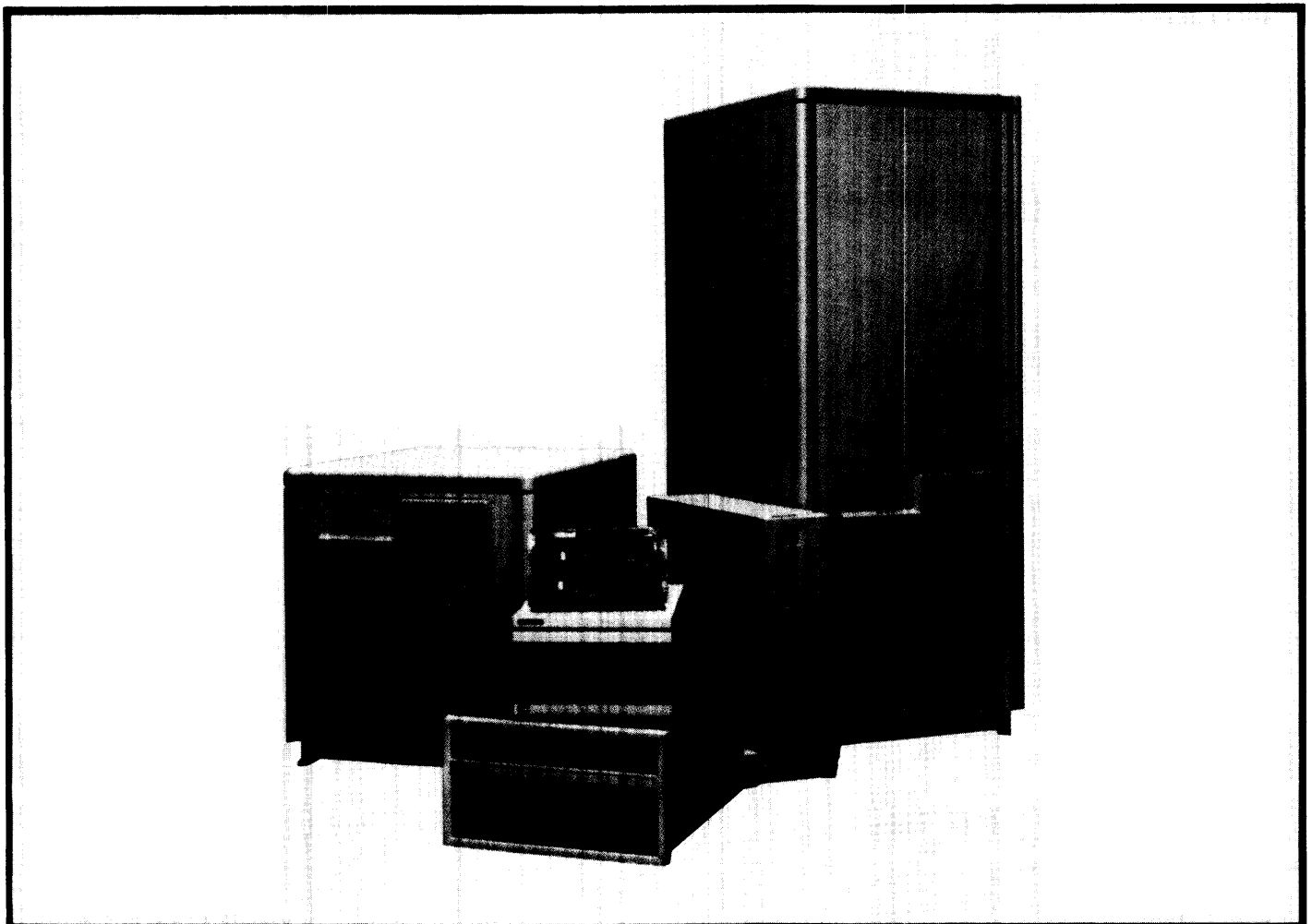# HP 1000 A400 Computer

Reference Manual

HP1000 A-Series

HP 1000 A400 Computer

Reference Manual

FEDERAL COMMUNICATIONS COMMISSION
RADIO FREQUENCY INTERFERENCE
INFORMATION

Printing History

The Printing History below identifies the edition of this manual and any updates that are included. Periodically, update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this printing history page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past updates; however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the Manual Numbering File or the Computer User's Documentation Index. (The Manual Numbering File is included with your software. It consists of an "M" followed by a five digit product number.)

```
First Edition ................. Dec 1986 ...................................
    Update 1 ................. Mar 1987 ...................................
Second Edition ............. Dec 1987 ...................................
```

Table of Contents

Chapter 3
Programming Information

Chapter 4
Dynamic Mapping System

Chapter 5
Code and Data Separation

Chapter 6
Interrupt System

Chapter 7
On-Board I/O (OBIO)

Chapter 8
Input/Output (I/O) System

Appendix A
Reference Tables and Conversions

List of Illustrations

Tables

ALPHABETICAL INDEX OF STANDARD INSTRUCTIONS

Introduction

General

The HP 1000 A400 Computers and Computer System (hereafter referred to as A400 computers) are low-cost members of the high-performance HP 1000 A-Series processor family. Memory and I/O are combined on the processor board, thereby eliminating two or more PC boards and some redundant logic. The A400 is available at these three levels of integration:

- HP 12100A Single board computer

- HP 2134A Computer in a 20-slot box

- HP 2424A Micro 14 Computer in low-cost 6-slot box

- HP 2434A Micro 24 Computer in 14-slot Micro/1000 box (with or without integral disc)

- HP 2484A/B Micro 24 System Processor Unit with RTE-A Operating System in Micro/1000 box, with or without integral disc.

Features

- Full A-Series compatibility
- 512k bytes of on-board parity memory
- 32k bytes of Boot PROM
- 10ms time base generator
- Full support of code and data separation (CDS)
- Battery backup and power fail recovery capabilities
- Support for up to four A-Series memory array cards
- 4-channel asynchronous MUX with two ports supporting modem control
- 300, 1200, 9600, 19.2k, and 76.8k baud transfers

Single-Board Computer Description

Product number 12100A is the A400 single-board computer that consists of a CPU, memory controller, 512k bytes of parity memory, and asynchronous serial I/O multiplexer. The A400 board will plug into any A600+/A700 backplane with up to four parity memory arrays and as many standard A-Series I/O cards as the backplane will hold.

The 12100A can be used as the foundation for building a customized computer system. A user-supplied computer cabinet provides the necessary backplane, power supply and cooling fans. The On-Board I/O (OBIO) cable is available for four 25-pin RS-232 serial I/O connections.

Since the A400 board computer is software compatible with current A-Series CPUs, the RTE-A Operating System and all existing A-Series software packages are available for building a computer system.

Table 1-1 gives a brief list of options and accessories available. The following publications provide additional information on selecting components and configuring a computer system.

- A400 Engineering and Reference Documentation, 02424-90003.

- HP 1000 Computer System Software Technical Data, Volume I, part no. 5953-8710 or replacement.

- HP 1000 Computer System Software Technical Data, Volume II, 5953-8721, part no. 5953-8721 or replacement.

- HP 1000 A-Series Computer Handbook, part no. 5954-8576 or replacement.

