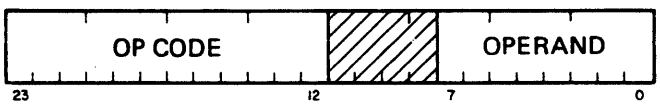
```
(1)     TOA     B7      A = '00000200   C = Positive
        . . .
        . . .
        QBB     B7                      C = Negative

(2)     TOA     B6      A = '00000100   C = Positive
        . . .
        . . .
        QBB     B6                      C = Positive

(3)     TNA     1       A = '77777777   C = Negative
        . . .
        . . .
        DMA     MASK    A = '40000000   C = Negative
        . . .
        . . .
MASK    DATA    '40000000
```

SOB Subtract Operand from Byte

Formula 0013:o **Affected** A,C



Operation

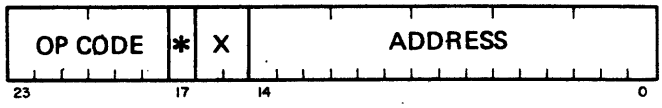
The 8-bit signed operand is algebraically subtracted from the contents of the B Register (A7-A0). Bits A23-A8 are unaffected.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result of the operation. Overflow is set if the arithmetic operation generates a carry into the sign bit without a carry out, or a carry out without a carry in.

TBM Transfer Byte to Memory

Formula 17.*+X:a **Affected** M

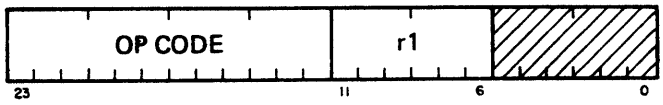


Operation

The contents of the B Register (A7-A0) replace the 8 least significant bits (7-0) of the contents of the effective memory address. Bits 23-8 of the memory word are unaffected.

TrB Transfer Register to Byte

Formula 0002.r1 **Affected** A



Operation

The least significant 8 bits (7-0) of the contents of the specified register replace the previous contents of the B Register (A7-A0). Bits A23-A8 are unchanged.

Notes

TrB is not a computer instruction mnemonic but represents a family of instruction mnemonics. r1 is coded as follows to select one-of-five general purpose registers.

r1 = 01 (I)
02 (J)
04 (K)
10 (E)
40 (T)

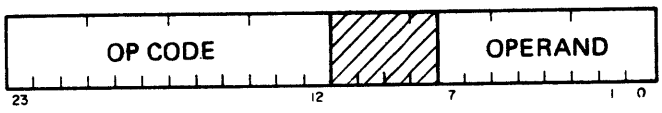
A code of 0002.01, for example, implements the Transfer I to Byte (TIB) instruction.

The Condition register is not affected.

r1 is selected by unitary bits. Therefore, none, all six, or any combination of registers may be selected. If more than one register is selected in group r1, they are logically ORed prior to the specified operation.

TOB Transfer Operand to Byte

Formula 0003:o **Affected** A,C



Operation

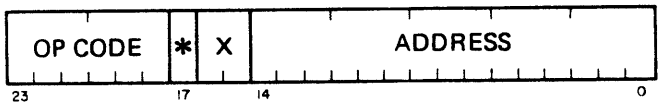
The 8-bit signed operand replaces the previous contents of the B Register (A7-A0). Bits A23-A8 are unaffected.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

TMB Transfer Memory to Byte

Formula 07.*+X:a **Affected** A,C



Operation

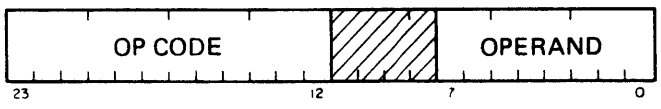
The 8 least significant bits (7-0) of the contents of the effective memory address replace the previous contents of the B Register (A7-A0). Bits A23-A8 are unaffected.

Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the B Register at the completion of the operation.

XOB Exclusive-OR Operand with Byte

Formula 0017:o **Affected** A,C



Operation

An exclusive-OR operation is performed between the 8-bit operand and the contents of the B Register (A7-A0). Bits A23-A8 are unchanged.

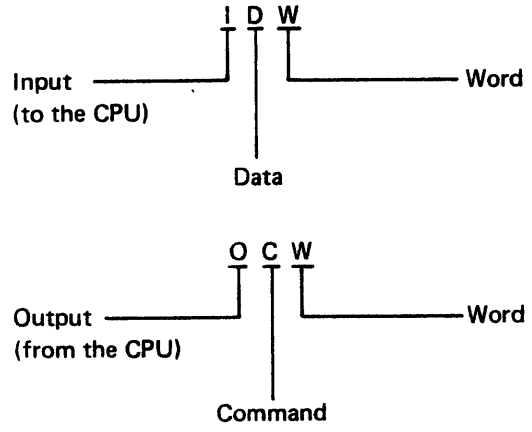
Note

The Condition Register is set to Positive, Negative, or Zero, based on the result in the Byte Register at the completion of the operation.

INPUT/OUTPUT INSTRUCTIONS

The input/output (I/O) instructions provide the required control for all communications between the CPU and the input/output structure. In addition to controlling data transfers between the CPU and peripheral units, the I/O instructions allow peripheral unit command functions and status testing to be placed under program control.

The specific I/O operation can be identified by examination of the individual instruction mnemonic. All I/O instruction mnemonics use the letter "W" to indicate that a full word is to be transferred between the CPU and the I/O structure. The first letter of the mnemonic indicates the direction of the transfer (input or output). The second letter indicates the type of word to be transferred. For example:



There is no "I/O hold", or delay, imposed by the hardware. All I/O instructions are executed unconditionally, i.e., the CPU is not forced to wait for a response from the I/O structure in order to complete the instruction execution cycle.

Although there is no built-in hold/delay provision, a programmed delay can be implemented if desired. At the beginning of each I/O instruction cycle, the Condition Register is cleared. At the end of the execution phase of each I/O instruction, bit 2 (Zero/Not Zero) is set to Zero if the selected channel was ready and accepted the command. If the selected channel was not ready, bit 2 of the

Condition Register remains set to Not Zero. The program can test the Not Zero state of bit 2 with a branch instruction following the I/O instruction. When bit 2 is set to Not Zero, a programmed delay is implemented. For example:

```
ODW    '0103    Output word to Channel 1, Unit 3
BNZ    *-1      Delay if not ready
---             Continue if ready
```

An example of a channel being not ready is when the peripheral unit's data transfer capability is slower than that of the program loop and therefore cannot accept data as it is available from the channel. Another example occurs in a channel/multiunit environment where the channel is connected to peripheral unit A and peripheral unit B is selected for a data transfer.

In this instance, the channel remains not ready until a disconnect/connect sequence is performed and peripheral unit B is connected to the channel. Two cycles are required for the disconnect/connect sequence.

NOTE

Status returned to the Condition Register immediately after completion of an I/O instruction refers to channel status only. A ready (Zero) condition indicates the channel accepted the I/O command. This does not imply the I/O operation was completed with the selected peripheral unit.

If the program selects a non-existent channel or unit, the channel accepts the command or data and leaves bit 2 of the Condition Register set to Not Zero to indicate not ready. The channel will remain not ready for any subsequent commands.

NOTE

Channel number 30_8 cannot be assigned to an I/O channel.

If the system is equipped with the Program Restrict/Instruction Trap option, all I/O instructions will be affected.

The I/O command modes are determined by the configuration of bits 5 and 4 of the OCW instruction and are as follows:

1. Normal – The Normal Channel Operation command is raised by bits 5 and 4 of the OCW being ZEROs (0,0).

2. Multiplex – This command is raised by bits 5 and 4 of the OCW being in a ZERO, ONE (0,1) configuration. (The CPU releases the channel to a master/slave pair of peripheral units.) (An XBC, IBC, or DMACP channel will not respond to a Multiplex command.)

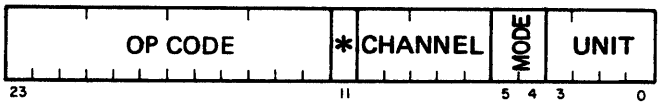
3. Offline – This command is the same as the Multiplex command, except the I/O drivers in the channel are turned off, allowing the second CPU to share peripherals without need of peripheral switches. (Assumes control of I/O bus.) The command is raised by bits 5 and 4 being in a ONE, ZERO (1,0) configuration.

4. Reset – This command operates the same as a Normal command, but resets the channel out of either the Multiplex or Offline mode. (Channel restored on-line, unit selected.) This command is raised by bits 5 and 4 being in a ONE, ONE (1,1) configuration.

The following instructions are included in the input/output group.

OCW Output Command Word

Formula 0070.*+C.U **Affected** C



Operation

An 8-bit or a 24-bit command word is transferred from the A Register to the specified channel/unit combination.

Notes

The Condition Register is cleared, then set to Zero if the I/O channel is ready. If the selected channel is not ready, the Condition Register remains set to Not Zero which allows a programmed delay if desired.

Bits 3-0 of the OCW instruction form a 4-bit paralleled unit code that is used to select a particular peripheral unit. The configuration of bits 4 and 5 determines the Multiplex or Offline mode for a particular channel. The configuration of bits 10-6 determines which channel is to be selected. Bit 11

is the Override Bit, and bits 23-12 define the general process that is to be performed. The only valid unit code for a DMACP channel is 10_8; all others are rejected.

If the Override Bit (*) is set (ONE), the command word assumes immediate control over the channel. The contents of the A Register are transferred to the channel and a disconnect/connect sequence is initiated. The Condition Register is set to Zero to indicate the channel has accepted but not necessarily executed the command. Upon completion of the disconnect/connect sequence, the channel transfers the command word to the unit. In the case of a DMACP channel, the Override bit clears the channel and forces the MPU to a halt; the Condition Register is not set to Zero, and no busy test is required.

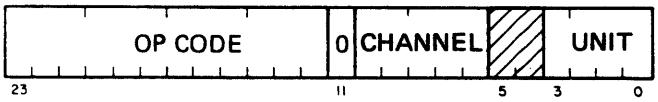
If the Override Bit is not set (ZERO) and the OCW specifies a unit other than the unit connected to the channel and the channel is ready, the command word is accepted by the channel. The Condition Register is set to Not Zero to indicate the channel is not ready. A disconnect/connect sequence is performed and the command is transferred to the unit. The Condition Register is reset to Zero to indicate ready.

Following the execution of an OCW the channel remains not ready until the peripheral unit accepts the data.

If the selected channel is a UBC channel and is actively engaged in a block transfer, executing an OCW with the Override Bit set terminates the transfer sequence leaving the contents of the TAR/PAR and WCR intact. If the Override Bit is not set and the UBC channel is engaged in a block transfer, the OCW instruction will be ignored. The Condition Register will remain set to Not Zero. Once a UBC channel is activated it will not accept an OCW with the Override Bit not set until the word count is complete; i.e., all words in the block have been transferred and WCR equals zero.

ISW Input Status Word

Formula 0073.00+C:U **Affected** A,C



Operation

A status word is transferred from the specified channel/unit combination to the A Register.

Notes

The Condition Register is cleared, then set to Zero if the I/O channel is ready. If the addressed channel/unit combination is not ready (see following notes) or status word is not available, the Condition Register is set to Not Zero to allow a programmed delay.

If the selected channel is in the process of executing a command (resulting from a previous OCW), the channel indicates not ready (Condition Register remains set to Not Zero) and ignores the ISW instruction until the peripheral unit accepts the OCW command. The channel indicates ready (Condition Register set to Zero) and accepts the ISW when it is executed again.

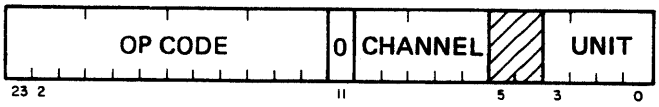
If the ISW specifies a unit other than the unit connected to the channel, the channel indicates not ready and ignores the command. A disconnect/connect is initiated.

If the selected channel is a UBC channel engaged in a block transfer, the Condition Register is set to Zero and a 24-bit status word is transferred to the A Register. Bits 7 through 0 contain the unit status and bit 23 contains the UBC busy status.

If the selected unit is receiving data as the result of an ODW instruction, the ISW is accepted and the Condition Register is set to Zero.

ODW Output Data Word

Formula 0071.00+C:U **Affected** C



Operation

A data word is transferred from the A Register to the specified channel/unit combination.

Notes

The Condition Register is cleared, then set to Zero if the I/O channel is ready. If the channel is busy and cannot accept the data word, the Condition Register is set to Not Zero to allows a programmed delay.

Although, a 24-bit word is transferred to the channel, the peripheral unit accepts only a predetermined number of bits (dictated by peripheral unit design).

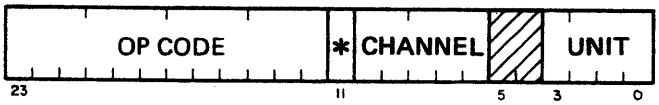
For character-oriented units and units accepting data words of less than 24 bits, the data for transfer must be right-justified in the A Register prior to executing the ODW instruction.

If the ODW instruction specifies a unit other than the unit connected to the channel and the channel is ready, the channel accepts the ODW, sets the Condition Register to Zero, and initiates a disconnect/connect sequence. After completion of the disconnect/connect sequence, the ODW is transferred to the unit. The channel indicates ready to subsequent I/O instructions.

If the ODW instruction specifies a UBC channel that is engaged in a block transfer, the Condition Register remains set to Not Zero and the ODW is ignored. A UBC channel, once activated, will not accept an ODW instruction until the word count is complete, i.e., all words in the block have been transferred and WCR equals zero.

IDW Input Data Word

Formula 0072.*+C:U **Affected** A,C



Operation

A data word is transferred from the specified channel/unit combination to the A Register.

Notes

The Condition Register is cleared, then set to Zero if the I/O channel is ready. If the channel is not ready or data from the specified unit is not available, the Condition Register is set to Not Zero to allow a programmed delay.

If the selected unit is in the process of executing a command as the result of a previous OCW instruction, the channel indicates not ready (Condition Register remains set to Not Zero) and the IDW is ignored. At the completion of the OCW, the Condition Register is set to Zero and the IDW instruction is accepted by the channel.

If the selected unit is in the process of receiving data as a result of an ODW instruction and data is available from the unit, an ODW will be accepted and the Condition Register set to Zero.

If the IDW instruction specifies a unit other than the unit connected to the channel, the channel indicates not ready (Condition Register remains set to Not Zero), ignores the instruction, and initiates a disconnect/connect sequence.

If an IDW instruction specifies a UBC Channel that is engaged in a block transfer, the Condition Register remains set to Not Zero (channel not ready) and the instruction is ignored. A UBC channel, once activated, will not accept an IDW instruction until the word count is complete; i.e., all words in the block have been transferred and WCR equals zero.

When a UBC channel is employed in a single-word programmed data transfer, an IDW instruction returns a Not Ready (C Register = Not Zero) condition if the channel is currently processing an output command. This situation is in effect regardless of the status of the input data from the peripheral unit.

The only valid unit code for a DMACP channel is 10_8; all others are rejected.

If the Merge bit (*) is ZERO the A Register is cleared prior to the data transfer. Input data is right-justified in the A Register.