- HP 1000 System Designer's Guide, part no. 92077-90001.
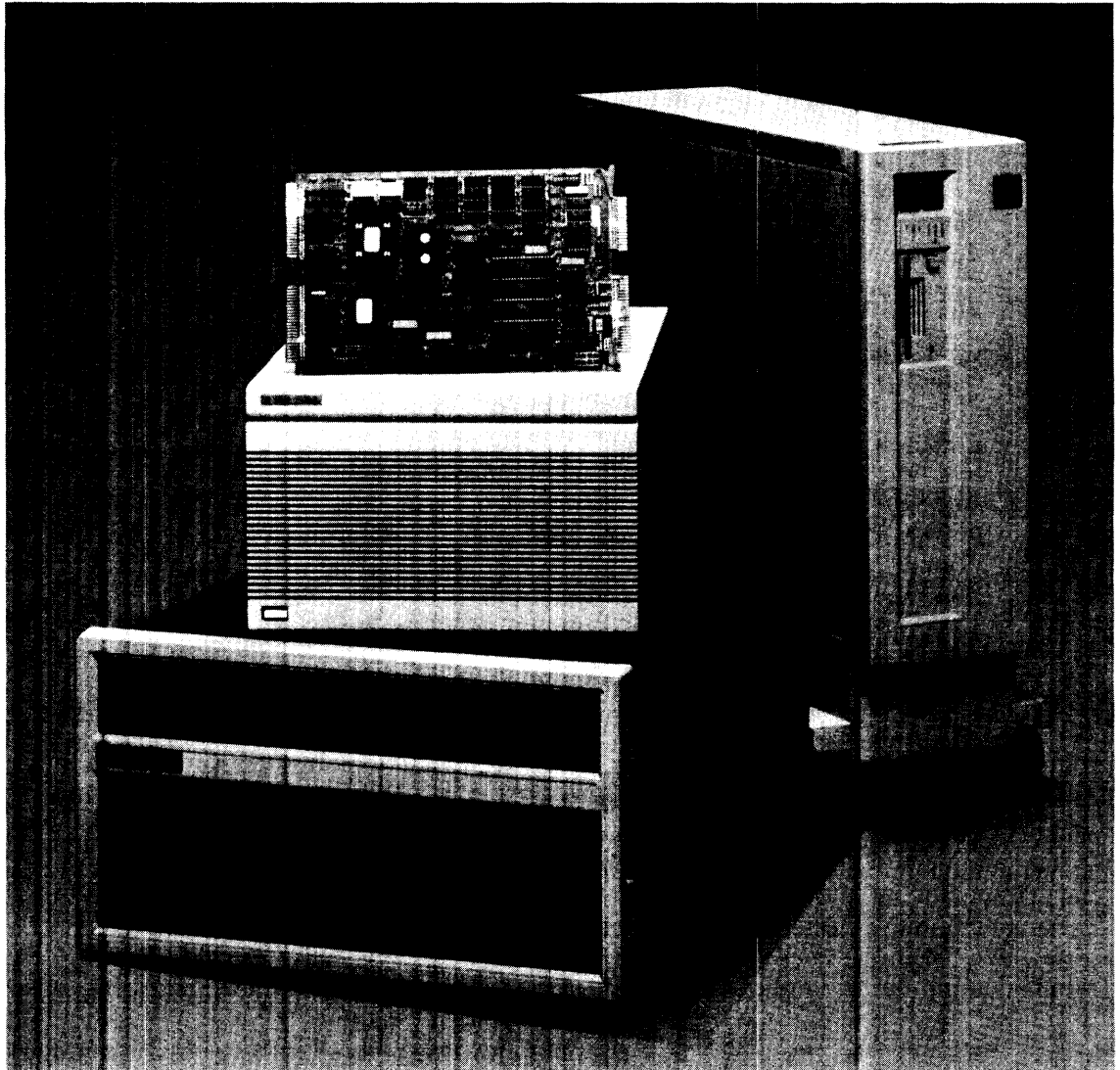
HP 2134A Box, HP 2424A Box, HP 12100A Board, and
HP 2434A/2484A/2484B Micro 24 Packages

Figure 1-1. HP 1000 A400 Computers

Architecture

The A400 computer architecture is based on a distributed intelligence concept that separates the processing of input/output (I/O) instructions from that of other instructions. Most central processor unit (CPU) circuitry resides on a single CMOS chip, packaged in a ceramic 168-pin grid array (PGA).

The A400 CPU executes the HP 1000 instruction set, including:

- Index instructions.
- Instructions for logical operations.
- Bit and byte manipulation.
- Base instructions that include
  - Single and double-precision floating point operations.
  - Double-integer and virtual memory addressing instructions.
  - Language instructions for increased program execution speed for higher level languages such as FORTRAN and Pascal.

The CPU also performs several system-level functions including memory protect, power fail/auto restart, time base generation, and parity error handling.

The entire micromachine plus much of the macromachine support circuitry (such as clock generation, memory control, interrupt control, and macro registers) is included in the CPU chip. The chip is implemented in 1.3 micron double-level CMOS standard cells and contains approximately 20,000 gate equivalents (1 gate equivalent = 2 input NAND gates), plus three custom register files (sixteen 16-bit registers). The mapping system, on-board memory array, control store, and backplane interface are implemented external to the processor chip on the PC board.

All I/O instructions are executed by custom CMOS input/output processor (IOP) integrated circuit chips that reside on the on-board I/O or on the individual I/O interface cards. A common backplane links the A400 with additional memory array and I/O cards. The instructions are fetched from memory and decoded by the CPU. When an instruction is decoded as being of the I/O type, it is broadcast on the backplane for execution by the appropriate I/O card. Because the OBIO and each I/O card is capable of operating independently of the CPU, the A400 can perform direct memory access (DMA) I/O transfers very efficiently. An I/O card interacts with the CPU only on DMA initiation and completion; beyond that, the entire high-speed transfer is handled by the I/O card, leaving the CPU free to work on other tasks. This achieves high efficiency in CPU and I/O throughput. Figure 1-2 is a simplified block diagram of the A400 computer.

Floating Point Processing

Floating point processing is implemented in firmware, providing high-speed dedicated logic that performs exceptionally fast single precision (32-bit) and double precision (64-bit) floating point arithmetic operations.

Virtual Control Panel

The Virtual Control Panel (VCP) program is an interactive program that enables an external device (such as a terminal) to control the CPU in a manner similar to a conventional computer control panel. It allows you to access the various registers (such as A, B, and P), examine or change memory, and control execution of a program. The VCP program is stored in the boot memory PROM of the A400. When not being used as the VCP, the VCP-enabled terminal can be used in the same way as any other terminal connected to the system, except that the BREAK key has a unique function in VCP mode. When the A400 computer is operating as a node in a computer network via NS/1000 or DS/1000-IV, the VCP device can be an adjacent computer in the network.

Bootstrap Loaders

There are several bootstrap loaders stored in the VCP PROM on the A400 board. The loaders provide program loading from several sources including disc drives, PROM storage modules, NS/1000 or DS/1000-IV network link, magnetic tapes, HP mini-cartridge tapes, and cartridge tapes of the disc drives. The first three loaders can be selected for auto-boot by switches on the A400 board; any of the loaders can be selected by operator commands via the VCP.

Self-Test Routines

Self-test routines are standard in the A400 computer and are stored in PROM on the A400 board. These routines are executed whenever computer power is turned on, or by operator command via the VCP, providing a check of specific areas of CPU logic.
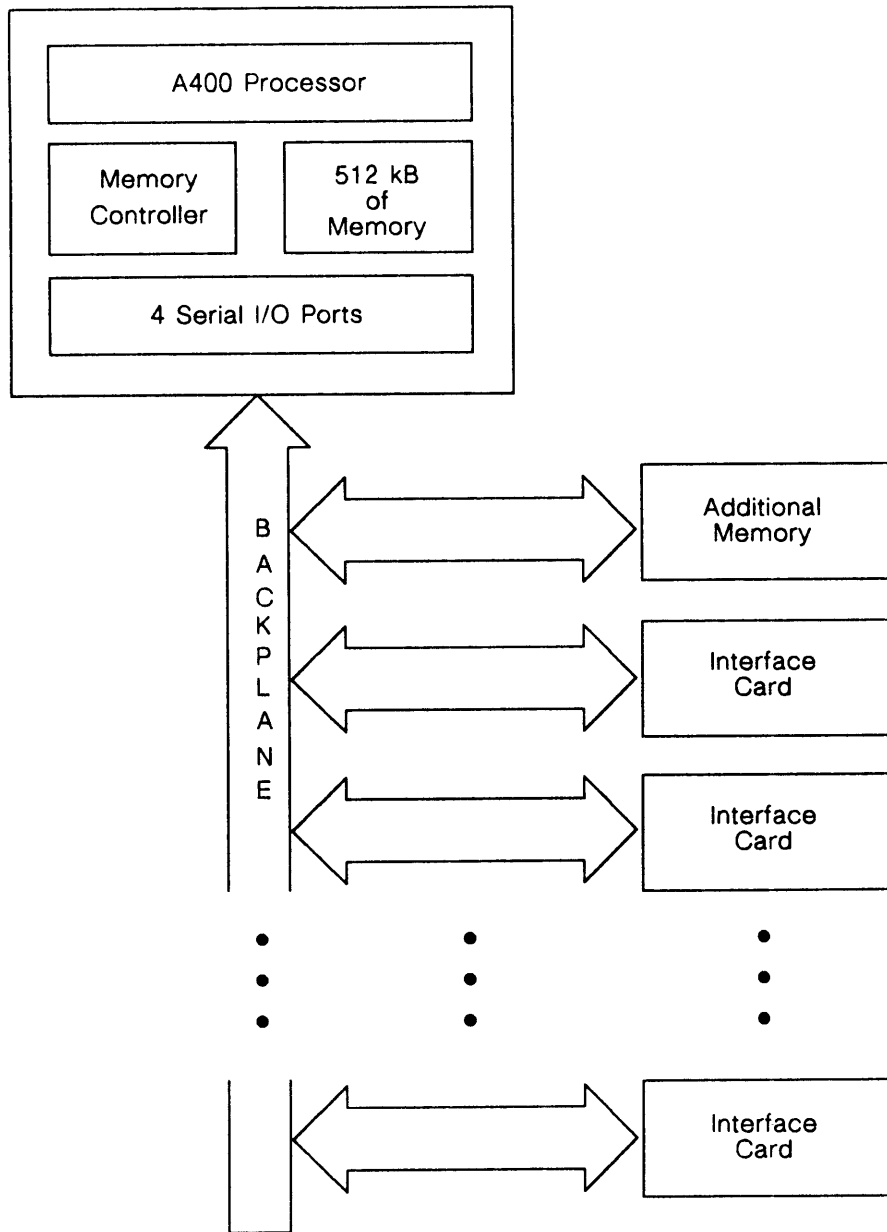
Figure 1-2. A400 Computer Simplified Block Diagram

L8600-36

Time Base Generator

The A400 board includes a time base generator (TBG) which can be used to update the real-time clock, establish timing points for task switching, and generate microcode timeout interrupts. The time base generator (TBG) can generate an interrupt every 10 milliseconds. The TBG is disabled at power up, and must be explicitly enabled using standard I/O instructions. Once enabled, the first TBG pulse will be generated 10 milliseconds later.