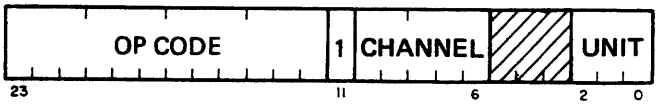
If the Merge Bit is a ONE, an OR is performed between the previous contents of the A Register and the incoming data word. This feature, in conjunction with a shift operation, allows input data characters to be packed in the A Register.

Example: Two 12-bit data characters are to be packed in the A Register.

IDW	'0102	Clear A and load first character from channel 01, Unit 02.
BNZ	*-1	Wait if busy
LLA	12	Shift the contents of A left 12 bits
IDW*	'0102	Merge second character
BNZ	*-1	Wait if busy
	...	Continue

OAW Output Address Word

Formula 0071.40+C:U **Affected** C



Operation

The contents of the A Register are transferred to an appropriate register in the specified channel, or unit in XBC Channel executions.

Notes

The Condition Register is cleared, then set to Zero if the I/O channel is ready.

The unit is addressed in XBC and DMACP channels (bits 0-2) and IBC channels (bits 0, 1) only.

A UBC channel will always indicate ready for an OAW instruction. However, if the OAW specifies an invalid channel number, it will receive a "not ready" indication and the Condition Register remains set to Not Zero. Since XBC/IBC channels involve a unit address, the unit must be "connected" before the instruction can be executed.

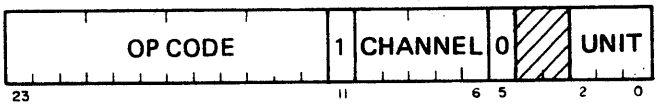
The OAW instruction does not activate a block-transfer channel. It transfers the starting address of the first of two parameter words from the A Register to the TAR or PAR in the selected channel. In XBC channel operations the OAW instruction transfers the contents of the A Register to the unit; the channel has no register dedicated to this function.

If an OAW instruction addresses a UBC channel during a block transfer sequence, the sequence will be terminated.

If the OAW instruction addresses a PIOC, the Condition Register remains set to Not Zero; the instruction is executed automatically. In this instruction the four least significant bits (3-0) of the A Register are transferred to the Interrupt Generator logic. These bits (unitarily) control the triggering of the one-to-four 1 microsecond interrupt pulses.

IAW Input Address Word

Formula 0073.40+C.0:U **Affected** A,C



Operation

The current contents of the Transfer Address Register (TAR) in the specified channel (UBC, IBC, or DMACP) are transferred to the A Register.

Notes

The Condition Register is cleared, then set to zero if the I/O channel is ready. If the IAW instruction specifies an invalid channel, the Condition Register remains set to Not Zero indicating channel not ready.

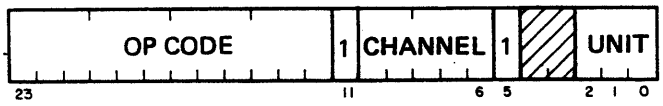
The unit is addressed in IBC and DMACP channels only.

Bit 5 at the ZERO level distinguishes between the IAW and IPW instructions.

The UBC channel always indicates ready to an IAW instruction. The IBC channel must go to "not busy" before executing the instruction.

IPW Input Parameter Word

Formula 0073.40+C.4:U **Affected** A,C



Operation

The current contents of the Parameter Address Register (PAR) in the specified channel (UBC, IBC, or DMACP) are transferred to the A Register.

Notes

The Condition Register is cleared, then set to zero if the I/O channel is ready. If the IPW instruction specifies an invalid channel, the Condition Register remains set to Not Zero, indicating channel not ready.

The unit is addressed in IBC and DMACP channels only.

IPW instructions addressed to an IBC channel must specify, via the unit address, which of three possible channel PARs is read.

Bit 5 at the ONE level distinguishes between the IPW and IAW instructions.

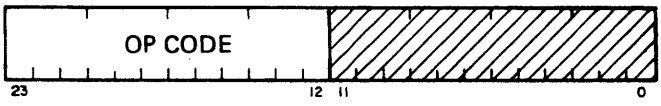
UBC channels always indicate ready to an IPW instruction. The IBC channel must go to "not busy" before executing the instruction.

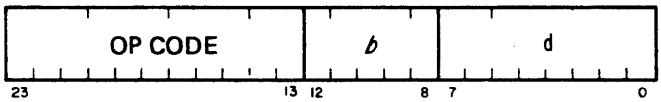
BIT PROCESSOR INSTRUCTIONS

The bit (Boolean function) processor group of instructions include branches, logical manipulation, and interrogation of a specified bit selected from an effective memory address or the H Register. In most instances, bit 2 (Zero/Not Zero) of the Condition Register is used to display either the result of an operation or the status of a bit before the operation is performed.

The bit processor employs two instruction word formats. The first format uses an Op Code (bits 23-12) to specify the operation to be performed. The remaining 12 bits (bits

11-0) are undefined. The second instruction format contains a displacement, bit specification, and an Op Code. Eight bits (bits 7-0) are added to the base address contained in the V Register to obtain a displacement from the base address which is an effective memory address for the word containing the bit in question. Five bits (bits 12-8) are used to select a specific bit in the effective memory address for an operation as specified in the 11-bit (bits 23-13) Op Code. Both instructions word formats are illustrated below.

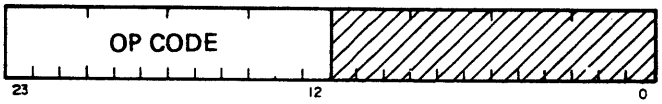




The following instructions are included in the bit processor group.

TZH Transfer Zero to H

Formula 7742. **Affected** H,C



Operation

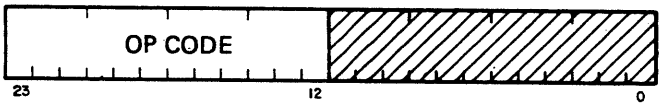
A ZERO is placed in the H Register. The Condition Register is set to reflect the original contents of H.

Note

If the original contents of the H Register were ZERO, Condition Register Bit 2 is set to 1 (Zero). If the contents were ONE, Bit 2 is set to 0 (Not Zero).

TFH Transfer Flag to H

Formula 7743. **Affected** H,C



Operation

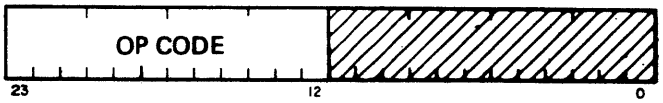
A ONE is placed in the H Register and the Condition Register is set to reflect the original contents of H.

Note

If the original contents of the H Register were ZERO, Condition Register Bit 2 is set to 1 (Zero). If the contents were ONE, Bit 2 is set to 0 (Not Zero).

TKV Transfer K to V

Formula 7744. **Affected** V

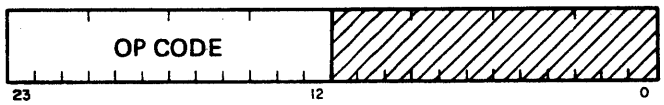


Operation

The 18 least significant bits of the K Register replace the present contents of the V Register. The Condition Register is unaffected.

TVK Transfer V to K

Formula 7745. **Affected** K

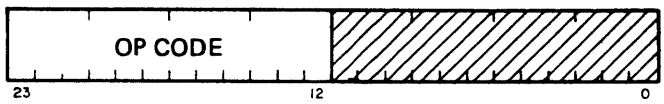


Operation

The contents of the V Register are transferred to the 18 least significant bit positions of the K Register. Bits 23-18 of the K Register are reset to ZEROs. The Condition Register is unaffected.

QBH Query Bit of H

Formula 7746. **Affected** C



Operation

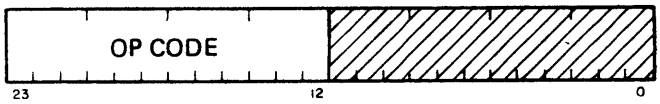
The H Register bit is tested and the Condition Register is set to display the result of the query.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the content is ONE, Bit 2 is set to 0 (Not Zero).

NHH Negate of H to H

Formula 7747. **Affected** H,C



Operation

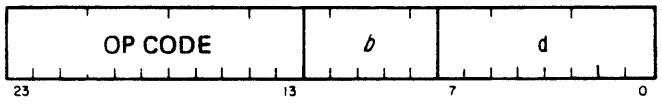
The current content of the H Register is complemented and returned to H. The Condition Register is set to display the result.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the content is ONE, Bit 2 is set to 0 (Not Zero).

DMH Dot Memory with H

Formula 7750. *b*:d **Affected** H,C



Operation

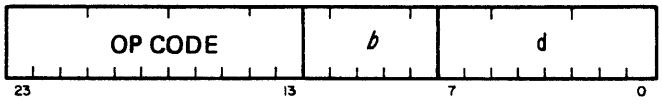
A logical AND is performed between the selected bit in the effective memory address and the contents of the H Register. The result is returned to the H Register and the Condition Register is set to display the result.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the content is ONE, Bit 2 is set to 0 (Not Zero).

DNH Dot Not (memory) with H

Formula 7752. *b*:d **Affected** H,C



Operation

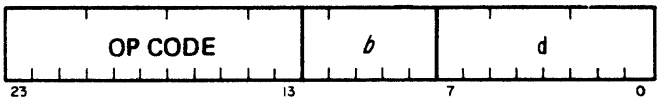
A logical AND is performed between the complement of the selected bit in the effective memory address and the content of the H Register. The result is returned to the H Register and the Condition Register is set to display the result.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the content is ONE, Bit 2 is set to 0 (Not Zero).

OMH OR Memory with H

Formula 7754. *b*:d **Affected** H,C



Operation

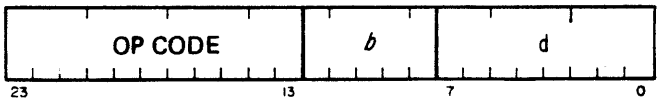
A logical OR is performed between the selected bit in the effective memory address and the content of the H Register. The Condition Register is set to display the result.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the content is ONE, Bit 2 is set to 0 (Not Zero).

ONH OR Not (memory) with H

Formula 7756. *b*:d **Affected** H,C



Operation

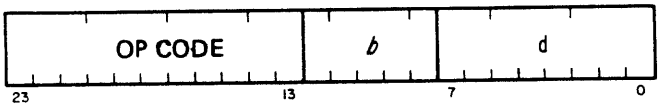
A logical OR is performed between the complement of the selected bit in the effective memory address and the content of the H Register. The Condition Register is set to display the result.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the content is ONE, Bit 2 is set to 0 (Not Zero).

XMH Exclusive-OR Memory with H

Formula 7760.*b*:d **Affected** H,C



Operation

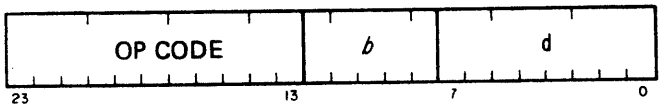
An exclusive-OR function is performed between the selected bit in the effective memory address and the content of the H Register. The Condition Register is set to display the result.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the content is ONE, Bit 2 is set to 0 (Not Zero).

XNH Exclusive-OR Not (memory) with H

Formula 7762.*b*:d **Affected** H,C



Operation

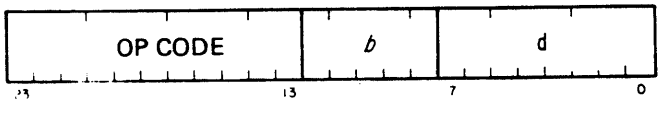
An exclusive-OR function is performed between the complement of the selected bit in the effective memory address and the content of the H Register. The Condition Register is set to display the result.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the content is ONE, Bit 2 is set to 0 (Not Zero).

TMH Transfer Memory to H

Formula 7764.*b*:d **Affected** H,C



Operation

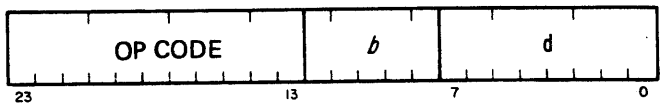
The selected bit in the effective memory address is transferred to the H Register. The Condition Register is set to display the resultant content of the H Register.

Note

The Condition Register is cleared. If the resultant content of the H Register is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the resultant content is ONE, bit 2 is set to 0 (Not Zero).

QBM Query Bit of Memory

Formula 7766.*b*:d **Affected** C



Operation

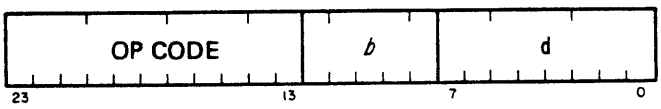
The selected bit in the effective memory address is tested and the Condition Register is set to display the result of the query.

Note

The Condition Register is cleared. If the resultant content of memory is ZERO, Condition Register Bit 2 is set to 1 (Zero). If the resultant content is ONE, Bit 2 is set to 0 (Not Zero).

THM Transfer H to Memory

Formula 7770.*b*:d **Affected** M

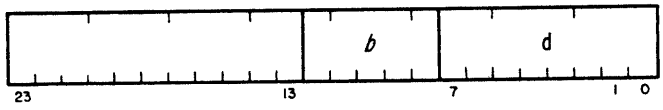


Operation

The content of the H Register is placed in the selected bit position in the effective memory address. The Condition Register is not affected.

FBM Flag Bit of Memory

Formula 7772.*b*:d **Affected** M,C



Operation

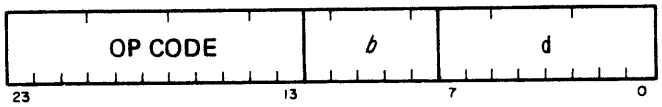
A ONE is placed in the selected bit position in the effective memory address. The Condition Register is set to display the original state of the selected bit in memory.

Note

If the original state of the selected bit in memory was ZERO, Condition Register Bit 2 is set to 1 (Zero). If the original state was ONE, Bit 2 is set to 0 (Not Zero).

ZBM Zero Bit of Memory

Formula 7774.*b*:d **Affected** M,C



Operation

A ZERO is transferred to the selected bit position in the effective memory address. The Condition Register is set to display the original state of the selected bit in memory.

Note

If the original state of the selected bit in memory was ZERO, Condition Register Bit 2 is set to 1 (Zero). If the original state was ONE, Bit 2 is set to 0 (Not Zero).

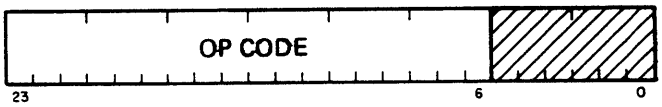
VIRTUAL MEMORY INSTRUCTIONS

The majority of the virtual memory instructions involve transfers between the paging registers and the A, E and D Registers. The remaining instructions are special control operations for activating and testing the virtual memory logic.

The following instructions are included in the virtual memory group.

TDS Transfer Double to Source and Destination Registers

Formula 006410. **Affected** VSR,VDR

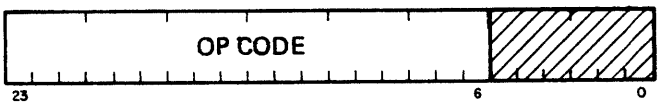


Operation

Bits 11-0 of the A Register replace the previous contents of the Virtual Destination Register (VDR) and bits 11-0 of the E Register replace the previous contents of the Virtual Source Register (VSR). The contents of A and E are not changed.