Power Supply

A400 computer packages have power supplies designed to continue normal operation in environments where AC line power may fluctuate widely. Input line voltages and frequencies may vary widely without affecting the operation of the computer.

Battery backup is supported by the A400 processor in 2134A, 2434A, and 2484A/B packages. An optional battery backup card and/or battery pack can be installed to sustain memory for 15 to 90 minutes (depending on memory size) in the event of a complete power failure, thus providing an automatic restart capability.

If power fail protection is needed in an HP 2424A package, an uninterruptable power supply (UPS) should be considered.

Another power supply enhancement to Micro/1000 configurations (and standard in the 2134A) is 25-kHz voltage that can be rectified at the load and used to power accessory plug-in cards used for measurement and control applications.

Input/Output (I/O)

Data transfer between the computer and I/O devices can take place under DMA control or program control. The DMA capability provides a direct link between memory and I/O devices. The total bandwidth available through multiple DMA channels is 4.4 million bytes (2.2 million words) per second.

On-Board I/O (OBIO)

The A400 computer contains on-board I/O (OBIO) circuitry consisting of an I/O processor chip, I/O master circuitry, and four serial I/O ports. The following functionality is provided by the OBIO circuitry:

Transmission Mode
    Bit-serial, Asynchronous

Capacity
    Four full duplex communications channels
    Two-channel modem support

Programmable Features
    Echo on or off
    Switch-programmable VCP port
    Record termination processing (any character)
    Baud rate selection (300, 1200, 9600, 19.2K, and 76.8K)
    Backspace processing
    ENQ/ACK handshake
    XON/XOFF handshake

Interface Levels
    RS-232 and V.28 (standard)
    RS-422/423 (user-supplied cable for each channel)
    Any mix of interface levels

Maximum Cable Length (depending on baud rate)
    RS-232, 15 meters (19.2k baud)
    RS-423, 50 meters (19.2k baud)
    RS-422, 1200 meters (19.2k baud)

Error Detection
    Framing errors and overrun errors
    Parity errors may be checked using software library routines.

Modem Control
    Interacts with six modem control lines: Clear to Send, Request to Send,
    Data Set Ready, Carrier Detect, Data Terminal Ready, Ring Indicator.

Two of the four ports drive the six modem control lines required for normal modem control. A "Break" on Port A will give you access to VCP, when enabled. The port processor also handles backspace processing to minimize the interrupts that the CPU must process. The I/O master and the single DMA channel that it provides are shared by the four port processors. The select code for OBIO is select code 77, and cannot be changed.

Support of A-Series I/O Cards

The A400 supports as many A-Series I/O cards as the card cage capacity allows. The I/O cards contain a custom I/O processor chip on each card, enabling each card to process its own I/O instructions and handle direct memory access (DMA) data transfers. The I/O system has a multilevel vectored priority interrupt structure with 60 distinct interrupt levels, each of which has a unique priority assignment. Any I/O device can be selectively enabled or disabled, or all I/O devices can be enabled or disabled under program control.

An important feature of the interface cards is a common-content Global Register which can be loaded with the select code of a specific I/O card. When the Global Register is enabled, all I/O instructions are executed only by the I/O card whose select code is in the Global Register. This not only facilitates setting up DMA transfers but also makes reconfiguration of an I/O driver a simple matter of changing the Global Register to the appropriate select code. Also, because the Global Register can direct I/O instructions to a specific I/O card, the I/O-instruction address bits can be used to access registers on an I/O card. This feature is utilized in the design of the I/O cards to increase their capabilities.

About one-third of the area on the interface cards is occupied by the I/O Master, which consists of an I/O processor chip and its associated logic. The I/O Master is also available in breadboard form for users who wish to design their own I/O cards. The I/O Master is described in detail in the HP 1000 A/L-Series Computer I/O Interfacing Guide, part no. 02103-90005.

Memory

The A400 computer contains a memory controller and 512k bytes of user RAM (with parity) resident on board. Also included are 2k bytes of boot RAM and 32k bytes of boot PROM on board.

Memory Controller

The memory controller interfaces to the processor, on-board memory, optional memory array cards, and I/O. The processor controls memory array card operations by connections across the backplane and a memory array front plane. Interaction with I/O occurs across the backplane.

The memory controller is responsible for the following functions:

- Generating handshake signals for backplane (BUSY, VALID).

- Latching data and addresses during memory cycles.

- Generating interrupt signals (MPV, PE).

- Arbitrating memory and refresh cycles.

- Generating read strobes for array cards.

- Protecting memory during illegal accesses.

- Accessing loader and front panel firmware.

- Inhibiting array cards during protected access.

- Maintaining memory data during power failures.

- Verifying that physical address indeed accesses an existing array card and if not drive all "1's" onto backplane.

- Determine whether the requested memory cycle should occur in two or three clock signal cycles.

Memory Access

The A400 memory system is "dynamically mapped" which provides the ability to access more than 32k words of 16-bit data. Map RAMs generate the physical address of data to be accessed during a memory cycle. Mapping widens the 15-bit address bus on the backplane (which can access up to 32k words of memory) to a 24-bit address bus (combined backplane and frontplane) which can access up to 16M words of memory. The memory array cards are word addressable.

Memory access can be initiated by either the processor or by an I/O device using DMA. Processor accesses can be read and/or write protected by two bits that are stored in the map RAMs. Thus, processor access to protected memory will cause an interrupt to occur (if memory protect is enabled) and the access will be aborted. An I/O device using DMA can access protected memory, however. This is true for either a read or write access.

The detection of a parity error on a read causes the assertion of a parity error signal on the backplane. This condition signal is handled by the processor or by the I/O card depending on which initiates the access.

Memory Expansion

The A400 processor allows main memory expansion beyond the on-board 512k bytes. A maximum of four megabytes of main memory can be supported by adding the following memory array cards:

- 12103C 512k byte parity array card
- 12103D 1M byte parity array card
- 12103K 2M byte parity array card
- 12103L 4M byte parity array card
- 12103M 8M byte parity array card

Error correcting code (ECC) memory and the 12103B 256k byte memory array card are not supported.

When a 12103C or 12103D array card is added to the system, its size must conform to the address space boundaries. A compatible size is indicated when an integer results from dividing the size of existing memory by the size of the array card to be added. The 12103K/L/M cards are self-configuring and the integer rule does not apply. They can be added on any half-megabyte boundary.

When installing array cards there is no need to physically identify the array cards (such as jumper or switch setting to set recognition of physical address space). The memory controller automatically designates the on-board memory as the first memory address space. The array cards configure themselves in ascending order going away from the on-board memory. The beginning of the memory is, therefore, the on-board memory and the end of memory is on the array farthest from the on-board memory.

It is possible to use partially loaded array cards as long as the total memory on the array is either 512k or 1M bytes. The partially loaded arrays can be incorporated into the module self-configuring scheme. There can be a total of up to four array cards in the system. For the proper order of installation, refer to the appropriate installation and service manual.

Software

Software support for the A400 computers begins with RTE-A, a member of HP's family of Real-Time Executive (RTE) operating systems. RTE-A is a real-time multi-programming, multi-user operating system designed to take full advantage of the A400 I/O structure to enhance overall CPU and I/O throughput. RTE-A offers a wide range of configurations, from a small, memory-based, execute-only system to a full disc-based system with on-line program development. Utilizing the A400 mapped memory system, RTE-A supports user partitions of up to 64k bytes and memory sizes up to 32 megabytes.

Memory can be divided into fixed and dynamically allocated partitions at system generation time. Critical programs can be made resident in fixed partitions to ensure fastest possible response to requests for their execution. Other programs can be assigned partitions from the dynamic memory pool according to need, using the smallest available block of memory.