TSD Transfer Source and Destination Registers to D

Formula 006510. **Affected** A,E

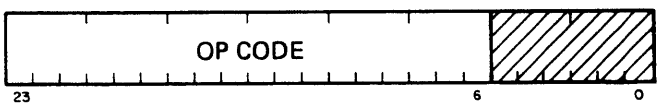


Operation

The contents of the Virtual Source Register (VSR) replace the previous contents of bits 11-0 of the E Register; the contents of the Virtual Destination Register (VDR) replace the previous contents of bits 11-0 of the A Register. Bits 23-12 of both A and E are cleared (reset to ZEROs). The contents of the VSR and VDR are not changed.

TAR Transfer A to 1 Virtual Address Register

Formula 006050. **Affected** VAR,VDR

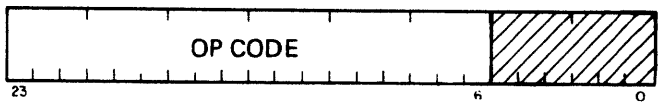


Operation

Bits 9-0 of the A Register replace the previous contents of the Virtual Address Register (VAR) specified by the Virtual Destination Register (VDR). The VDR is incremented by one. The contents of the A Register are not changed.

TDR Transfer Double to 2 Virtual Address Registers

Formula 006430. **Affected** VAR(1),VAR(2), VDR

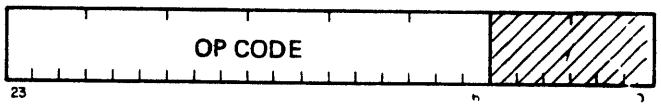


Operation

Bits 9-0 of the E Register replace the previous contents of the Virtual Address Register (VAR) specified by the Virtual Destination Register (VDR); the VDR is then incremented by one to specify the second VAR. Bits 9-0 of A replace the previous contents of the second VAR. The VDR is again incremented by one. The contents of the E and A Registers are not changed.

TRD Transfer 2 Virtual Address Registers to Double

Formula 006530. **Affected** E,A,VSR



Operation

The contents of the Virtual Address Register (VAR) specified by the Virtual Source Register (VSR) replace the previous contents of bits 23-16, 9, and 8 of the E Register. The VSR is then incremented by one to specify the second VAR. The contents of the second VAR replace the previous contents of bits 23-16, 9, and 8 of the A Register. The VSR is again incremented by one. Bits 15-10 and 7-0 of both E and A are cleared (reset to ZERO).

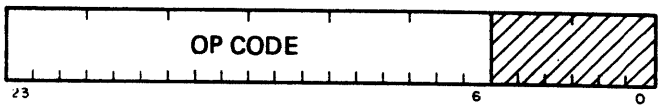
Note

The Transfer format is shown below.

From	Bit No.	To	Bit No.
VAR#1/#2	9	E/A	9
VAR#1/#2	8	E/A	8
VAR#1/#2	7	E/A	23
VAR#1/#2	6	E/A	22
VAR#1/#2	5	E/A	21
VAR#1/#2	4	E/A	20
VAR#1/#2	3	E/A	19
VAR#1/#2	2	E/A	18
VAR#1/#2	1	E/A	17
VAR#1/#2	0	E/A	16

TDP Transfer Double to Paging Limit Registers

Formula 006450. **Affected** VBR,VLR

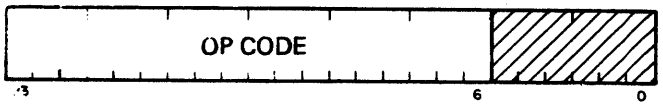


Operation

Bits 11-0 of the A Register replace the previous contents of the Virtual Base Register (VBR), and bits 9-0 of the E Register replace the previous contents of the Virtual Limit Register (VLR). The contents of A and E are not changed.

TPD Transfer Paging Limit Registers to Double

Formula 006550. **Affected** E,A

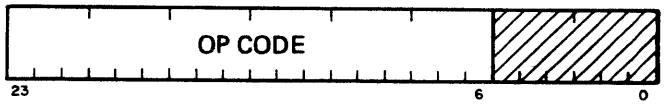


Operation

The contents of the Virtual Base Register (VBR) replace the previous contents of A Register bits 11-0, and the contents of the Virtual Limit Register (VLR) replace the previous contents of E Register bits 9-0. The remaining bits of both A and E are reset to ZEROs. The contents of the VBR and VLR are not changed.

TUD Transfer Usage Base Register and Demand Page Register to Double

Formula 006570. **Affected** E,A



Operation

The contents of the Virtual Demand Page Register (VPR) replace the previous contents of A Register bits 23-16 and 3-0, and the contents of the Virtual Usage Base Register (VUB) replace the previous contents of E Register bits 7-0. A Register bits 15-4 and E Register bits 23-8 are reset to ZEROs. The contents of the VPR and VUB are not changed.

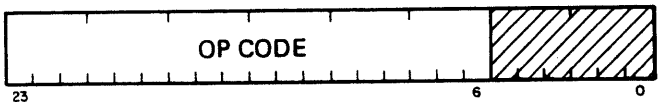
Note

The VPR transfer format is as follows:

From	To
VPR Bit 11	A23
VPR Bit 10	A22
VPR Bit 9	A21
VPR Bit 8	A20
VPR Bit 7	A19
VPR Bit 6	A18
VPR Bit 5	A17
VPR Bit 4	A16
VPR Bit 3	A3
VPR Bit 2	A2
VPR Bit 1	A1
VPR Bit 0	A0

TEU Transfer E to Virtual Usage Base Register

Formula 006470. **Affected** VUB

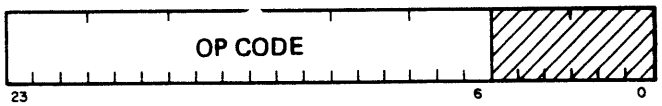


Operation

The contents of E Register bits 7-0 replace the previous contents of the Virtual Usage Base Register (VUB). The E Register contents are not changed.

QUR Query Usage Register

Formula 007030. **Affected** VUR,VUB,C

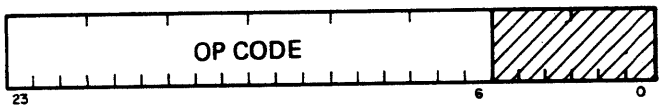


Operation

The contents of the Virtual Usage Register (VUR) – specified by the Virtual Usage Base Register (VUB) – is tested. The Condition Register is set to "Not Zero" or "Zero" if the content of the VUR is ONE or ZERO, respectively. The specified VUR is cleared and the VUB is incremented by one.

QNR Query Not-modified Register

Formula 007070. **Affected** VNR,VUB,C

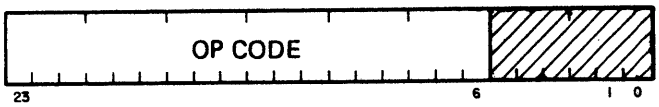


Operation

The contents of the Virtual Not-modified Register (VNR) – specified by the Virtual Usage Base Register (VUB) – is tested. The Condition Register is set to "Not Zero" or "Zero" if the content of the VNR is ONE or ZERO, respectively. The specified VNR is cleared and the VUB is incremented by one.

ROM Release Operand Mode

Formula 006010. **Affected** None



Operation

The operand address of the following instruction is translated.

Notes

If bit 8 of the Virtual Limit Register is a ONE, the ROM will be nullified; i.e., the following instruction will not be translated.

The ROM, together with the following instruction, will be treated as an USP or AOM instruction with respect to a demand page.

No double-word instructions (USP, AOM) are permitted after a ROM.

If an unconditional branch is executed following the ROM, the User Mode will automatically be established.

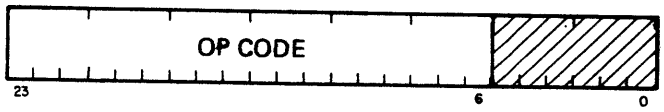
If a conditional branch follows the ROM, the User Mode will be established regardless of the outcome of the conditional test.

The User Mode is not established during indirect addressing until completion of the indirect chain.

Execution of the ROM instruction inhibits merging of the Map bit (PC15) in the following instruction, regardless of the state of bit 8 of the VLR.

RUM Release User Mode

Formula 006030. **Affected** None



Operation

The User Mode is established upon completion of the following instruction.

Notes

In practice, the instruction following the RUM should always be a branch.

No conditional branches are permitted following the RUM.

The User Mode will not be activated until after the operand address has been calculated. Indexing and/or indirect references are permitted. When indirected, the User Mode will not be established until completion of the indirect chain.

The RUM, together with the following instruction, will be handled as an EXM instruction with respect to a demand page.

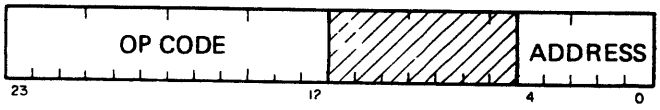
Execution of the RUM instruction inhibits merging of the Map bit (PC15).

PROGRAM RESTRICT INSTRUCTIONS

The following instructions provide control for the program restrict system in the Program Restrict and Instruction Trap.

BLU Branch and Link Unrestricted

Formula 0067:a **Affected** J,P



Operation

The next sequential address (program address +1) replaces the contents of the J Register and the contents of the P Register (current program address) are replaced by the 5-bit immediate memory address.

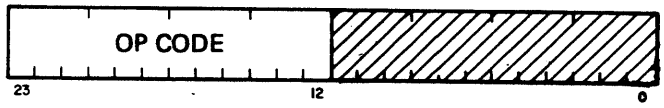
Notes

If Program Restrict is enabled, execution of the BLU instruction will turn off the Program Restricted Flag (PRF). If the computer is in a HALT condition and the PRF is on, the BLU instruction will be treated as a NOP instruction.

If virtual memory is enabled, execution of the BLU instruction will automatically establish the Monitor Mode. The 5-bit immediate memory address will not be mapped. Bit 20 of the J Register will be set (ONE) if the system was in the User Mode, and reset (ZERO) if the Monitor Mode was active when the BLU was executed.

TDL Transfer Double to Limit Registers

Formula 0056. **Affected** LL,UL

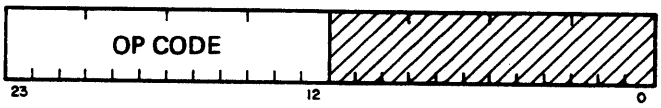


Operation

The contents of bits E17-E0 replace the previous contents of the Lower Limit (LL) Register and the contents of bits A17-A0 replace the previous contents of the Upper Limit (UL) Register. Bits A21 and A22 set the restrict mode flags.

TLD Transfer Limit Registers to Double

Formula 0057. **Affected** E,A



Operation

The contents of the limit registers replace the previous contents of the D Register (E and A). The Upper Limit Register contents are transferred to bits A17-A0 and the contents of the Lower Limit Register are transferred to E17-E0. The states of the restrict mode flags are transferred to bits A21 and A22. All other bits in E and A are reset to ZERO.

PRIORITY INTERRUPT CONTROL INSTRUCTIONS

The priority interrupt instruction group provides the means for program control of external interrupts. External interrupts may be selectively armed, disarmed, enabled or inhibited under program control. Other instructions provide the means for holding and releasing external interrupts, while others are available for transferring control upon interrupt detection. For a detailed description of the priority interrupt system, refer to Section V of this manual.

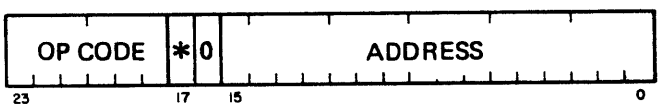
If the system is equipped with the Program Restrict and Instruction Trap, the following priority interrupt instructions will be affected.

a) Hold External Interrupts (HXI)
b) Release External Interrupts (RXI)
c) Unitarily Arm Group 1 Interrupts (UA1)
d) Unitarily Disarm Group 1 Interrupts (UD1)
e) Unitarily Enable Group 1 Interrupts (UE1)
f) Unitarily Inhibit Group 1 Interrupts (UI1)
g) Transfer Double to Group 1 (TD1)
h) Transfer Double to Group 1 (TD4)

The following instructions are included in the priority interrupt group.

BSL Branch and Save return Long

Formula 25.*+0:A **Affected** P



Operation

The next sequential address (program address +1), along with the contents of the Condition Register are stored in the 16-bit effective memory address (EMA). The contents of Register P (current program address) are then replaced by the address following the effective memory address (EMA+1).

Notes

This instruction is used to enter an interrupt subroutine because it provides a means of returning to the main program at the point of interrupt and saves the machine status (condition) at the time of the interrupt.

The contents of the Condition Register are stored in bit positions 19-16 of the EMA and the return address (program address +1) is stored in bits 15-0. The remaining bits are set to ZEROs; however, refer to last note for variation on bit 20.

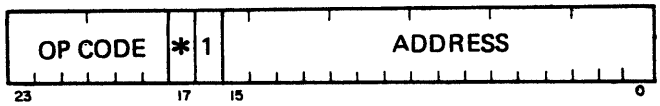
The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

When an interrupt occurs, the status of the virtual memory system is recorded. Bit 20 is set to ONE if the system is in the User Mode at the time of interrupt; bit 20 is set to ZERO if the Monitor Mode is active.

BRL Branch and Reset interrupt Long

Formula 25.*+2:A **Affected** C,P



Operation

The highest-level active interrupt is reset (i.e., returned to the inactive state) and the contents of the P Register (current program address) are replaced by the 16-bit effective memory address.

Notes

BRL is normally used to exit an interrupt subroutine. If BRL contains an indirect reference, the last word in the indirect address chain contains the previous status of the virtual memory system in bit M20, the previous machine status (i.e., C Register contents at the time of the interrupt) in bit positions M19-M16, and the return address in bit positions M15-M0 as a result of the BSL instruction. The C Register is restored and the program branches to the return address (restarting the machine to the pre-interrupt status).

Example:

```
          .
          .
          TMA
L         AMA
          SMA          Interrupt occurs (EXM K).
          .
          .
K         BSL    M     Dedicated interrupt location.
M         ***          M becomes L+1 as a result of BSL
          .            at K. The C Register contents are
          .            stored in M19-M16.
          BRL*   M
```

The BRL will not reset the interrupt if external interrupts have been held by an HXI instruction. Control will be returned to the effective memory address.

Those executive traps, which are not affected by the HXI instruction, will be reset by the BRL.

The immediate memory reference cannot be indexed; however, indexing indirect references is permitted, e.g.

```
          BRL*   X
          .
          .
X         DAC    Y,K
```

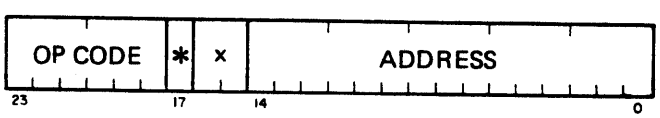
If the BRL instruction is not indirected, the Condition Register is not affected.

External interrupts are prohibited for the period of one instruction following this instruction.

In virtual memory systems, if an indirect BRL is executed in Monitor Mode, bit 20 of the effective memory address determines mode of operation machine returns to. If bit 20 is set, User Mode is established; if reset, the Monitor Mode is established.

HTx Hold Interrupts and Transfer Register to Memory

Formula 27.*+x:a **Affected** M



Operation

The contents of the I, J, or K Register replace the previous contents of the effective memory address and external interrupts are prohibited for the period of one instruction following the execution of this instruction.

Notes

HTx is not a computer instruction mnemonic but represents a family of instruction mnemonics. x is coded as follows to select one of the index registers.

x = 1 (I)
2 (J)
3 (K)

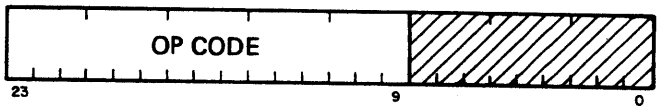
A code of 27.*+1:a, for example, implements the Hold Interrupt and Transfer I to Memory (HTI) instruction.

The immediate memory reference cannot be indexed; however, indexing of indirect references is permitted, e.g.,

```
          HTI*   M
          .
          .
M         DAC    A,K
```

HXI Hold External Interrupts

Formula 00660. **Affected** None



Operation

The activation of any external interrupt is prohibited. The prohibition is effective immediately upon execution of the instruction and lasts until the interrupts are released (see RXI instruction). Executive traps (Group 0, Levels 5-7) are prohibited from becoming active while the HXI is in effect.

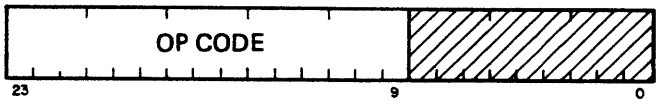
Notes

Only the three executive traps mentioned are affected by this instruction.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

RXI Release External Interrupts

Formula 00664. **Affected** None



Operation

The prohibition imposed by the HXI instruction is removed, allowing any external interrupt to be activated 1 cycle after this instruction. This permits the next sequential instruction to be executed without external interruption.

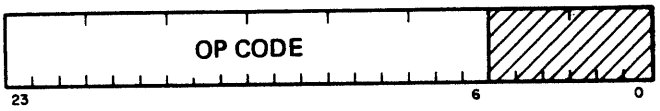
Notes

If any of the affected executive traps have been triggered while an HXI was in effect, the highest level will come in first after the RXI instruction.

External interrupts are prohibited for the period of one instruction following the execution of the instruction.

TD1 Transfer Double to Group 1

Formula 006401. **Affected** 1 A/D, 1 E/I



Operation

The contents of the D Register (E and A) replace the previous contents of the Arm/Disarm (A/D) and Enable/Inhibit (E/I) Registers of interrupt group 1. The contents of E are transferred to the A/D Register and the contents of A are transferred to the E/I Register.

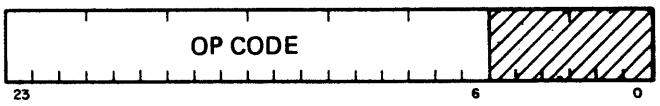
Notes

The external interrupt structure is cleared by the execution of this instruction.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

T1D Transfer Group 1 to Double

Formula 006501. **Affected** E,A



Operation

The contents of the Arm/Disarm (A/D) and Enable/Inhibit (E/I) Registers of interrupt group 1 replace the previous contents of the D Register (E and A). The contents of the A/D Register are transferred to the E Register and the contents of the E/I Register are transferred to the A Register.

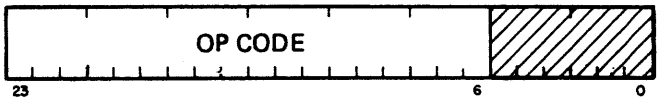
Notes

The states of the external interrupts are not affected by the execution of this instruction.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

T4D Transfer Group 1 to Double

Formula 006541. **Affected** E,A



Operation

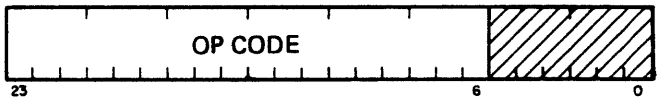
The contents of the Request and Active Registers of interrupt group 1 replace the previous contents of the D Register (E and A). The contents of the Request Register are transferred to E, and the contents of the Active Register are transferred to A.

Note

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

TD4 Transfer Double to Group 1

Formula 006441. **Affected** 1 Request, Active



Operation

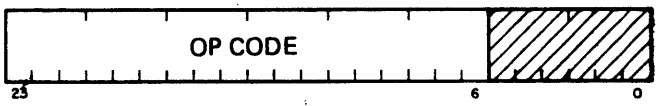
If armed, the contents of the D Register (E and A) are ORed with the current contents of the Request and Active Registers of interrupt group 1. The contents of E are ORed with the request Register and the contents of A are ORed with the Active Register.

Note

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

UA1 Unitarily Arm group 1 interrupts

Formula 006001. **Affected** 1 A/D



Operation

Any number of the 24 interrupt levels in group 1 are selectively armed; i.e., the selected bit(s) of the Arm/Disarm (A/D) Register is (are) set to ONE.

Notes

The corresponding bit(s) of the A Register must be set to select the appropriate level(s) prior to executing this instruction.

Example: Arm levels 1 and 3, group 1

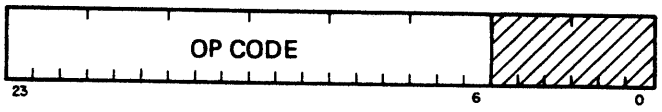
TOA	B1B3	Select levels 1 and 3 (set bits 1 and 3 of A)
UA1		Arm selected levels of group 1

Execution of this instruction does not clear the interrupt structure and, therefore, does not affect any interrupt levels other than those selected. If a level selected for arming is already armed, it is not cleared by the execution of this instruction.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

UD1 Unitarily Disarm Group 1 Interrupts

Formula 006101. **Affected** 1 A/D



Operation

Any number of the 24 interrupts levels in group 1 are selectively disarmed i.e., the selected bits of the Arm/Disarm (A/D) Register are reset to ZERO.

Notes

The corresponding bit(s) of the A Register must be set to select the appropriate level(s) prior to executing this instruction.

Example: Disarm level 2, group 1

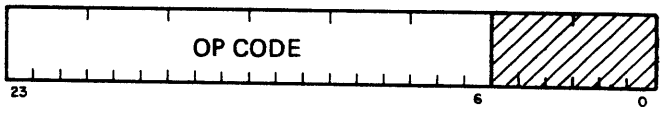
TOA	B2	Select level 2 (set bit 2 of A)
UD1		Disarm selected level of group 1

Execution of this instruction will clear only those levels which are selected. The remaining levels will not be affected.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

UE1 Unitarily Enable Group 1 Interrupts

Formula 006201. **Affected** 1 E/I



Operation

Any number of the 24 interrupt levels in group 1 are selectively enabled, i.e., the selected bits of the Enable/Inhibit (E/I) Register are set to ONE.

Notes

The corresponding bit(s) of the A Register must be set to select the appropriate level(s) prior to executing this instruction.

Example: Enable levels 0, 2 and 5, group 1

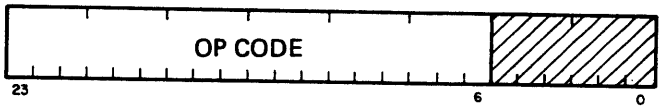
TOA	B0B2B5	Select levels 0, 2 5 (set bits 0, 2 and 5 of A)
UE1		Enable selected levels of group 1

Execution of this instruction does not clear the interrupt structure and, therefore, does not affect any interrupt levels other than those selected. If a level selected for enabling is already enabled, it is not cleared by the execution of this instruction.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

UI1 Unitarily Inhibit Group 1 Interrupts

Formula 006301. **Affected** 1 E/I



Operation

Any number of the 24 interrupt levels in group 1 are selectively inhibited; i.e., the selected bits of the Enable/Inhibit (E/I) Register are reset to ZERO.

Notes

The corresponding bit(s) of the A Register must be set to select the appropriate level(s) prior to executing this instruction.

Example: Inhibit levels 1, 4 and 7 of group 1

```
TOA    B1B4B7    Select levels 1, 4, 7
                 (set bits 1, 4 and 7 of A)
UI1              Inhibit selected levels of group 1
```

Execution of this instruction does not clear the interrupt structure and, therefore, does not affect any interrupt levels other than those selected. If one or more of the selected levels is active upon execution of this instruction, the level(s) will be placed in a "permissive" state.

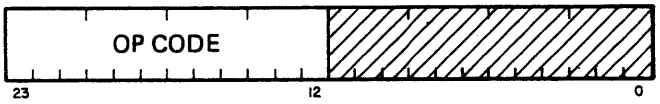
External interrupts are prohibited for the period of one instruction following execution of this instruction.

MISCELLANEOUS INSTRUCTIONS

The following instructions are included in the miscellaneous group because they do not fall into any defined functional group.

HLT Halt

Formula 0000. **Affected** P

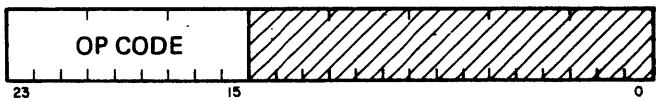


Operation

The program address (i.e., the contents of the P Register) is advanced by one and program execution is terminated. When the RUN switch is depressed, execution will begin at the location defined by the program address.

NOP No Operation

Formula 620. **Affected** P

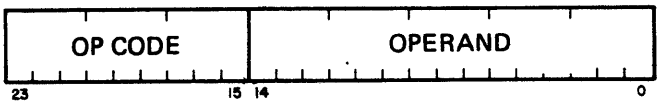


Operation

The program address is advanced by one and program execution continues with the next instruction.

GAP Generate Argument Pointer

Formula 244:o **Affected** I,J



Operation

The contents of the J Register are assumed to be the first address in an indirect memory reference sequence. The effective memory address derived from this indirect sequence replaces the previous contents of the I Register. The contents of the J Register and the 15-bit operand are added, and the result is placed in the J Register.

Notes

If the final EMA in the indirect sequence is a DAC format, bits 15-0 replace the contents of I. If the final EMA is a LAC, bits 20-0 replace the contents of I.

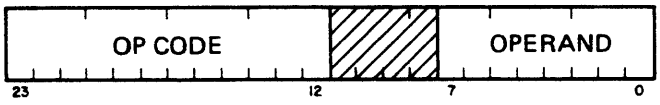
The purpose of a GAP instruction is to generate an effective memory address which points to one or more data words not directly available to a subroutine. This is illustrated in the following example where subroutine B requires the data contained in location Y.

```
A     BLJ     B     (J) = C, (P) = B
C     DAC*    X
D     ...           RETURN
      ...
X     DAC     Y
Y     DATA    2
      ...
B     GAP     1     (I) = Y, (J) = (J) +1
      TMA     0,I   (A) = 2
      BUC     0,J   (P) = D
```

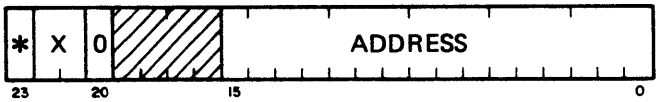
USP or USP DAC LAC — Update Stack Pointer

Formula 0055:o (word 1) **Affected** K,C
*+X.0:A or *+X.1:A (word 2)

WORD 1 (USP)

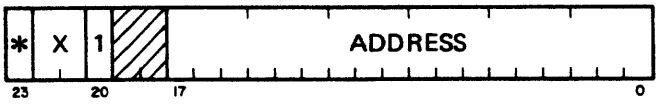


WORD 2 (DAC)



or

WORD 2 (LAC)



Operation

The contents of the K Register are replaced by the contents of the effective memory address. The 8-bit signed operand is then added to the contents of the effective memory address.

Notes

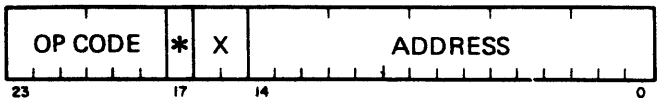
```
        BLJ     ENT     Call re-entrant routine
        ...
ENT     TRM*    SP      Save registers in stack
        USP     5       Update Stack Pointer
                        [(K) = stack, (SP)
                        = stack +5]
        DAC     SP
        ...
        HTK     SP      Reset stack pointer
        TMR*    SP      Restore registers
        BUC     0,J     Return
SP      DAC     STACK   Stack pointer
STACK   BLOK    5N      Where N represents
                        maximum number of
                        re-entrant levels
```

The Condition Register is set to reflect the result of the operand addition.

External interrupts are prohibited for the period of one instruction following this instruction.

EXM — Execute Memory

Formula 40.*+X:a **Affected** See Notes



Operation

The instruction located in the effective memory address is executed as though it were at the address of the EXM.

Notes

In the case that the referenced instruction is a two word instruction, the second word must follow the EXM.

Example:

```
        EXM     M
        DAC     L       Second word
        ...
M       AOM     10      Two word instruction
        AOM     20
        AOM     30
```

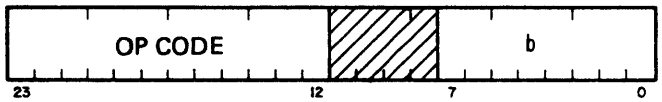
The registers affected will depend on the instruction in the effective memory address.

External interrupts are prohibited for the period of one instruction following the execution of this instruction.

The program address (contents of P Register) is not advanced when this instruction is executed.

QBB — Query Bits of Byte

Formula 0011:b **Affected** C



Operation

A logical AND is performed between operand bits 7-0 and the contents of the B Register. The Condition Register is set according to the status of the result; i.e., Positive, Negative, or Zero.

Note

Examples:

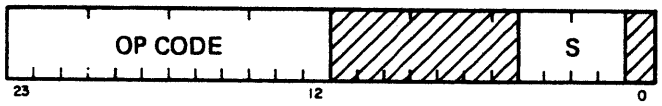
```
        (1) TOA     B7      A = '00000200     C = Positive
            ...
            ...
            QBB     B7                        C = Negative

        (2) TOA     B6      A = '00000100     C = Positive
            ...
            ...
            QBB     B6                        C = Positive

        (3) TNA     1       A = '77777777     C = Negative
            ...
            ...
            DMA     MASK    A = '40000000     C = Negative
            ...
            ...
MASK    DATA        '40000000
```

QSS Query Sense Switches

Formula 0001:s **Affected** C



Operation

A logical AND is performed between operand bits 4-1 and the state(s) of the sense switches. The Condition Register is set to Positive, or Zero based on the result.

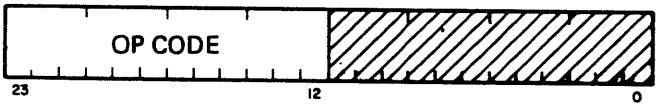
Note

Example: Test to see if either SS2 or SS3 are on, or if both are on.

```
QSS     B2B3
```

EZB Extend Zeros from Byte

Formula 0007. **Affected** A



Operation

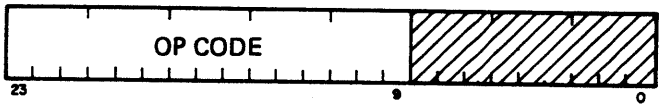
Bit positions A23-A8 are set to ZERO. The contents of the B Register (A7-A0) are not affected.