RTE-A also supports Virtual Memory Addressing (VMA) for access to data arrays much larger than main memory (up to 12.6 megabytes). The disc functions as an extension of main memory so far as data is concerned, in a manner that is transparent to the user and does not require any special programming. In addition, RTE-A supports a special case of VMA, called Extended Memory Area (EMA). With EMA, up to two megabytes of a program's data can be in main memory at once, which affords faster processing of data arrays small enough to use the EMA capability. The programmer chooses the data array handling mode at program load time.

Disc-based RTE-A systems support program development in FORTRAN 77, Pascal/1000, BASIC/1000, and Macro/1000 Assembly Language. Program development for the A400 can also be performed on an HP 1000 System under RTE-6/VM or RTE-IVB.

The diagnostic packages listed in Table 1-1 may be used for testing and fault location.

HP Interface Bus (HP-IB)

Among the I/O interface cards available for the A400 computer is the HP 12009A HP-IB Interface Card which can interface the A400 computer to a variety of HP peripherals and other equipment compatible with the Hewlett-Packard Interface Bus (HP-IB). (HP-IB is the Hewlett-Packard implementation of IEEE standard 488-1978, "Digital Interface for Programmable Instrumentation".) A single HP 12009A can control up to 14 HP-IB instruments and several can be used to achieve concurrent operation of multiple HP-IB instrumentation clusters under the RTE-A multiprogramming operating system.

Computer Network

You can configure the A400 computer into an HP NS/1000 or DS/1000-IV Distributed System by using either an HP 12007B or an HP 12044A HDLC Interface. Both of these interfaces support the high-level data link communications (HDLC) protocol, functioning as a preprocessor to handle low and medium levels of protocol processing. The A400 computers can be easily mixed with other members of the HP 1000 family in a single computer network. The HP 12042B Programmable Serial Interface (PSI) card allows sophisticated OEMs to design customized protocols for networks. Hewlett-Packard offers a customer training course on how to program the PSI card.

Expansion and Enhancement

Table 1-1 lists accessory products available to expand or enhance the A400 computers.

Specifications

The HP 1000 A-Series Computer Handbook, part no. 5954-8576 or replacement, provides complete specifications for the A400 computers and systems.

Table 1-2 provides an abridged set of A400 specifications. Except where indicated, the specifications are applicable to the HP 2134A/2424A/2434A Computer and the HP 2484A/B Computer System. Both the computer and the computer system have been product accepted by the Underwriters' Laboratories (UL) and the Canadian Standards Association (CSA). The A400 computer and system also meet the RFI standards of the Federal Communications Commission (FCC) and Verband Deutches Electrotechnikes (VDE).

Table 1-1. Options and Accessories

DESCRIPTION	HP PRODUCT NO.
230 Vac Operation	Opt 015
512K Byte Memory Array Card	12103C
1M Byte Memory Array Card	12103D
2M Byte Memory Array Card	12103K
4M Byte Memory Array Card	12103L
8M byte Memory Array Card	12103M
Memory Connector for one memory array card	12038A
Memory Connector for two memory array cards	12038B
Memory Connector for three memory array cards	12038C
Memory Connector for four memory array cards	12038D
Asynchronous Serial Interface	12005B
Parallel Interface	12006A
HDLC Interface (modem operation)	12007A
PROM Storage Module	12008A
HP-IB Interface	12009A
Intelligent Breadboard	12010A
Extender Board	12011A
Priority Jumper Card	12012A
Input/Output Extender	12025A/B-002
8-Channel Asynchronous Multiplexer	12040D
Multi-use 8-channel Multiplexer	12041A/B
Programmable Serial Interface	12042A
Multi-use Programmable Serial Interface	12043A
HDLC Interface (hard-wired operation)	12044A
High-Level Analog Input Card*	12060A
Expansion Multiplexer Card*	12061A
Analog Output Card*	12062A
16-In/16-Out Isolated Digital I/O Card*	12063A
Color Video Interface	12065A
DS/1000-IV Data Link Slave Interface	12072A
DS/1000-IV Modem Interface to HP 3000	12073A
LAP-B Network Interface	12075A
DS/1000-IV Direct Connect Interface to HP 3000	12082A
LAN Interface	12076A
Data Link Master Interface	12092A
HP-IB Extender Card	37203L
Integral Modem Interface	37222A
Battery Backup Card	12154A
Battery Backup for 2134A	12157B
25 kHz Sine Wave Card	12159A
Diagnostic Package for A400 processor and interfaces**	24612A
Diagnostic Package for A400-compatible hard disc drives and magnetic tape units**	24398B
HP-1B Extender card	37203L
Integral Modem Interface	37222A

  * Accessory for HP 2134A, 2434A, and 2484A/B packages only.
 ** Included with the HP 2484A/B System.

Table 1-2. Specifications

SPECIFICATIONS COMMON TO THE HP 2134A, 2424A, 2434A, AND 2484A/B

CENTRAL PROCESSOR

Word Size: 16

Instruction Set: 205 standard instructions.
  Memory Reference: 14
  Register Reference: 43
  Input/Output: 13
  Extended Arithmetic: 10
  Index: 34
  Bit, Byte, Word Manipulation: 10
  Floating Point (single precision):   8
  Language: 11
  Dynamic Mapping: 40
  Double Integer: 12
  Virtual Memory: 9
  Operating System: 4
  Floating Point (double precision):   8
  Code and Data Separation: 21

Registers:
  Accumulators: Two (A and B), 16 bits each. Implicitly addressable, also explicitly addressable as memory locations.

  Index: Two (X and Y), 16 bits each.

  Memory Register: One (P), 15 bits.

  Base: One (Q), 15 bits; one (C), 1 bit.

  Bounds: One (Z), 16 bits.

  Supplementary: Two (overflow and extend), 1 bit each.

Time Base Generator
  Interrupt: A time base generator interrupt is provided for maintaining a real time clock. The interrupt request is made when the CPU signals, at 10-millisecond intervals, that its internal clock is ready to roll over. Timing accuracy of the time base generator is ±2.16 seconds per 24-hour day.

Table 1-2. Specifications (Continued)

SPECIFICATIONS COMMON TO THE HP 2134A, 2424A, 2434A, AND 2484A/B (Continued)

MICROMACHINE

Address Space:	16,384 words (256 blocks of 64 words each).
Microinstruction Word Size:	32 bits.
Microaddress Space	16k microwords.
Word Types:	six
Cycle Time:	227 nanoseconds
Microorders:	198
Operations:	14
Special:	56
ALU and Conditional:	48
Store:	32
B-Bus:	32
A-Bus:	16

MEMORY

Memory Structure:	64 pages minimum of 2048 bytes per page, with direct access to current page or base page (page 0), and indirect or mapped access to all pages.
Memory Size:	512 kb parity memory on board. Up to 32 Mbytes with 12103 parity array modules.
Memory Cycle Time:	454 nanoseconds.
Memory Parity Checking:	Parity logic in the memory controller continuously generates correct parity for all words written into memory and monitors the parity of all words read out of memory. Either odd or even parity can be selected programmatically. A parity error generates an interrupt to memory location 00005, which can contain a JSB to a user-supplied parity error handling subroutine or a halt instruction.

INPUT/OUTPUT

Determination of I/O Address:	I/O address select code is set for each interface card by select code switches on the card and is therefore independent of interface card position in the card cage. Select code of OBIO is 77 and cannot be changed.

Table 1-2. Specifications (Continued)

SPECIFICATIONS COMMON TO THE HP 2134A, 2424A, 2434A, AND 2484A/B (Continued)

INPUT/OUTPUT (Continued)

I/O Device Interrupt Priority:	Depends upon I/O interface card position in the card cage with respect to the processor. OBIO always has the highest priority in the system.
Interrupt Masking:	The I/O Master Logic includes an interrupt mask register which provides for selective inhibition of interrupts from specific interfaces under program control. This capability can be programmed to temporarily cut off undesirable interrupts from any combination of interfaces when they could interfere with crucial transfers.
Interrupt Latency:	8.00 to 60.00 microseconds. 10.00 microseconds typical. (Interrupts cannot be serviced until a DMA cycle or an instruction in progress has completed execution.) The worst-case latency of 60.00 microseconds is based upon time to complete a double-precision floating point divide (.TDIV), the longest non-interruptible standard instruction.
Direct Memory Access (DMA):	The I/O processor chip supports DMA capability on OBIO and each I/O interface, which reduces the number of interrupts from one data item (byte or word) to one per complete DMA block.
DMA Latency:	Time interval from Service Request by an I/O device through completion of the DMA I/O data transfer to or from the I/O interface is 0.83 microseconds for input, 1.25 microseconds for output for the interface with highest hardware I/O priority.
Data Packing Under DMA:	When byte mode is specified in Control Word instructions, the I/O processor chip automatically manages byte packing or unpacking.
Maximum Achievable DMA Burst Rate:	2.2 million words (4.4 megabytes) per second.