Note

The Condition Register is not affected.

HIT Hold Interval Timer

Formula 00770. **Affected** None

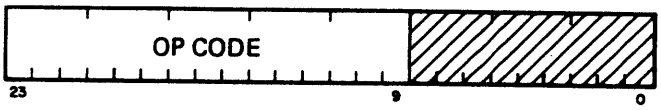


Operation

The CPU's Interval Timer is halted and will remain so until released by an RPT or RCT instruction.

RPT Release Processor Time

Formula 00774. **Affected** None



Operation

The CPU's Interval Timer is started; i.e., allowed to begin counting CPU time.

Notes

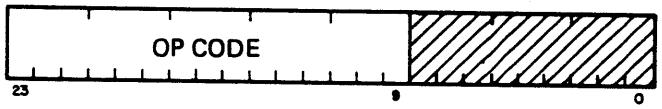
The Processor Time Mode allows the Interval Timer to count CPU time only. Counting is inhibited when an I/O block controller channel takes a memory cycle or when an interrupt is active.

Once started, the timer counts until held by a HIT instruction or until the CPU is halted.

At each one microsecond interval, the contents of the T Register are decremented by one and tested for zero. If the contents of T are zero, an executive interrupt is triggered. The interrupt does not stop the timer.

RCT Release Clock Time

Formula 00776. **Affected** None



Operation

The CPU's Interval Timer is started; i.e., allowed to begin counting continuously.

Notes

The Clock Time Mode causes the Interval Timer to count continuously.

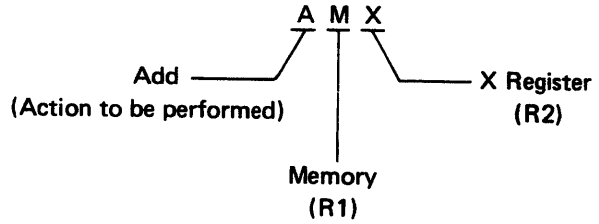
Once started, the timer will count until held by a HIT instruction.

At each one microsecond interval, the contents of the T Register are decremented by one and tested for zero. If the contents of T are zero, an executive interrupt is triggered. The interrupt does not stop the timer.

SCIENTIFIC ARITHMETIC UNIT INSTRUCTIONS

The instruction set for the Scientific Arithmetic Unit is divided into five functional groups; arithmetic, transfer, branch, compare, and interrupt control. Concurrent time, if any, occurs after the instruction has been initiated by the SAU. The SAU is designed to operate on normalized floating point numbers, and all descriptions of the arithmetic instructions are based on this fact. If an unnormalized operand is used in an arithmetic operation the results are not considered valid. The results of an arithmetic operation are truncated, not rounded.

Standard arithmetic instructions – add, subtract, multiply, and divide – as well as square, square root, fix and float are included in the group. The instruction mnemonics provide a brief definition of specific operations to be performed. The first letter in the mnemonic specifies the action or type of operation that is to be performed. The second letter identifies the first quantity or reference (R1) to be used in the operation, and the third letter identifies the second reference (R2). For example:



In the majority of SAU arithmetic instructions, the result of the operation remains in R2 while R1 remains unchanged (except where R1 and R2 are the same).

Unless otherwise noted, each arithmetic operation sets a bit in the SAU condition (Y) register to reflect the status of the result. Various conditions are described below:

a) Positive – The result is arithmetically greater than zero, indicated by a ONE in bit position 3 of the Y Register. A ZERO in bit position 3 indicates "Not Positive".

b) Zero – All of the mantissa bits comprising the quantity under consideration are ZERO and the exponent is '201, indicated by a ONE in bit position 2 of the Y Register. A ZERO in bit position 2 indicates "Not Zero".

c) Negative – The result is arithmetically less than zero, indicated by a ONE in bit position 1 of the Y Register. A ZERO in bit position ONE indicates "Not Negative".

d) Overflow – An overflow results from an arithmetic operation which causes exponent overflow, i.e., an exponent greater than $2^7 - 1$ (127) or less than -2^7 (-128).

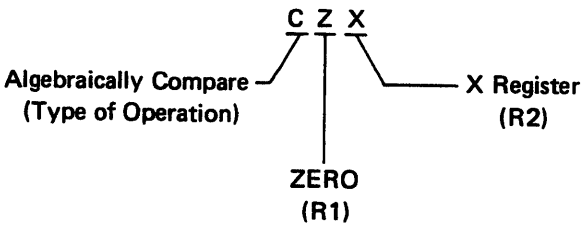
NOTE

If the SAU Overflow/Underflow executive trap is enabled, any instruction causing the overflow bit of the Y Register to be set will cause an interrupt.

Bits 1, 2 and 3 (Negative, Zero, Positive) of the Y Register are normally mutually exclusive. In certain instances it is desirable to know what operation caused an Overflow, e.g., a division by zero. The following operations cause more than two bits to be set in the Y Register:

a) Division by zero sets bits 0, 2, 3 ('15)
b) $\sqrt{-x}$ sets bits 0, 1, 2, 3 ('17)
c) Float to Fix, X>8388607 sets bits 0, 1, 3 ('13)

The algebraic compare instructions which are included in the SAU instruction set compare two referenced, signed (+ or -) quantities. The Y (condition) Register is set according to the result of the comparison. Algebraic comparisons are identified by the letter "C" as the first letter in the instruction mnemonic (e.g., CZX). The second letter in the mnemonic code identifies the first of the compared quantities (R1) and the remaining letter identifies the second quantity (R2). For example:



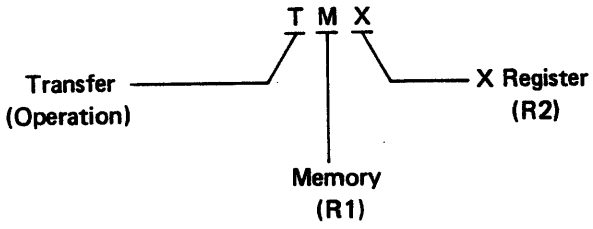
Comparisons are performed according to the following formula:

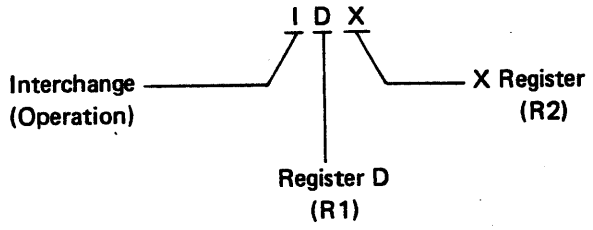
R2 – R1 = Y (Positive, Zero, or Negative)

Therefore, R2 > R1, R2 < R1, and R2 = R1, will set the condition (Y) register to Positive (+), Negative (-), and Zero (0), respectively.

Two instructions provide control of the SAU interrupt. These instructions either release or hold the interrupt.

The transfer instruction group includes various types of operations. Among these are transfers between memory and registers, registers and memory, and register-to-register. The transfer operation mnemonic code describes the individual operation. What operation is to be performed is described by the first letter in the mnemonic; "T" for transfer and "I" for interchange. The second and third letters of the mnemonic specify the source (R1) and destination (R2) of the transfer, respectively. Listed below are two examples:





With the exception of the interchange instruction, the transfer source (R1) is not altered as a result of the execution of a transfer instruction.

The following instructions are included in the SAU group.

ARITHMETIC

BRANCH

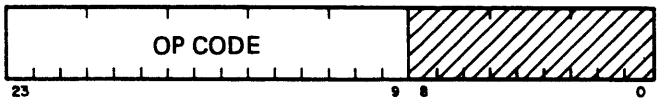
COMPARE

INTERRUPT

TRANSFER

AAX Add A Register to X Register

Formula 77070. **Affected** X,Y

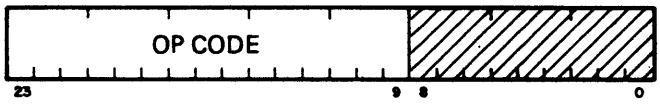


Operation

The signed integer in the A Register is converted to floating-point format and added to the number in the X Register. The sum replaces the previous contents of the X Register.

ADX Add D Register to X Register

Formula 77100. **Affected** X,Y

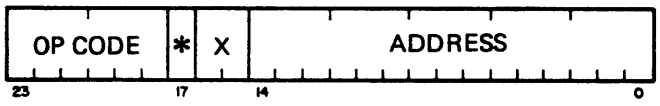


Operation

The floating-point number in the D Register is added to the number in the X Register. The sum replaces the previous contents of the X Register.

AMX Add Memory to X Register

Formula 73.*+X:a **Affected** X,Y

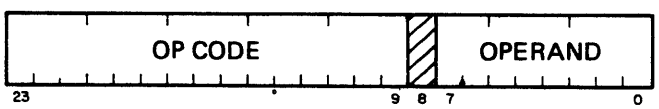


Operation

The contents of the effective memory address (EMA) and the next sequential address (EMA+1) are added to the contents of the X Register. The sum replaces the previous contents of the X Register.

AOW Add Operand to W Register (exponent)

Formula 77012:o **Affected** W,Y



Operation

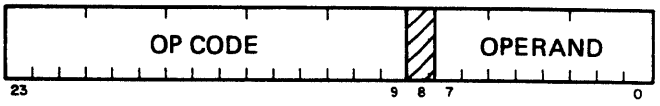
The 8-bit, signed operand is algebraically added to the contents of the W Register.

Note

A subtraction may be accomplished by adding a negative operand.

AOX Add Operand to X Register

Formula 77060:o **Affected** X,Y

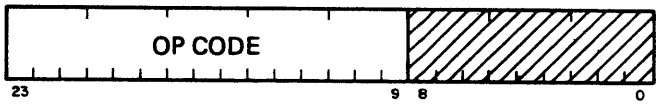


Operation

The signed, 8-bit integer operand is converted to floating-point format and added to the contents of the X Register. The sum replaces the previous contents of the X Register.

DAX Divide A Register (integer) into X Register

Formula 77073. **Affected** X,Y



Operation

The signed integer in the A Register is converted to floating point format. The contents of the X Register are divided by the converted number. The quotient replaces the previous contents of the X Register.

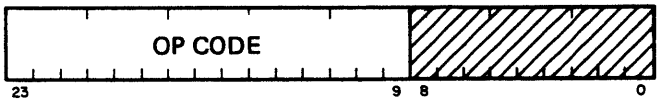
Notes

If division by zero occurs, the condition register (Y) is set to Overflow, Positive, and Zero, i.e., (Y) = '15.

In setting up to divide, the least significant bit of the mantissa is zeroed. The most obvious case is when X is divided by 1. If the least significant bit of the mantissa is 1, it will be 0 after the divide.

DDX Divide D Register (floating-point) into X Register

Formula 77103. **Affected** X,Y



Operation

The floating-point contents of the D Register are divided into the contents of the X Register. The quotient replaces the previous contents of the X Register.

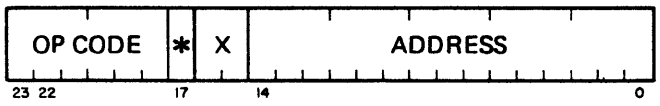
Notes

If division by zero occurs, the condition register (Y) is set to Overflow, Positive, and Zero, i.e., (Y) = '15.

In setting up to divide, the least significant bit of the mantissa is zeroed. The most obvious case is when X is divided by 1. If the least significant bit of the mantissa is 1, it will be 0 after the divide.

DMX Divide Memory into X Register.

Formula 76.*+X:a **Affected** X,Y



Operation

The contents of the X Register are divided by the contents of the effective memory address (EMA) and the next sequential address (EMA+1). The quotient replaces the previous contents of the X Register.

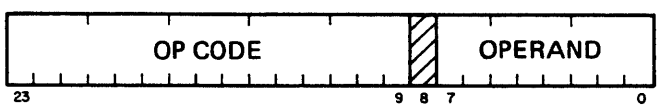
Notes

If division by zero occurs, the condition register (Y) will be set to Overflow, Positive, and Zero, i.e., (Y) = '15.

In setting up to divide, the least significant bit of the mantissa is zeroed. The most obvious case is when X is divided by 1. If the least significant bit of the mantissa is 1, it will be 0 after the divide.

DOX Divide Operand into X Register

Formula 77063:o **Affected** X,Y



Operation

The signed, 8-bit integer operand is converted to floating-point and is divided into the contents of the X Register. The quotient replaces the previous contents of the X Register.

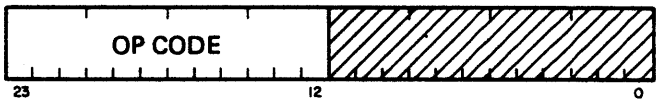
Notes

If division by zero occurs, the condition register (Y) will be set to Overflow, Positive, and Zero, i.e., (Y) = '15.

In setting up to divide, the least significant bit of the mantissa is zeroed. The most obvious case is when X is divided by 1. If the least significant bit of the mantissa is 1, it will be 0 after the divide.

FAX Floating Normalize of A Register to X Register

Formula 7703. **Affected** X,Y



Operation

The signed integer quantity in the A register is converted to a floating-point normalized quantity which replaces the previous quantity in the X Register.

Notes

A positive normalized number will have as the sign and most significant bit the following pattern:

01

A negative normalized number (where the value is not -1) has the configuration

10

A negative normalized number, where the value is -1, results in the mantissa having a bit pattern of all ONEs.

11

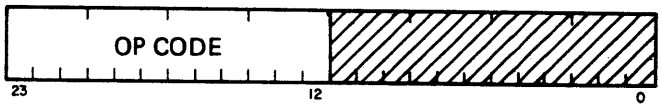
If the result is zero, the mantissa will be zero and the exponent will be set to a full scale negative value, i.e., (W) = '201.

The FAX instruction gives a different result than the FNO instruction for -2^N ($0 \leq N \leq 23_{10}$).

FNO of -2^N = 1.10...0 EXP = (N + 1)
FAX of -2^N = 1.00...0 EXP = N

FXA Fix of X Register to A Register

Formula 7713. **Affected** A,Y



Operation

The floating-point number in the X Register is converted to a 24-bit signed integer which replaces the previous contents of the A Register.

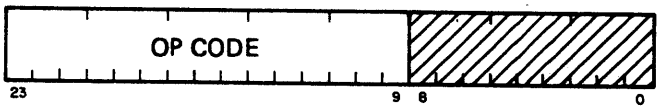
Notes

If the exponent is greater than 23, the condition register (Y) will be set to Overflow, Negative and Positive, i.e., (Y) = '13.

If the mantissa is negative, the result when truncated will be "rounded" toward the greater negative number.

INX Inverse of X Register

Formula 77050. **Affected** X,Y



Operation

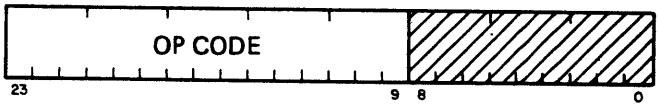
The inverse of the contents $\left[\frac{1}{(X)}\right]$ of the X Register replaces the contents of the X Register.

Note

If division by zero occurs, the condition register will be set to Overflow, Positive, and Zero, i.e., (Y) = '15.

MAX Multiply A Register (integer) and X Register

Formula 77072. **Affected** X,Y

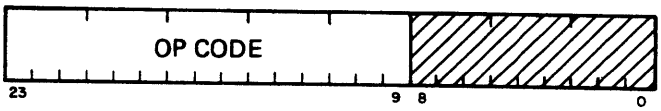


Operation

The signed integer in the A Register is converted to floating-point format and multiplied by the contents of the X Register. The product replaces the previous contents of the X Register.

MDX Multiply D Register (floating point) and X Register

Formula 77102. **Affected** X,Y



Operation

The floating-point contents of the D Register are multiplied by the contents of the X Register. The product replaces the previous contents of the X Register.

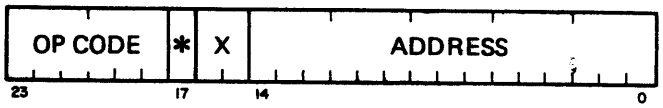
Notes

If the sum of the exponents is ≥200, Overflow will be generated. However, the final result may be corrected, i.e., (0.100...0E177) X (0.100...E001) = 0.100... E(177) + Overflow.

If both operands are MNG (1.00...0) and the sum of their exponents is 177, Overflow will be generated.

MMX Multiply Memory and X Register

Formula 75.*+X:a **Affected** X,Y

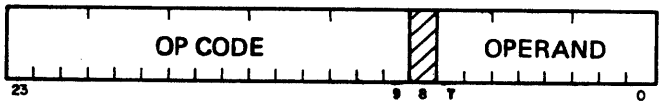


Operation

The contents of the X Register are multiplied by the contents of the effective memory address (EMA) and the next sequential address (EMA+1). The product replaces the previous contents of the X Register.

MOX Multiply Operand and X Register

Formula 77062:o **Affected** X,Y

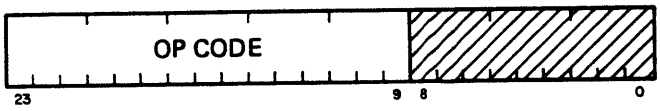


Operation

The signed, 8-bit integer operand is converted to floating-point format and is multiplied by the contents of the X Register. The floating-point product replaces the previous contents of the X Register.

NXX Negative of X Register to X Register

Formula 77041. **Affected** X,Y



Operation

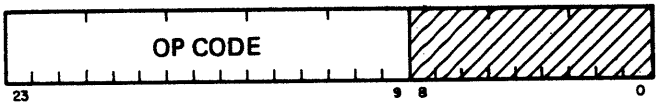
The mantissa in the X Register is two's complemented and the result is loaded into the X Register. The Y Register is changed to reflect the status of the new quantity.

Note

If the bit pattern of the mantissa is 100....0, the one's complement will be generated.

PXX Positive of X Register to X Register

Formula 77040. **Affected** X,Y



Operation

The absolute value of the contents of the X Register replaces the previous contents of the X Register.

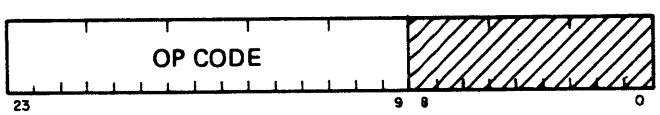
Notes

If the bit pattern of the mantissa is 100....0, the one's complement will be generated.

The operation noted above may cause a significant difference in a result, i.e., TNA (1), FAX, NXX, FXA generate A = 0; the result should have been 1. However, this may be alleviated by preceeding the NXX with an AOX (0) to normalize the X Register.

SAX Subtract A Register (integer) from X Register

Formula 77071. **Affected** X,Y

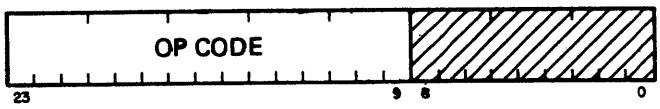


Operation

The signed integer in the A Register is converted to floating-point format and subtracted from the contents of the X Register. The difference replaces the previous contents of the X Register.

SDX Subtract D Register (floating point) from X Register

Formula 77101. **Affected** X,Y

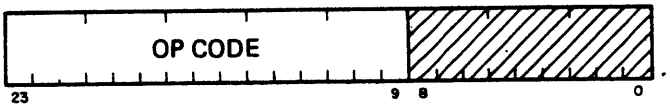


Operation

The floating-point contents of the D Register are subtracted from the X Register. The difference replaces the previous contents of the X Register.

SEX Square X Register

Formula 77051. **Affected** X,Y

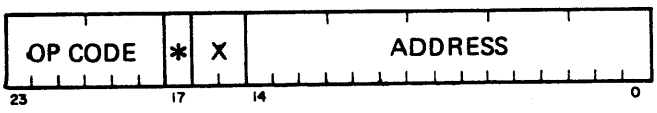


Operation

The square of the contents of the X Register replaces the previous contents of the X Register. (i.e., the X Register is replaced by X times X.)

SMX Subtract Memory from X Register

Formula 74.*+X:a **Affected** X,Y

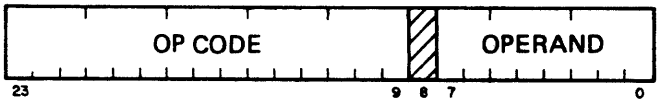


Operation

The contents of the effective memory address (EMA) and the next sequential address (EMA+1) are subtracted from the contents of the X Register. The difference replaces the contents of the X Register.

SOX Subtract Operand from X Register

Formula 77061:o **Affected** X,Y

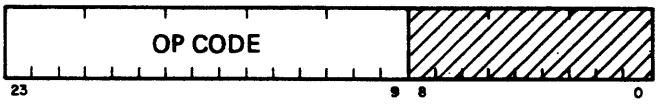


Operation

The signed, 8-bit integer operand is converted to a floating-point format and subtracted from the contents of the X Register. The difference replaces the previous contents of the X Register.

SRX Square Root of X Register

Formula 77052. **Affected** X,Y



Operation

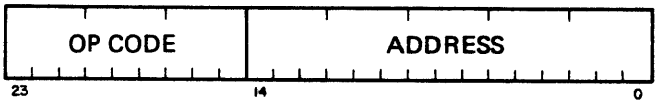
The square root of the contents of the X Register replaces the previous contents of the X Register.

Note

If the content of the X Register is negative, the condition register is set to Positive, Zero, Negative and Overflow, i.e., (Y) = '17.

BNR Branch on Negative Reset

Formula 630:a **Affected** P

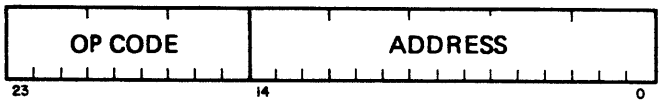


Operation

The contents of the condition (Y) register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the effective memory address. If the specified condition is not present, the program address advances to the next sequential location (program address +1).

BNS Branch on Negative Set

Formula 637:a **Affected** P

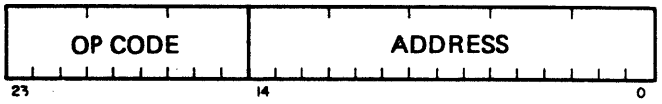


Operation

The contents of the condition (Y) register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the effective memory address. If the specified condition is not present, the program address advances to the next sequential location (program address +1).

BZR Branch on Zero Reset

Formula 640:a **Affected** P

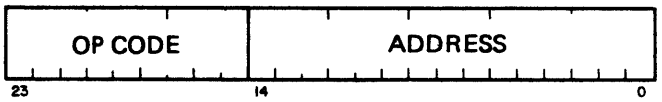


Operation

The contents of the condition (Y) register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the effective memory address. If the specified condition is not present, the program address advances to the next sequential location (program address +1).

BZS Branch on Zero Set

Formula 647:a **Affected** P

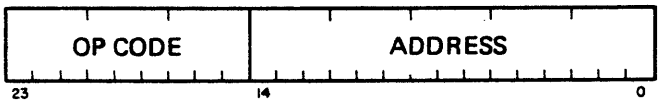


Operation

The contents of the condition (Y) register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the effective memory address. If the specified condition is not present, the program address advances to the next sequential location (program address +1).

BPR Branch on Positive Reset

Formula 650:a **Affected** P

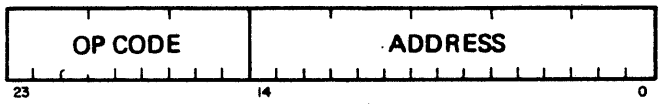


Operation

The contents of the condition (Y) register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the effective memory address. If the specified condition is not present, the program address advances to the next sequential location (program address +1).

BPS Branch on Positive Set

Formula 657:a **Affected** P

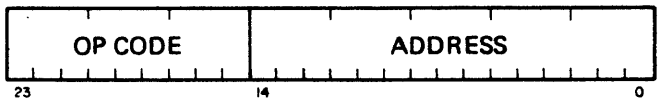


Operation

The contents of the condition (Y) register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the effective memory address. If the specified condition is not present, the program address advances to the next sequential location (program address +1).

BOR Branch on Overflow Reset

Formula 772:a **Affected** P

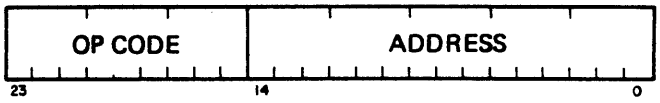


Operation

The contents of the condition (Y) register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the effective memory address. If the specified condition is not present, the program address advances to the next sequential location (program address +1).

BOS Branch on Overflow Set

Formula 773:a **Affected** P

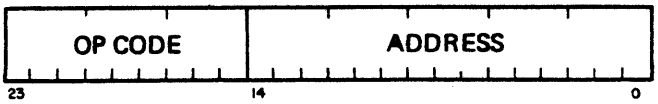


Operation

The contents of the condition (Y) register are tested for the specified condition. If the condition is present, the contents of the P Register (current program address) are replaced by the effective memory address. If the specified condition is not present, the program address advances to the next sequential location (program address +1).

BOX Branch on SAU Ready

Formula 627:a **Affected** P

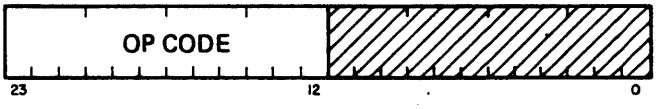


Operation

A determination is made as to whether or not the SAU is processing an instruction (the SAU busy latch is tested). If the SAU is able to process another instruction (i.e., ready) then the contents of the P Register (current program address) are replaced by the effective memory address. If the SAU is currently processing an instruction (i.e., not ready) the program address advances to the next sequential location (program address +1).

CDX Compare D Register to X Register

Formula 7712. **Affected** Y



Operation

The contents of the D Register and the contents of the X Register are compared and the Y (condition) Register is set to the status of the result.

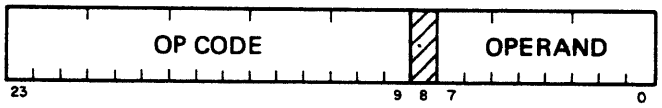
Note

Comparison results are as follows:

If X is greater than D; Y = Positive
If X is equal to D; Y = Zero
If X is less than D; Y = Negative

COW Compare Operand to W Register (exponent)

Formula 77013:o **Affected** Y



Operation

The 8-bit, signed operand and the contents of the W Register are algebraically compared and the Y (condition) Register is set to the status of the result.

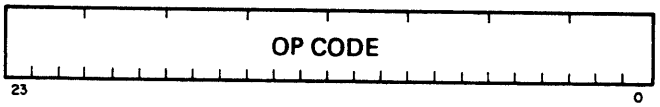
Note

Comparison results are as follows:

If W is greater than the operand; Y = Positive
If W is equal to the operand; Y = Zero
If W is less than the operand; Y = Negative

CZX Compare Zero to X Register

Formula 77060000 **Affected** Y,X



Operation

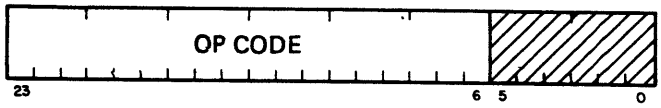
The contents of the X Register and floating-point zero are compared and the Y (condition) Register is set to the status of the result.

Note

Overflow will result if the mantissa has the pattern 1100...0 and the exponent has the pattern 10000000. The least significant bit of X will be set to a 1.

HSI Hold SAU Overflow Interrupt

Formula 770200. **Affected** None

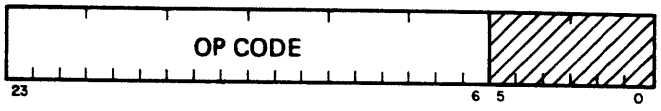


Operation

This instruction disarms the overflow/underflow interrupt (Executive trap Group 0, Level 6). The trap remains disarmed until the execution of the release instruction.

RSI Release SAU Overflow Interrupt

Formula 770201. **Affected** None

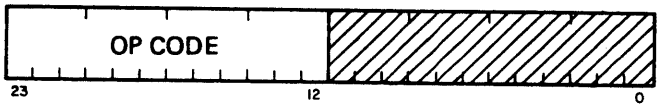


Operation

This instruction arms the overflow/underflow interrupt (Executive Trap Group 0, Level 6). When the trap is armed, and not inhibited by an HXI instruction, any SAU operation which causes bit 0 of the Y Register to be set (Overflow) will generate an interrupt request.

IDX Interchange D Register and X Register

Formula 7711. **Affected** D,X,Y



Operation

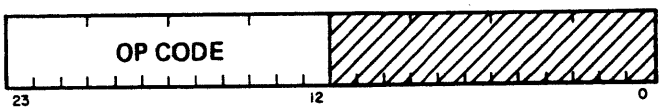
The contents of the X Register and the D Register are interchanged. The Y (condition) Register is set to the status of the X Register on completion of the instruction.

Note

The SAU uses the two most significant bits of the mantissa and the sign of the exponent to set the Y Register.

TDX Transfer D Register to X Register

Formula 7714. **Affected** X,Y



Operation

The contents of the D Register replace the previous contents of the X Register.

Notes

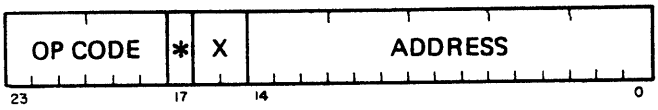
An unnormalized number transferred to X may not set the Y Register properly.

The SAU uses the two most significant bits of the mantissa and the sign of the exponent to set the Y Register.

A binary zero transferred to X will set Positive.

TMX Transfer Memory to X Register

Formula 71.*+X:a **Affected** X,Y



Operation

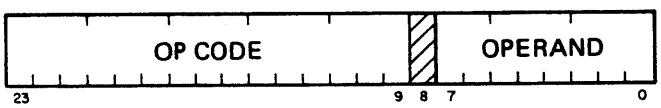
The contents of the effective memory address (EMA) and the next sequential address (EMA+1) replace the previous contents of the X Register. EMA and EMA+1 replace the most significant and least significant part of X, respectively.

Note

The SAU uses the two most significant bits of the mantissa and the sign of the exponent to set the Y Register.