SAFETY AND RFI QUALIFICATION	HP 1000 A400-Series products are UL listed and CSA certified to meet Underwriters' Laboratories (UL) and the Canadian Standards Association (CSA) standards for safety. The A400-Series also complies with the RFI standards of the Federal Communications Commission (FCC) and Verband Deutches Electrotechnikes (VDE).

Table 1-2. Specifications (Continued)

SPECIFICATIONS COMMON TO THE HP 2134A, 2424A, 2434A, AND 2484A/B (Continued)

DC REQUIRED

MODEL	+5V dc	+5M dc	+12V dc	−12V dc
12100A Processor Board	4.2A	1.5A	0.065A	0.090A
(standby)*		0.63A		
12103C 512kb Memory	1.1A	1.0A		
(standby)*		0.6A		
12103D 1024kb Memory	1.3A	1.6A		
(standby)*		1.0A		
12103K 2Mb Memory	0.9A	1.0A		
(standby)*		0.61A		
12103L 4Mb Memory	0.9A	1.3A		
(standby)*		0.65A		
12103M 8Mb Memory	0.9A	2.1A		
(standby)*		0.71A		
12005B Async. Serial Interface	1.6A		0.2A	0.1A
12006A Parallel Interface				
With +5V logic	1.94A		0.18A	
With +12V logic	1.61A		0.18A	
12007A/B HDLC Interface (modem)	2.6A		0.4A	0.2A
12008A PROM Storage Module	2.0A		0.1A	
12009A HP-IB Interface	2.1A		0.1A	
12010A Breadboard Interface**	0.8A		0.4A	
12040B/C/D Async. Multiplexer	2.5A		0.1A	0.1A
12041A/B Multi-use MUX	2.5A		0.1A	0.1A
12042B Prog. Serial Interface	2.6A		0.4A	0.2A
12043B Multi-use PSIF	2.6A		0.4A	0.2A
12044A HDLC Interface (direct)	2.4A		0.3A	0.1A
12060A Analog Input†	1.1A			
12061A Analog Multiplexer†	0.1A			
12062A Analog Output†	1.2A			
12063A Digital I/O†	1.0A			
12065A Color Video Interface	3.7A		0.5A	0.02A
12072A Data Link Interface	1.5A		0.2A	0.1A
12073A Modem HP 3000 Interface	2.6A		0.4A	0.2A
12075A LAP-B Network Interface	2.6A		0.4A	0.2A
12076A LAN Interface	4.5A		0.5A	0.38A
12082A 3000 Interface (direct)	2.4A		0.3A	0.1A
12092A Data Link Master IF	2.6A		0.4A	0.2A
37203L HP-IB Extender Card	0.8A			
37203L w/opt 001	0.8A			
37222A Integral Modem IF	1.2A		0.1A	0.1A

* Standby indicates the current supplied from the optional battery backup during power failure.

** 12010A current requirement listed here is for the I/O Master circuitry only; logic added by the user will require additional current.

† Requires 25 kHz power supply accessory. Available only in the HP 2134A, 2434A, and HP 2484A/B.

Table 1-2. Specifications (Continued)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2424A COMPUTER

ELECTRICAL SPECIFICATIONS

AC Power Required:	
Line Voltage:	86-140V (120V -28%/+;17%) standard; 172-276V (240V -28%/+15%) option 015.
Line Frequency:	47.5 to 66 Hz.
Inrush Current (cold power-up):	20.0A max (at 120 Vac). 40.0A max (at 240 Vac).
Nominal Current:	2.5A (120 Vac); 2.5A max (86 Vac). 1.3A (240 Vac); 1.3A max (172 Vac).
Maximum Power Required:	216 Watts (300 VA).
Power Factor:	0.70 minimum, current leading the voltage.

PHYSICAL CHARACTERISTICS

Dimensions:	
Height:	205 mm (8.07 in.)
Height w/feet:	208 mm (8.19 in.)
Width:	325 mm (12.80 in.)
Depth:	500 mm (19.69 in.)
Weight:	13.2 kg (29 lb)
Ventilation:	Air intake is through the front; exhaust is through the rear.
Volume:	Approximately 70 cubic feet per minute.

ENVIRONMENTAL SPECIFICATIONS

Temperature:	
Operating:	$0°$ to $60°$ C ($32°$ to $140°$ F)
Non-operating:	$-40°$ to $75°$ C ($-40°$ to $167°$ F)
Relative Humidity:	5% to 95% non-condensing.
Altitude:	
Operating:	To 4.6 km (15,000 ft.)
Non-operating:	To 15.3 km (50,000 ft.)

Table 1-2. Specifications (Continued)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2424A COMPUTER (Continued)

ENVIRONMENTAL SPECIFICATIONS (Continued)

Vibration and Shock:

Shock

Operating: 1.5 g peak, $^1/_2$ sine, 6 to 9 ms duration, 45 Hz crossover.

Non-operating: 7.0 g peak, $^1/_2$ sine, 6 to 9 ms duration, 45 Hz crossover.

Operating Vibration:

Frequency (Hz)	Power Spectral Density (g²/Hz)
5	0.002
5 - 15	-1/.5 dB/octave
15	0.0015
12 - 200	-6.0 dB/octave
200 - 350	0.00012
350 - 500	-6.0 dB/octave

g's rms = 0.43

POWER SUPPLY

Output:

Dc voltages and tolerances:
+5V   ±2%
+12V   +6/-3%
-12V   +6/-4%

Maximum Output Current Ratings:

+5V   +12V   -12V
23A   3.6A   1.1A

Short Circuit Protection:

All dc and ac power outputs are fault protected for short circuits. The power supply will shut down if any of the outputs are short circuited at turn on.

5V Output Overvoltage Protection:

The +5V output is sensed for overvoltage and the +5V supply shuts down if its output voltage exceeds 5.5V. The ac power switch must be cycled to reset the +5V output.

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2434A AND 2484A/B SYSTEM

ELECTRICAL

AC Power Required:
  Line Frequency:

47.5 to 66 Hz

  Line Voltage:

88-138V (115V -25%/+20%) standard;
178-276V (230V -23%/+20%) Option 015.

  Operating Current:

6A, max. in 115V configuration; 3A, max. in 230V configuration.

PHYSICAL CHARACTERISTICS

Dimensions:
  Height: 178 mm (7 in.)
  Width: 483 mm (19 in.)
  Depth: 648 mm (26 in.)

Table 1-2. Specifications (Continued)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2434A AND 2484A/B SYSTEM (Continued)

PHYSICAL CHARACTERISTICS (Continued)

Weight:

Without Integrated Discs:	16.3 kg (36 lb)		
Integrated Disc Added:	using 12022A Disc Controller Interface:	15Mb 2.27 Kg (5 lb)	20Mb 2.7 kg (6 lb)
	using 12009A HP-IB Interface: 3.0 kg (6.6 lb)		

Ventilation: Air intake is through the left; exhaust is out through the right.

ENVIRONMENTAL SPECIFICATIONS

Temperature:

Operating: $0°$ to $55°$ C ($32°$ to $131°$ F) up to 3.1 km (10,000 ft); without internal discs. Maximum temperature is linearly derated $2°$ C ($3.6°$ F) for each 304.8m (1000 ft) increase in altitude. Resulting temperature range is $0°$ to $45°$ C ($32°$ to $113°$ F) at 4572 meters (15,000 ft).

                $10°$ to $40°$ C ($40°$ to $113°$ F) with internal discs; maximum rate of change $<10°$ C ($18°$ F) per hour.

Non-operating: $-40°$ to $75°$ C ($-40°$ to $167°$ F) (maximum temperature with internal discs is $60°$ C ($140°$ F)).

Relative Humidity:

Without Internal Disc: Operating: 5% to 95% with maximum wet bulb temperature not to exceed $40°$ C ($140°$ F), excluding all conditions which cause condensation.

With Internal Disc: Operating: 20% to 80% with maximum wet bulb temperature not to exceed $29°$ C ($85°$ F), excluding all conditions which cause condensation.

Non-operating: 5% to 95% non-condensing.

Altitude:

Operating: To 4.6 km (15,000 ft.)

Non-operating   To 15.3 km (50,000 ft.)

Vibration and Shock: HP 1000 A400-Series products are type tested for normal shipping and handling shock and vibration. (Contact factory for review of any application that requires operation under continuous vibration).

Table 1-2. Specifications (Continued)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2434A AND 2484A/B SYSTEM (Continued)	
POWER SUPPLY Output: Maximum DC Current Available to I/O and Memory Cards:	DC voltages and tolerances: +5.1V ±2% +12V +6/-3% -12V +/-6% +5.1V +5M +12V -12V 45.8A 5.5A 6.9A 2.9A Note that combined current available from +5.1V and +5M supplies is 50A, maximum. All dc power outputs are fault protected for short circuits. The power supply will shut down if any output is short-circuited at turn-on.
25 KHZ AC VOLTAGE	HP 12159A 25 kHz Power Module The Power Module provides 27.0V p-p± 8%. 25 kHz nominal, split phase from three pins on the backplane-mating connector. Maximum output power is 30 watts.
BATTERY BACKUP Recharge Time: Battery Type:	HP 12154A Battery Backup Card The Battery Backup Card provides from 45 to 210 minutes of memory sustaining power, depending upon system configuration, state of charge, and temperature; additional hold-up time can be achieved by connecting an external battery. 14 hours for fully discharged battery pack. Nickel cadmium.

Table 1-2. Specifications (Continued)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2134A COMPUTER

ELECTRICAL SPECIFICATIONS

AC Power Required:

Line Voltage:	86-138V(115V -25%/+20%) standard; 178-276V (230V -23%/+20%) option 015.
Line Frequency:	47.5 to 66 Hz.
Maximum Power Required:	700 Watts.

PHYSICAL CHARACTERISTICS

Dimensions:

Height:	266 mm (10.5 in.).
Width:	483 mm (19 in.).
Depth:	612 mm (24 in.).
Weight:	29.5 kg (65 lb).
Ventilation:	Four fans provide approximately 10.1 cubic meters per minute (360 CFM) front-to-rear airflow, half through the card cage and half to cool the power supply.

ENVIRONMENTAL SPECIFICATIONS

Temperature:

Operating:	$0°$ to $55°C$ ($32°$ to $131°F$) up to 3048 meters (10,000 ft); $0°$ to $45°C$ (32 to $113°$ F) up to 4572 meters (15,000 ft).
Non-operating:	$-40°$ to $75°$ C ($-40°$ to $167°$ F). derated to: $-40°$ to $60°C$ ($-40°$ to $140°F$) with 12157B battery backup.
Relative Humidity:	5% to 95% non-condensing.

Altitude:

Operating:	To 4.6 km (15,000 ft).
Non-operating:	To 15.3 km (50,000 ft).
Vibration and Shock:	HP 1000 A400-Series products are type tested for normal shipping and handling shock and vibration. (Contact factory for review of any application that requires operation under continuous vibration.)

Table 1-2. Specifications (Continued)

SPECIFICATIONS APPLICABLE ONLY TO THE HP 2134A COMPUTER (Continued)

POWER SUPPLY

Output:
DC voltages and tolerances:
+5.1V   ±2%
+12V   +6/−3%
−12V   +/−6%

Maximum DC Current Available
to I/O and Memory Cards:

+5.1V	+5M	+12V	−12V
65.8A	8.5A	5.4A	3.4A

Note that combined current available from +5.1V and +5M supplies is 70A, maximum.

All dc power outputs are fault protected for short circuits. The power supply will shut down if any output is short-circuited at turn-on.

25 KHZ AC VOLTAGE

Voltage: 39V RMS

Current: 1.5A, max.

Power: 50 watts, max.

BATTERY BACKUP

12157B Battery Backup

The Battery Backup provides from 15 to 90 minutes of sustaining power for up to four memory array cards, depending upon system configuration, state of charge, and temperature. Additional hold-up time can be achieved by connecting an external battery.

Battery Type: Sealed lead-acid.

Operating Features

This chapter describes the bootstrap loaders, the Virtual Control Panel (VCP) program, and the central processor registers accessible to the programmer.

Hardware Registers

The A400 computer has several working registers that can be selected for display and modification via the VCP program. (Interface card registers are described in the I/O System chapter of this manual and in the interface card reference manuals.) The functions of the A400 registers are described in the following paragraphs.

A-Register

The A-Register is a 16-bit accumulator that holds the results of arithmetic and logical operations performed by programmed instructions. This register can be addressed directly by any memory reference instruction as location 000000 (octal), thus permitting interrelated operations with the B-Register (for example, "add B to A" and "compare B with A") using a single-word instruction.

B-Register

The B-Register is a second 16-bit accumulator that can hold the results of arithmetic and logical operations completely independent of the A-Register. The B-Register can be addressed directly by any memory reference instruction as location 000001 (octal) for interrelated operations with the A-Register.

P-Register

The 15-bit P-Register holds the address of the next instruction to be fetched from memory.

Extend (E) Register

The one-bit extend (E) Register is used by rotate instructions to link the A- and B-Registers or to indicate a carry from the most-significant bit (bit 15) of the A- or B-Register by an add instruction or an increment instruction. This is of significance primarily for multiple-precision arithmetic operations. If already set (logic 1), the extend bit cannot be cleared by a carry. However, the extend bit can be selectively set, cleared, complemented, or tested by programmed instructions.

Overflow (O) Register

The one-bit overflow (O) Register indicates that an add instruction, divide instruction, or an increment instruction referencing the A- or B-Register has caused (or will cause) the accumulators to exceed the maximum positive or negative number that can be contained in these registers. The overflow bit can be selectively set, cleared, or tested by programmed instructions.

Central Interrupt Register

The central interrupt register is a six-bit register that holds the select code of the last interface card or internal condition whose interrupt request was serviced.

Violation Register

The violation register is a 15-bit register that records the logical address of any fetched instruction that violates memory protection rules.

Parity Error Register

The parity error register is a 24-bit register that stores the physical address of the last memory location that caused a parity error.

Interrupt System Register

The interrupt system register is a one-bit register that indicates the status of the interrupt system. When set (logic 1), the interrupt system is enabled; when cleared (0), the interrupt system is disabled.

The X- and Y-Registers are 16-bit registers that are accessed through the use of the 32 index register instructions and two jump instructions described in Programming Information chapter.

WMAP-Register

The WMAP-Register is a 16-bit register that holds the logical map numbers used for memory references by Dynamic Mapping System instructions. (Refer to the Dynamic Mapping System chapter for more information.)

Virtual Registers

Two virtual registers, M and T, are created by the Virtual Control Panel program and can be accessed, via the VCP, to examine or change a program in memory or to manually create a program in memory.

M-Register

The M-Register holds the address of the memory cell currently being read from or written into by the Virtual Control Panel.

T-Register

The T-Register indicates the contents of the memory location currently pointed to by the M-Register.

Self-Test

Self-test is a microcoded diagnostic executed when the computer is powered up. Test results are displayed by the LEDs on the A400 board.

The microcoded self-test tests the CPU kernel prior to the VCP pre-test loading and expanding on the kernel testing. The first section of the test verifies the correct operation of the micromachine sequencer. Processor chip data paths are tested, followed by some basic functions of the ALU and testing of the PROM area where VCP is located.

Before execution of self-test, all of the LEDs will be illuminated. The entire self-test executes within a few seconds. In the event of a test failure, the LED pattern will indicate the last test that was started but not finished. Refer to Table 2-1. A summary of the four sections of CPU logic tested by the microcode self-test follows:

1. CT-Register Test. Verifies that the CT-Register can store and read data. There is an initial test for stuck bits, followed by a test of the ability of the ALU circuitry to pass immediate data and execute an "AND" micro-order. After the initial test of the CT-Register passes, a test of storing all possible values into the CT-Register is performed.

2. Micromachine Stack Test. This test verifies that the stack logic of the micromachine and the "JSB" and "JSBL" micro-orders are functioning. R00 and R01 of the register file (A- and B-Registers) also are tested for their ability to store values.

3. P-Register Test. This test verifies the ability of the P-Register to store a value. It also checks the integrity of the A-bus and the "IP" micro-order's ability to increment the P-Register.

4. VCP Prom Test. This test verifies the ability to read the VCP PROM area and to test the integrity of the data stored in the VCP PROM. At this point, the VCP pre-test attempts to boot.

Table 2-1. Self-Test Failure Indicators

SELF-TEST FAILURE INDICATIONS		NORMAL INDICATIONS	
●●●●●●●●	Microcode self-test	○○○○○○○●	Loader executing
●●●●●●●○	1st Instr fetch	○○○○○●●●	VCP executing
●●●●●●○○	Basic CPU	○○○○●●●●	BCM executing
●●●●●○○○	1K boot RAM	○○○●○○○○	DDL executing
●●●●○○○○	I/O interrupts	○○○○○○○○	Prgm executing
●●●○○○○○	Main Memory *		
●●○○○○○○	Interface card *	● = on, ○ = off * – May flash	

VCP Pre-Test

The VCP pre-test is the primary power-up test. It is automatically executed upon the completion of the microcode self-test, during initial power-up, after any powerfail/restart, or by the %T command from the VCP prompt. VCP pre-test will test and verify the following CPU macroinstructions:

● Memory reference instructions
● Register reference instructions
● Input/output instructions
● Extended arithmetic memory reference instructions
● Extended arithmetic register reference instructions
● Extended instructions

The VCP pre-test also tests the 1k of boot RAM memory, memory controller functions, the memory arrays, CPU I/O, and the individual I/O chips.