TOW Transfer Operand to W Register (exponent)

Formula 77011:o **Affected** W,Y



Operation

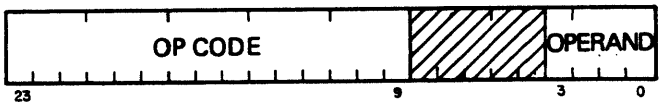
The 8-bit, signed operand replaces the previous contents of the W Register. All other bits within the X Register are unaffected.

Note

The Y (condition) Register is set to the status of the X and XW Registers upon completion of the instruction. The SAU uses the two most significant bits of the mantissa and the sign of the exponent to set the Y Register.

TOY Transfer Operand to Y Register

Formula 77010:o **Affected** Y



Operation

The four bit operand replaces the previous contents of the Y (condition) Register.

Note

Operand definition is as follows:

Bit 0 = ONE = Overflow
= ZERO = No Overflow

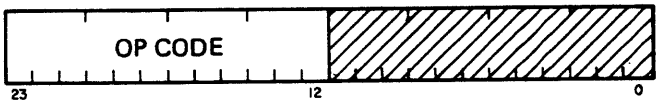
Bit 1 = ONE = Negative
= ZERO = Not Negative

Bit 2 = ONE = Zero
= ZERO = Not Zero

Bit 3 = ONE = Positive
= ZERO = Not Positive

TXD Transfer X Register to Memory

Formula 7715. **Affected** D

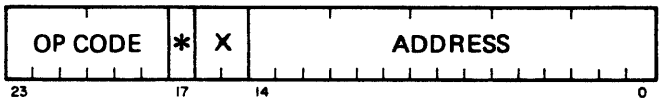


Operation

The contents of the X Register replaces the previous contents of the D Register. The X Register is unchanged.

TXM Transfer X Register to Memory

Formula 72.*+X:a **Affected** M



Operation

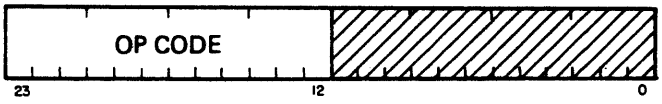
The contents of the X Register replaces the previous contents of the effective memory address (EMA) and the next sequential address (EMA+1). The most and least significant portions of X are transferred to EMA and EMA+1, respectively.

Note

The SAU uses the two most significant bits of the mantissa and the sign of the exponent to set the Y Register.

TYA Transfer Y Register to A Register

Formula 7700. **Affected** A



Operation

The contents of the Y Register are transferred to the A Register and the status of the SAU overflow/underflow interrupt is placed in bit position 6 in the A Register.

Note

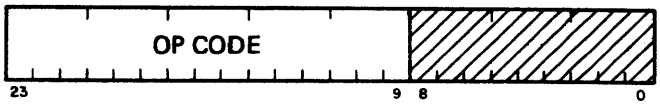
The following table shows the bit placements of the various Y (condition) Register settings when transferred to the A Register.

A Register	Bit Function
Bit 0 = 1	Overflow/Underflow
Bit 1 = 1	Negative
Bit 2 = 1	Zero
Bit 3 = 1	Positive
Bit 6 = 0	SAU Interrupt Enabled
Bit 6 = 1	SAU Interrupt Disabled

All other bits within the A Register are set to zero.

TZX Transfer Zero to X Register

Formula 77042. **Affected** X



Operation

The floating-point representation of zero (0000000000000201) replaces the previous contents of the X Register. The Y (condition) Register is unaffected.

APPENDIX A
INSTRUCTION EXECUTION TIMES

COMPUTING INSTRUCTION EXECUTION TIMES

This appendix provides the formulas for computing the execution times (in microseconds) of the computer instructions. The time required to execute any particular instruction is not constant, but is dependent upon certain variables.

Instruction execution time is primarily a function of the program. The time required to execute a particular instruction is dependent on its location within the program. Other factors affecting instruction execution time include type of memory board used, memory access time, address translation time, indexed and indirected operations, and system configurations. When a Scientific Arithmetic Unit or I/O expansion chassis is included in the system, instruction execution time is affected.

Each time that memory is accessed for a read operation, two words are read out and loaded into a two-word data register located on the memory board. This register is referred to as the Content Addressable Buffer (CAB). When a memory read operation is performed, if the addressed word is in the CAB, no memory access is required. If the addressed word is not in the CAB, then a memory access is performed to read out the addressed word. Thus, access time is not constant. A memory access made to read out a word is referred to as a normal access. If the desired word is located in the CAB, the operation is referred to as a fast access.

Since instructions are overlapped by one microcycle during execution, the decode microcycle time is omitted when calculating instruction execution time. Execution time is based primarily on the number of microinstructions executed where each microinstruction is executed in one 0.3 microsecond microcycle. Added to this time is the wait time, A, which is the time period from the end of one instruction decode sequence to the beginning of the subsequent decode sequence. For a memory read operation, wait time is dependent on whether the desired word is in memory or the CAB. A memory wait time is also incurred following a write operation if an access is made to the same memory board. This wait time is designated by the letter W. The wait periods, A and W, for the various memory configurations and conditions are provided in Table A-1.

Instruction execution time may be calculated by using the following basic formula.

Instruction Time =

A + W + (0.3 X Number of Execution Cycles)

More than one A or W time may be involved. If a memory reference instruction, such as a transfer memory to register instruction, is executed, two memory accesses (A time) are made; one to fetch the instruction and one to fetch the operand. When the Transfer Registers to Memory (TRM) instruction is executed, several successive memory write operations are performed so that W time is the sum of the delays incurred while waiting for memory to finish the required number of cycles.

The basic instruction time formula is modified by particular conditions and system configurations. The SAU instructions, for example, in addition to the A and W wait times, incur a delay referred to as the resynchronizing wait time, RS. Since the CPU, SAU, and SAU interface boards contain individual clocks, the clock pulses must be synchronized when executing SAU instructions. RS varies from a minimum time of 0.18, to a maximum of 0.48 microsecond.

When the virtual memory system is in the User Mode of Operation, all addresses are translated. An additional 0.12 microsecond is added to A or W for memory read and write operations. This delay is a result of the address translation performed by the virtual memory hardware.

For memory reference instructions, an additional 0.33 microsecond is added for an index operation. For each indirect reference operation, additional time equal to A plus 0.33 microsecond is added to the basic instruction execution time.

When a system has the input/output expansion chassis installed, execution of any of the I/O instructions results in a delay of 0.06 microsecond. This delay is the result of increased signal propagation times caused by the addition of the I/O port board, I/O interface board, and interconnecting cables.

Table A-1. Instruction Execution Wait Times*

FUNCTION		MOS	CORE
A	(normal access – word in main memory)	0.3	0.245
	(fast access – word in CAB)**	0 or 0.03	0 or 0.03
W	(access to memory board written to on previous cycle)	0.15	0.20
	(access to memory board not written to on previous cycle)	0 or 0.03	0 or 0.03
RS		0.18 to 0.48	0.18 to 0.48
VM Address Translation		0.12	0.12
Indexed		0.33	0.33
Indirected (each)		A + 0.33	A + 0.33
Expanded System (I/O instructions only)		0.06	0.06
Error Correct Cycle		0.75	NA
Refresh Cycle		0.15 or 0.45	NA
Block I/O Channel Memory Operation	(read)	0.3	0.3
	(write)	0.45	0.5

* **Time is in microseconds**

** **Time is dependent on memory configuration**

NA **Not Applicable**

If the CPU initiates a memory read cycle to MOS memory and a parity error is recognized, the memory performs an error correction cycle if configured with error correct hardware. This operation increases instruction execution time by about 0.75 microsecond.

MOS memory boards perform a refresh cycle approximately every 32 microseconds. If a refresh cycle occurs during the execution of an instruction, additional time of either 0.15 or 0.45 microseconds is added to the instruction execution time.

When a block I/O channel performs a DMA operation, the execution time is increased by 0.3 microsecond if the channel performs a read operation. A write operation performed by the channel adds 0.45 microsecond for a MOS memory board, and 0.5 microsecond for a core memory board.

System performance is upgraded when the memory boards are interleaved. Interleaving reduces A time so that the instruction execution time is reduced. The time that any particular instruction is affected is program dependent.

Table A-2 provides the basic time formulas for all instructions.

Table A-2. Basic Instruction Time Formulas

Mnemonic	Formula	Notes	Mnemonic	Formula	Notes
AAM	2A + W + 0.9		BUL	A + 0.6	
AAX	A + 1.8 + RS	1,2	BWx	A + 0.9	
ADX	A + 1.8 + RS	1,2	BZR	A + 0.6	1
AEM	2A + W + 0.9		BZS	A + 0.6	1
AMA	2A + 0.6		CDX	A + 2.1 + RS	1,2
AMB	2A + 0.6		CMA	2A + 0.6	
AMD	3A + 2.4		CMB	2A + 1.8	
AME	2A + 0.6		CME	2A + 0.6	
AMx	2A + 0.6		CMx	2A + 0.9	
AMX	3A + 2.1 + RS	1,3	COB	A + 1.8	
AOB	A + 0.3		COW	A + 1.0 + RS	1,6
AOM	3A + 2.4	4	Crr	A + 1.5	7
	3A + 2.7	5		$A + 3.0 + 0.3N_S + 0.3N_D$	9,10,15
AOr	A + 0.3		CZD	A + 3.9	
AOW	A + 1.0 + RS	1,6	CZM	2A + 0.6	
AOX	A + 1.0 + RS	1,6	CZr	A + 1.5	7
Arr	A + 0.3	7		$A + 3.3 + 0.3N_D$	9,10,16
	$A + 3.0 + 0.3N_S + 0.6N_D$	8,9,10	CZX	A + 1.6 + RS	1,17
AUM	2A + 0.9 + W		DAX	A + 9.6 + RS	1,18
AxM	2A + 0.9 + W		DDX	A + 9.9 + RS	1,18
BBI	A + 1.2	11	DMA	2A + 0.6	
	A + 1.8	12	DMH	2A + 3.0	
	A + 2.1	13	DMX	3A + 10.8 + RS	1,19
BBJ	A + 1.2	11	DNH	2A + 3.0	
	A + 1.8	12	DOB	A + 0.3	
	A + 2.1	13	DOX	A + 9.4 + RS	1,20
BJL	A + 0.6		Drr	A + 0.3	7
BLL	A + 0.9			$A + 2.7 + 0.3N_S + 0.6N_D$	9,10,15
BLU	A + 0.9		DVM	2A + 12.3	21,23,24,25
BLx	A + 0.9			2A + 12.6	22,23,24,25
BNR	A + 0.6	1	DVO	A + 12.3	21,23,24,25
BNS	A + 0.6	1		A + 12.6	22,23,24,25
BOc	A + 0.6		DVT	A + 12.3	21,23,24,25
BOR	A + 0.6	1		A + 12.6	22,23,24,25
BOS	A + 0.6	1	DVx	A + 12.3	21,23,24,25
BOX	A + 0.6	1		A + 12.6	22,23,24,25
BPR	A + 0.6	1	DV2	A + 1.5 + 2.1W	
BPS	A + 0.6	1	EMB	2A + 4.8	26
BRL	A + 0.9			2A + 2.4	27
BSL	A + 1.2 + W		ESA	A + 0.9	28
BUC	A + 0.6			A + 1.5	29
				A + 1.8	30

Table A-2. Basic Instruction Time Formulas (Cont'd.)

Mnemonic	Formula	Notes	Mnemonic	Formula	Notes
ESB	A + 0.6		LRD	A + 1.5 + 0.15n	34,40
EXM	A + 0.6			A + 1.2 + 0.15n	34,39
EZB	A + 0.3		MAX	A + 5.4 + RS	1,41
FAX	A + 1.2 + RS	1,14	MDX	A + 5.7 + RS	1,41
FBM	2A + 3.0 + W		MMX	3A + 5.6 + RS	1,42
FNO	A + 2.4	31	MOX	A + 5.2 + RS	1,43
	A + 3.6 + 0.3n	32,34	MYM	2A + 5.7	44
	A + 4.2 + 0.3n	33,34		2A + 6.0	45
FXA	A + 1.2		MYO	A + 5.7	44
GAP	A + 1.2			A + 6.0	45
HIT	A + 0.3		MYr	A + 5.7	44
HLT	A + 0.6			A + 6.0	45
HSI	A + 0.9		NBB	A + 0.3	
HTx	A + 0.9 + W		NDD	A + 1.8	39
HXI	A + 0.3			A + 2.4	38
IAW	A + 0.3	35		A + 2.1	37
	A + 0.36	36	NHH	A + 0.9	
IDW	A + 0.3	35	NOP	A + 0.3	
	A + 0.36	36	NSr	A + 0.3	7
IDX	A + 1.5		Nrr	A + 0.3	7
IMA	2A + 1.5 + W			$A + 3.0 + 0.3N_S + 0.6N_D$	8,9,10
IME	2A + 1.5 + W		NXX	A + 1.0 + RS	1,6
IMx	2A + 1.5 + W		OAW	A + 0.3	35
INX	A + 9.4 + RS	1,20		A + 0.36	36
IPW	A + 0.3	35	OCW	A + 0.3	35
	A + 0.36	36		A + 0.36	36
Irr	A + 1.5	7	ODW	A + 0.3	35
ISW	A + 0.3	35		A + 0.36	36
	A + 0.36	36	OMA	2A + 0.6	
KOB	A + 0.9		OMH	2A + 3.0	
Krr	A + 1.2	7	ONH	2A + 3.0	
	$A + 2.7 + 0.3N_S + 0.3N_D$	9,10,15	OOB	A + 0.3	
LAA	A + 0.6 + 0.15n	34	Orr	$A + 2.7 + 0.3N_S + 0.3N_D$	9,10
LAD	A + 3.6 + 0.15n	34,37	PBB	A + 0.6	
	A + 3.9 + 0.15n	34,38	PDD	A + 1.5	46
	A + 3.3 + 0.15n	34,39		A + 2.1	47
LLA	A + 0.6 + 0.15n	34		A + 1.8	48
LLD	A + 1.5 + 0.15n	34,40		A + 2.7	49
	A + 1.2 + 0.15n	34,39		A + 2.4	50
LRA	A + 0.6 + 0.15n	34	Prr	A + 0.9	7
				$A + 3.0 + 0.3N_S + 0.6N_D$	8,9,10

Table A-2. Basic Instruction Time Formulas (Cont'd.)