The path of communication between the VCP pre-test and the user is DIP switch U1601, the LEDs, and the VCP terminal console, if there is one in use. By setting DIP switch U1601 with the proper pattern, the pre-test can be looped unconditionally or looped until an error is detected. As portions of the pre-test complete, a corresponding LED becomes extinguished. When an error occurs, the LEDs display the section number that failed, followed by information on the error.

If there is a VCP terminal console active, the error will be displayed if possible. If it is a soft error, then the VCP prompt along with information about the system will be displayed along with the error. The ID and select code of all the I/O cards found in the system, and a CPU identification message is also displayed.

User Interface and Control

There are two switch settings that are reserved for the looping control of the VCP pre-test. The remainder of the switch settings indicate the proper action to take when the test completes. Refer to Table 2-2 and 2-3.

If the failure is in the CPU section or 1k boot RAM section, the test will loop on the address of the failure and the LEDs will indicate that the CPU section or 1k boot RAM section failed. If the failure information indicated by the VCP pre-test is not sufficient to determine the failing component, it may be necessary to load and execute the DDL diagnostics. If the failure is in the CPU section, it may not be possible to execute DDL diagnostics. A failure of the CPU section indicates a defective processor chip.

If a memory error occurs, the memory size and section will be displayed. If an I/O error occurs, the type of error and the select code will be displayed. The pre-test attempts to display the error information immediately after the error is detected. The error display is followed by a VCP prompt, indicating that no further testing was done, and control was passed to VCP.

If no VCP is found, an error condition is displayed even though it is not necessary to have a VCP with automatic boot. If multiple VCPs are found, an error will also occur.

A status message is displayed with a VCP prompt under the following conditions:

- The self-test passes or the VCP pre-test fails the memory or I/O portion, and looping is not activated.

- A VCP terminal is found, and pre-test concludes.

At this point, the VCP pre-test may be re-executed using the %T command or any of the other VCP commands may be executed.

Table 2-1 indicates the LED indications for the self-test. For more information, refer to the RTE-A Quick Reference Guide, part no. 92077-90020, or the appropriate installation and service manual. If the failure occurs in either the main memory test or the I/O interface test, additional information is displayed on the LEDs following the section failure indicated in Table 2-1.

NOTE

Only parity memory is supported by the A400 computer. If ECC memory is detected, the section indicator will be all 1's.

If there is a main memory failure, the section indicator followed by 10xxxxxx (where xxxxxx is the 32k block of memory that failed), is displayed on the LEDs. If xxxxxx is 0, the problem could be in the parity circuitry or in the map RAMs, and could be caused by bad memory controller circuitry. If xxxxxx is all 1's, the problem is that ECC memory was found, which the processor does not support.

If there is an I/O interface failure, the section indicator followed by 10xxxxxx is displayed on the LEDs, where xxxxxx is either:

1. The select code of the failing interface card, if xxxxxx is greater than 17B; or

2. The I/O subsection that failed, if xxxxxx is less than 20B.

Table 2-2. A400 Switch Locations
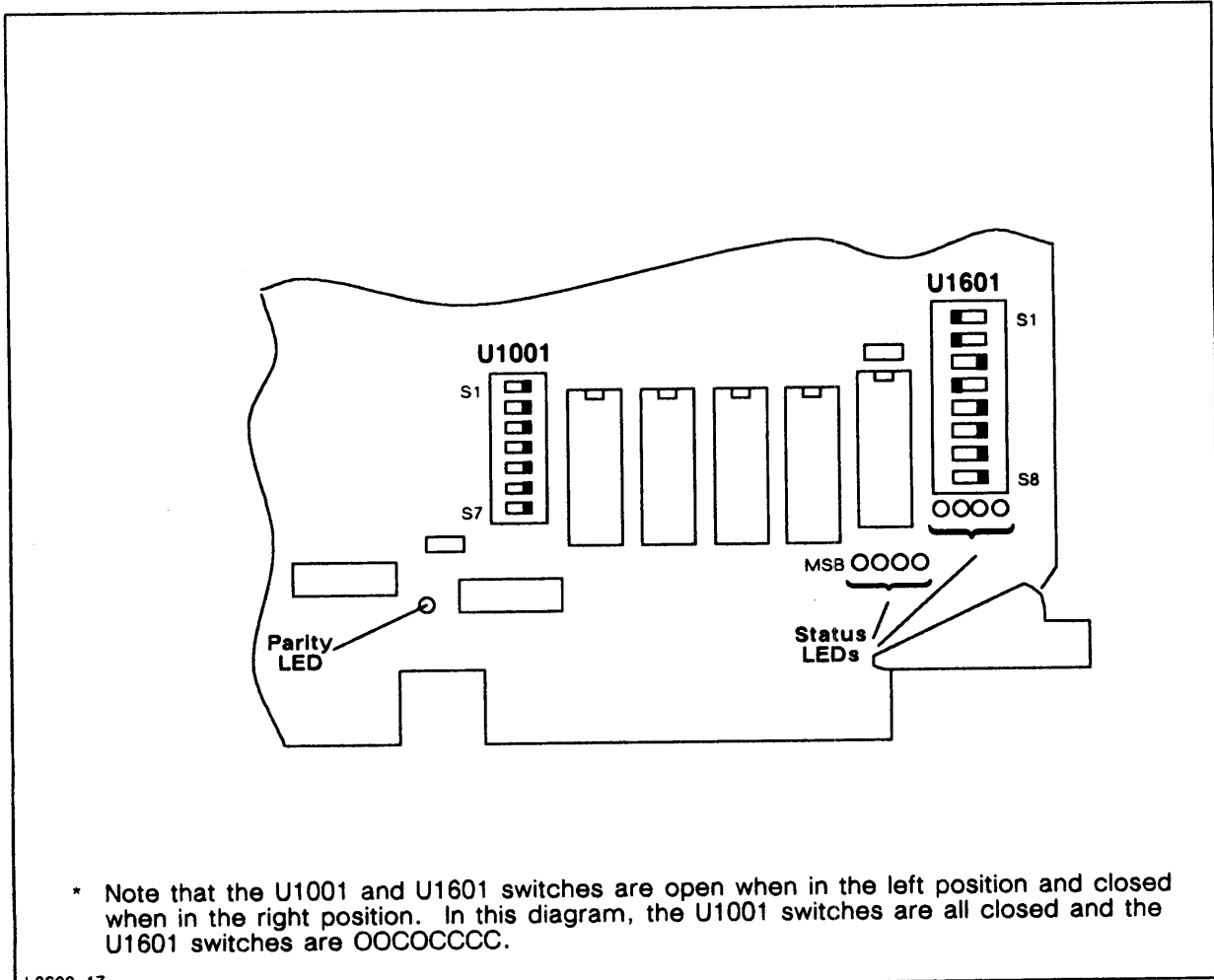
* Note that the U1001 and U1601 switches are open when in the left position and closed when in the right position. In this diagram, the U1001 switches are all closed and the U1601 switches are OOCOCCCC.

L8600-17

Switch		Description
U1001	S1 thru S4	Selects RS-232 or RS-422/423 on all ports. Refer to Table 2-3.
	S5 and S6	Selects RS-232 or V.28 for modem control line DSR on Ports B and C. Refer to Table 2-3.
	S7	Determines if Port A will be the VCP interface. Refer to Table 2-3.
U1601	S1 thru S6	Start-Up (BOOT SEL) – Used during normal operation to select the bootstrap source. They also control the operation of the computer while VCP Pre-Test is executing. Refer to Table 2-3.
	S7	Don't care.
	S8	Memory Lost – The Memory Lost switch is an auto-restart override switch which can be set open when battery back-up is available. Because the Micro 14 Computer does not support battery back-up, this switch must be set closed.