Mnemonic	Formula	Notes	Mnemonic	Formula	Notes
PXX	A + 1.0 + RS	1,6	SRE	A + 66.0	
QBB	A + 0.3		Srr	A + 0.3	7
QBH	A + 0.6			A + 3.0 + 0.3N_S + 0.6N_D	8,9,10
QBM	2A + 3.0		SRT	A + 35.0	
QNR	A + 0.3		SRX	A + 8.2 + RS	1,53
QSS	A + 0.9		TAM	A + 0.6 + W	
QUR	A + 0.3		TAR	A + 0.6	
RAA	A + 0.6 + 0.15n	34	TBM	A + 0.6 + W	
RAD	A + 2.1 + 0.15n	34,40	TDL	A + 1.2	
	A + 1.8 + 0.15n	34,39	TDM	A + 1.8 + W	
RBM	2A + 4.8 + W	51	TDP	A + 0.6	
	2A + 2.4 + W	52	TDR	A + 0.6	
RCT	A + 0.3		TDS	A + 0.6	
RLA	A + 0.6 + 0.15n	34	TDX	A + 0.6	1
RLD	A + 1.5 + 0.15n	34,40	TD1	A + 1.2	
	A + 1.2 + 0.15n	34,39	TD4	A + 1.2	
ROM	A + 0.6		TEM	A + 0.6 + W	
RPT	A + 0.3		TEU	A + 0.6	
RRA	A + 0.6 + 0.15n	34	TFH	A + 0.6	
RRD	A + 1.2 + 0.15n	34,39	TFM	A + 0.9 + W	
	A + 1.5 + 0.15n	34,40	THM	2A + 3.0 + W	
Rrr	A + 0.9	7	TIM	A + 0.6 + W	
	A + 3.0 + 0.3N_S + 0.6N_D	8,9,10	TJM	A + 0.6 + W	
RSI	A + 0.9	1	TKM	A + 0.6 + W	
RUM	A + 0.6		TKV	A + 1.2	
RXI	A + 0.3		TLD	A + 1.2	
SAX	A + 1.8 + RS	1,2	TLO	A + 0.3	
SDX	A + 2.1 + RS	1,2	TMA	2A + 0.6	
SEX	A + 5.2 + RS	1,43	TMB	2A + 0.6	
SMA	2A + 0.6		TMD	3A + 1.5	39
SMB	2A + 0.6			3A + 2.1	38
SMD	3A + 2.4	39		3A + 1.8	37
	3A + 3.0	38	TME	2A + 0.6	
	3A + 2.7	37	TMH	2A + 3.0	
SME	2A + 0.6		TMI	2A + 0.6	
SMx	2A + 0.6		TMJ	2A + 0.6	
SMX	3A + 2.1 + RS	1,3	TMK	2A + 0.6	
SOB	A + 0.3		TMQ	2A + 0.6	
SOr	A + 0.3		TMR	6A + 3.0	
			TMX	3A + 0.9	
SOX	A + 1.6 + RS	1,17	TNr	A + 0.3	

Table A-2. Basic Instruction Time Formulas (Cont'd.)

Mnemonic	Formula	Notes	Mnemonic	Formula	Notes
TOB	A + 0.3		TZH	A + 0.9	
TOC	A + 0.3		TZM	A + 0.9 + W	
TOr	A + 0.3		TZr	A + 0.3	56
TOW	A + 1.0 + RS	1,6		A + 3.0 + 0.3N_D	10,57
TOY	A + 0.9	1	TZX	A + 0.6	1
TPD	A + 0.6		T1D	A + 1.2	
TrB	A + 0.3	54	T4D	A + 1.2	
	A + 2.4 + 0.3N_S	9,55	UA1	A + 1.2	
TRD	A + 1.2		UD1	A + 1.2	
TRM	A + 3.0 + W		UE1	A + 1.2	
Trr	A + 0.3	7	UI1	A + 1.2	
	A + 2.7 + 0.3N_S + 0.6N_D	9,10,15	USP	3A + 3.3	
TSD	A + 0.6		XMA	2A + 0.6	
TSr	A + 0.9		XMH	2A + 3.0	
TUD	A + 0.6		XNH	2A + 3.0	
TVK	A + 1.2		XOB	A + 0.3	
TXD	A + 0.6	1	Xrr	A + 0.3	7
TXM	A + 2W + 1.2	1		A + 2.7 + 0.3N_S + 0.6N_D	9,10,15
TYA	A + 0.6	1	ZBM	2A + 3.0 + W	

NOTES:

1. Applicable only to model 615A SAU
2. Concurrent time available = 1.2 + RS
3. Concurrent time available = 0.9 + RS
4. If operand is positive
5. If operand is negative
6. Concurrent time available = 0.4 + RS
7. If single source and destination
8. If multiple source and destination
9. N_S = number of registers selected in group r1
10. N_D = number of registers selected in group r2
11. If bits 23 and 22 of index register are set to 01_2 or 10_2 prior to instruction execution
12. If bits 23 and 22 of index register are set to 11_2 prior to instruction execution (branch not taken)
13. If bits 23 and 22 of index register are set to 11_2 prior to instruction execution (branch taken)
14. Concurrent time available = 0.6 + RS
15. If multiple source and destination or T
16. If multiple destination or T is destination
17. Concurrent time available = 1.0 + RS
18. Concurrent time available = 9.0 + RS
19. Concurrent time available = 9.6 + RS
20. Concurrent time available = 8.8 + RS
21. If signs of dividend and divisor are not equal
22. If signs of dividend and divisor are equal

NOTES:

23. Add 0.6 if correction cycle is required
24. Add 0.3 if overflow occurs
25. Add 0.9 if divisor is equal to zero
26. If bits 23 and 22 of J are set to 01_2 or 10_2
27. If bits 23 and 22 of J are set to 11_2 or 00_2
28. If sign bit is negative
29. If sign bit is positive
30. If sign bit is zero
31. If D is equal to zero or $4000...0_8$
32. If normalization takes place
33. If normalization takes place and result is equal to $4000...0_8$
34. n = number of shifts
35. If non-expanded system
36. If expanded system
37. If result in D is equal to zero
38. If result in E is equal to zero
39. If result in E is not equal to zero
40. If result in D or E is equal to zero
41. Concurrent time available = 4.8 + RS
42. Concurrent time available = 4.7 + RS
43. Concurrent time available = 4.6 + RS
44. If multiplicand is positive
45. If multiplicand is negative
46. If D initial value is positive and result in E is not equal to zero
47. If D initial value is positive and result in E is equal to zero, or if D initial value is negative and result in E is not equal to zero.
48. If D initial value is positive and result in D is equal to zero
49. If D initial value is negative and result in E is equal to zero
50. If D initial value is negative and result in D is equal to zero
51. If bits 23 and 22 of I are set to 01_2 or 10_2
52. If bits 23 and 22 of I are set to 11_2 or 00_2
53. Concurrent time available = 7.6 + RS
54. If single source
55. If multiple source or T
56. If single destination
57. If multiple destination

APPENDIX B
INSTRUCTION INDEX

Mnemonic	Instruction	Page
CDX	Compare D Register to X Register	7-68
CMA	Compare Memory and A	7-19
CMB	Compare Memory and Byte	7-20, 7-38
CME	Compare Memory and E	7-20
CMx	Compare Memory and Register	7-19
COB	Compare Operand and Byte	7-20, 7-38
COW	Compare Operand to W Register	7-69
Crr	Compare Register and Register	7-21
CZD	Compare Zero and Double	7-21
CZM	Compare Zero and Memory	7-20
CZr	Compare Zero and Register	7-20
CZX	Compare Zero to X Register	7-69
DAX	Divide A Register into X Register	7-63
DDX	Divide D Register into X Register	7-64
DMA	Dot Memory with A	7-22
DMH	Dot Memory with H	7-48
DMX	Divide Memory into X Register	7-64
DNH	Dot Not (memory) with H	7-48
DOB	Dot Operand with Byte	7-22, 7-39
DOX	Divide Operand into X Register	7-64
Drr	Dot Register with Register	7-23
DVM	Divide by Memory	7-6
DVO	Divide by Operand	7-6
DVT	Divide by T	7-7
DVx	Divide by Register	7-7
DV2	Divide by 2	7-7
EMB	Extract Memory Byte	7-28, 7-39
ESA	Extend Sign of A	7-8
ESB	Extend Sign of Byte	7-8, 7-39
EXM	Execute Memory	7-59
EZB	Extend Zeros from Byte	7-39, 7-60
FAX	Floating Normalize of A Register to X Register	7-64
FBM	Flag Bit of Memory	7-49
FNO	Floating Normalize	7-8
FXA	Fix of X Register to A Register	7-65
GAP	Generate Argument Pointer	7-58
HIT	Hold Interval Timer	7-60
HLT	Halt	7-58
HSI	Hold SAU Overflow Interrupt	7-69
HTx	Hold Interrupts and Transfer Register to Memory	7-55
HXI	Hold External Interrupts	7-55

Mnemonic	Instruction	Page
IAW	Input Address Word	7-46
IDW	Input Data Word	7-45
IDX	Interchange D Register and X Register	7-69
IMA	Interchange Memory and A	7-29
IME	Interchange Memory and E	7-29
IMx	Interchange Memory and Register	7-29
INX	Inverse of X Register	7-65
IPW	Input Parameter Word	7-46
Irr	Interchange Register and Register	7-29
ISW	Input Status Word	7-44
KOB	Kompare Operand and Byte	7-21, 7-40
Krr	Kompare Register and Register	7-21
LAA	Left Shift Arithmetic A	7-25
LAD	Left Shift Arithmetic Double	7-25
LLA	Left Shift Logical A	7-25
LLD	Left Shift Logical Double	7-25
LRA	Left Rotate A	7-26
LRD	Left Rotate Double	7-26
MAX	Multiply A Register and X Register	7-65
MDX	Multiply D Register and X Register	7-65
MMX	Multiply Memory and X Register	7-65
MOX	Multiply Operand and X Register	7-65
MYM	Multiply by Memory	7-8
MYO	Multiply by Operand	7-8
MYr	Multiply by Register	7-9
NBB	Negate of Byte to Byte	7-9, 7-40
NDD	Negate of Double to Double	7-10
NHH	Negate of H to H	7-48
NOP	No Operation	7-58
NSr	Negate Sign of Register	7-10
Nrr	Negate of Register to Register	7-9
NXX	Negative of X Register to X Register	7-66
OAW	Output Address Word	7-45
OCW	Output Command Word	7-43
ODW	Output Data Word	7-44
OMA	OR Memory with A	7-23
OMH	OR Memory with H	7-48
ONH	OR Not (memory) with H	7-47
OOB	OR Operand with Byte	7-23, 7-40
Orr	OR Register with Register	7-23

Mnemonic	Instruction	Page
PBB	Positive of Byte to Byte	7-10, 7-40
PDD	Positive of Double to Double	7-10
Prr	Positive of Register to Register	7-11
PXX	Positive of X Register to X Register	7-66
QBB	Query Bits of Byte	7-41, 7-59
QBH	Query Bit of H	7-47
QBM	Query Bit of Memory	7-49
QNR	Query Not-modified Register	7-52
QSS	Query Sense Switches	7-60
QUR	Query Virtual Usage Register	7-52
RAA	Right Shift Arithmetic A	7-26
RAD	Right Shift Arithmetic Double	7-26
RBM	Replace Byte in Memory	7-30, 7-40
RCT	Release Clock Time	7-60
RLA	Right Shift Logical A	7-26
RLD	Right Shift Logical Double	7-27
ROM	Release Operand Mode	7-52
RPT	Release Processor Time	7-60
RRA	Right Rotate A	7-27
RRD	Right Rotate Double	7-27
Rrr	Round of Register to Register	7-11
RSI	Release SAU Overflow Interrupt	7-69
RUM	Release User Mode	7-53
RXI	Release External Interrupts	7-55
SAX	Subtract A Register from X Register	7-66
SDX	Subtract D Register from X Register	7-66
SEX	Square X Register	7-66
SMA	Subtract Memory from A	7-12
SMB	Subtract Memory from Byte	7-12
SMD	Subtract Memory from Double	7-12
SME	Subtract Memory from E	7-12
SMx	Subtract Memory from Register	7-11
SMX	Subtract Memory from X Register	7-66
SOB	Subtract Operand from Byte	7-13, 7-41
SOr	Subtract Operand from Register	7-13
SOX	Subtract Operand from X Register	7-67
SRE	Square Root Extended	7-14
Srr	Subtract Register from Register	7-13
SRT	Square Root	7-13
SRX	Square Root of X Register	7-67

Mnemonic	Instruction	Page
TAM	Transfer A to Memory	7-35
TAR	Transfer A to 1 Virtual Address Register	7-50
TBM	Transfer Byte to Memory	7-34, 7-41
TDL	Transfer Double to Limit Registers	7-53
TDM	Transfer Double to Memory	7-35
TDP	Transfer Double to Paging Limit Registers	7-51
TDR	Transfer Double to 2 Virtual Address Registers	7-51
TDS	Transfer Double to Source and Destination Registers	7-50
TDX	Transfer D Register to X Register	7-70
TD1	Transfer Double to Group 1	7-56
TD4	Transfer Double to Group 1	7-56
TEM	Transfer E to Memory	7-35
TEU	Transfer E to Usage Base Register	7-52
TFH	Transfer Flag to H	7-47
TFM	Transfer Flag to Memory	7-35
THM	Transfer H to Memory	7-49
TIM	Transfer I to Memory	7-35
TJM	Transfer J to Memory	7-36
TKM	Transfer K to Memory	7-36
TKV	Transfer K to V	7-47
TLD	Transfer Limit Registers to Double	7-53
TLO	Transfer Long Operand to K	7-33
TMA	Transfer Memory to A	7-31
TMB	Transfer Memory to Byte	7-30, 7-42
TMD	Transfer Memory to Double	7-30
TME	Transfer Memory to E	7-31
TMH	Transfer Memory to H	7-49
TMI	Transfer Memory to I	7-31
TMJ	Transfer Memory to J	7-32
TMK	Transfer Memory to K	7-32
TMQ	Transfer Memory to Query Register	7-31
TMR	Transfer Memory to Registers	7-32
TMX	Transfer Memory to X Register	7-70
TNr	Transfer Negative Operand to Register	7-32
TOB	Transfer Operand to Byte	7-32, 7-42
TOC	Transfer Operand to Condition Register	7-33
TOr	Transfer Operand to Register	7-33
TOW	Transfer Operand to W Register	7-70
TOY	Transfer Operand to Y Register	7-70
TPD	Transfer Paging Limit Registers to Double	7-51
TrB	Transfer Register to Byte	7-34, 7-41
TRD	Transfer 2 Virtual Address Registers to Double	7-51
TRM	Transfer Register to Memory	7-36
Trr	Transfer Register to Register	7-36
TSD	Transfer Source and Destination Registers to Double	7-50

Mnemonic	Instruction	Page
TSr	Transfer Switches to Register	7-33
TUD	Transfer Usage Base Register and Demand Page Register to Double	7-51
TVK	Transfer V to K	7-47
TXD	Transfer X Register to D Register	7-70
TXM	Transfer X Register to Memory	7-71
TYA	Transfer Y Register to A Register	7-71
TZH	Transfer Zero to H	7-47
TZM	Transfer Zero to Memory	7-35
TZr	Transfer Zero to Register	7-34
TZX	Transfer Zero to X Register	7-71
T1D	Transfer Group 1 to Double	7-56
T4D	Transfer Group 1 to Double	7-56
UA1	Unitarily Arm Group 1 Interrupts	7-56
UD1	Unitarily Disarm Group 1 Interrupts	7-57
UE1	Unitarily Enable Group 1 Interrupts	7-57
UI1	Unitarily Inhibit Group 1 Interrupts	7-57
USP	Update Stack Pointer	7-59
XMA	Exclusive-OR Memory with A	7-24
XMH	Exclusive-OR Memory with H	7-49
XNH	Exclusive-OR Not (memory) with H	7-49
XOB	Exclusive-OR Operand with Byte	7-24, 7-42
Xrr	Exclusive-OR Register with Register	7-24
ZBM	Zero Bit of Memory	7-50