Table 2-3. A400 Switch Settings

CPU SWITCH SUMMARY (U1601)

*1	2	3	4	5	6	7	8	MEANING
C	C	C	C	-	-	-	-	Self-test loop
C	C	O	C	-	-	-	-	Loop, stop on error
C	O	O	C	-	-	-	-	VCP execution
O	C	C	C	-	-	-	-	*VCP execution
O	O	C	O	-	-	-	-	*VCP, speed sense
O	C	C	O	-	-	-	-	*Disc int loader
O	C	O	C	-	-	-	-	*PROM loader
O	O	C	C	-	-	-	-	*HDLC loader
O	O	O	C	-	-	-	-	*HP-IB disc ldr
-	-	-	-	O	-	-	-	Break disabled
-	-	-	-	C	-	-	-	Break enabled
-	-	-	-	-	O	-	-	ENQ/ACK disabled
-	-	-	-	-	C	-	-	ENQ/ACK enabled
-	-	-	-	-	-	-	C	#ARS disabled
-	-	-	-	-	-	-	O	#ARS enabled

OBIO SWITCH SUMMARY (U1001)

1	2	3	4	5	6	7	MEANING	
O	-	-	-	-	-	-	RS-422/423	Port A
C	-	-	-	-	-	-	RS-232	Port A
-	O	-	-	-	-	-	RS-422/423	Port B
-	C	-	-	-	-	-	RS-232	Port B
-	-	O	-	-	-	-	RS-422/423	Port C
-	-	C	-	-	-	-	RS-232	Port C
-	-	-	O	-	-	-	RS-422/423	Port D
-	-	-	C	-	-	-	RS-232	Port D
-	-	C	-	O	-	-	Modem Port C, V.28	
-	-	C	-	C	-	-	Modem Port C, RS-232	
-	C	-	-	-	O	-	Modem Port B, V.28	
-	C	-	-	-	C	-	Modem Port B, RS-232	
-	-	-	-	-	-	C	VCP break enabled	

O = Open = off;   C = Closed = on

* = Auto restart if enabled
# = Auto restart after powerfail

Table 2-4. LED Display for I/O Cards

STATUS INDICATION	L7	L6	L5	L4	L3	L2	L1	L0	Octal
NO I/O cards in system	1	0	0	0	0	0	0	0	200
More than one interface has been enabled as VCP	1	0	0	0	0	0	0	1	201
Priority chain broken	1	0	0	0	0	0	1	0	202
Duplicate select code	1	0	0	0	0	0	1	1	203
Select code = < 20B	1	0	0	0	0	1	0	0	204
Terminal not connected for VCP	1	0	0	0	0	1	0	1	205
Unexpected TBG interrupt	1	0	0	0	0	1	1	0	206
Unexpected MP interrupt	1	0	0	0	0	1	1	1	207
Unexpected UIT interrupt	1	0	0	0	1	0	0	0	210
Invalid select code for OBIO	1	0	0	0	1	0	0	1	211
Invalid ID no. for OBIO	1	0	0	0	1	0	1	0	212
Speed Sensing failure on OBIO or 12040D MUX VCP port	1	0	0	0	1	0	1	1	213

NOTE: 1 = LED on;   0 = LED off.

Table 2-4 indicates the failing subsection corresponding to the LED pattern displayed if the error does not correspond to a failing I/O interface.

When a failure occurs in the main memory test or the I/O interface test, the section indicator and the subcode indicator (as described above) alternate on the LEDs. The same patterns that are displayed on the LEDs are also displayed in the corresponding octal format on the VCP terminal console if the console exists and the information can be displayed.

Along with error information being displayed on the VCP terminal console, other useful information is displayed:

- The amount of memory found in the system
- Contents of the various registers
- A table of select codes and ID numbers for all I/O cards in the system
- CPU identification

The A-Register contains the number of I/O chips that were tested during the pre-test. The B-Register contains the revision code of the VCP PROMs unless the pre-test detects a duplicate I/O select code and displays it in the B-Register.

Bootstrap Loaders

The A400 bootstrap loaders are contained in 32k bytes of PROM space located on the processor board. The PROM also stores VCP code. The bootstrap loaders share the RAM with extra scratch pad registers, and with code for MAP and diagnostics. The bootstrap loading devices are:

- Disc drives (via HP-IB).
- PROM storage modules.
- DS/1000-IV (HDLC) network links.
- HP 264x mini-cartridge tapes.
- CS/80 cartridge tapes.
- Micro/1000 internal discs.
- 1600 bpi magnetic tapes.

The loader can be invoked by auto-boot when power comes up, or by VCP command. Auto-boot can only invoke four of the loaders: disc via HP-IB, PROM module, NS/1000 or DS/1000-IV (HDLC), and Micro/1000 internal disc. The VCP can invoke any of the loaders by a command from the operator. The VCP load commands are discussed later in this chapter.

Loader Selection for Auto-Boot

Auto-boot is selected by setting one of the four U1601 BOOT SEL switches located on the A400 board. These switches, the startup switches, are set during installation and also provide options other than auto-boot selection. When a loader has been selected for auto-boot and the self-test completes, the boot loader executes if memory was lost; or the program in memory executes if memory was sustained by the optional battery backup pack. Refer to Table 2-2 and 2-3 for startup switch settings.

Program Starts

When an auto-boot completes without error, the loaded program starts execution at memory location 02. The loader sets the contents of the A- and B-Registers as follows:

1. Cold start (memory not sustained):

   a. A = loader command parameters
   b. B = pointer to string area

2. Auto-restart (memory sustained; execution starts at location 04):

   a. A = 0
   b. B = 0

3. %E command from VCP:

   a. A = -1
   b. B = 0

4. %B command from VCP:

   a. A = loader command parameters.

   b. B = pointer to a string area where:
      Word 1 = memory size (64k blocks)
      Word 2 = string length (in bytes)
      Word 3 = first word of string
      Word n = n-2 word of string

VCP Re-Entry for Extended Boot Loading

The VCP loader can be re-entered from a program to boot load. It executes a program from a loading device. The VCP code is re-entered as follows:

1. A VCP boot loader call allows you to call any of the VCP loaders. This allows a complete call back sequence including a checkout routine. The following is a sample VCP loader call back checkout program:

LDA COUNT	Negative number of characters in the boot string
LDB POINTER	Starting address of the string
HLT 0,C	Call VCP loader sequence
<----->	VCP loader is started and the new program is loaded
COUNT DEC -12	Negative number of characters (bytes) in the string
POINTER DEF*+1	Starting address of the string ASC 06,DC2027SYSTEM

   This string can be any allowable string entered after the %B command (%Bxxffffbusctext). Note that %B is not actually entered but is assumed when using this call.

   If the VCP loader encounters an error, the loader will report the error and return to the VCP prompt.

2. With the disc loader, re-enter to boot load the specific program described by the "ABS" code in the following call back programming sequence.

CLA,CLE,INA	Indicate disc call back -- do not suspend
HLT 3,C	Enable PROM
ABS...	HP-IB bus address
ABS...	Device unit number (head for 7906)
ABS...	Absolute starting sector (Vector 1 for 7908/11/12)
ABS...	Cylinder offset (Vector 2 for 7908/11/12/14)
ABS...	Vector 3 for 7908/11/12/14

This sequence assumes that the Global Register is set prior to entry to the loader and that the absolute starting sector is the combined cylinder/head/sector for that drive. When the load is completed, the loader will start execution in the standard JMP 2 manner. If a suspend after load was specified by the E-Register being set when called, the program will halt after the load. In the case of the halt the operator can enter either a %E or a %R to continue. Any error will return to the VCP, if present, or restart the original load.

The 7906 will be accessed in the surface mode only, and all other discs will be accessed in the cylinder mode.

Device Parameters and Media Formats

There is a specific data format for each combination of loader, interface card, loading device, and media. The data formats are described in Figure 2-1.

Virtual Control Panel (VCP)

The VCP program is an interactive program that enables an external device (such as a terminal) to control the CPU in a manner similar to a conventional computer control panel. That is, it allows the operator to load programs using the loaders, access the various registers (for example, A, B, P, and I/O registers), examine or change memory, and control execution of a program.

The VCP program supports the following VCP interfaces:

- HP 12005A/B Async Serial Interface card
- HP 12007/12044 DS/1000-IV card
- Port A of the On-board I/O (OBIO)
- HP 12040B/C/D MUX card

The select code for OBIO is set at 77B and cannot be changed. The select code for the other interfaces does not matter. Only one interface card in the computer can serve as a VCP interface; the card selection is established by selecting a switch on the desired I/O card during system installation. To use OBIO Port A to connect to VCP, set the OBIO switch (U1001-7) to "on" (closed).

VCP Program Operation

The VCP program is executed from PROM as a software program and uses the various machine registers (such as A and B) during its execution. Therefore, these registers are automatically saved upon entry to the VCP code. (The save area is in boot RAM on the memory controller section of the A400 board.) Thus, the response to an inquiry is the data that was saved at the time of entry to the program. The exceptions to this are indicated by the absence of an asterisk in Table 2-5. When you enter the Run (%R) command, the VCP program restores the machine with the current data in the save area and starts execution as specified by the program execution address in the P-Register.

The VCP program can be entered in the following three ways:

1. After a power-up, PROM execution is directed to the VCP program instead of a boot load routine;

2. When the VCP interface card requests a slave cycle to enable the VCP program (for example, BREAK key pressed on VCP); or

3. When a HLT (halt) instruction is fetched and one I/O card is enabled for break (otherwise, the instruction has no effect).

After a power-up, the computer type, the I/O table, and the amount of memory are displayed on the VCP screen. The A-Register is set to the number of I/O chips that were tested during the self-test. This enables you to verify that all installed memory and I/O cards were tested. (Also, except when the self-test detects a duplicate I/O select code and reports it in the B-Register, the B-Register contains the revision code of the VCP PROMs.) When entered, the VCP displays the basic set of registers (P, A, B, RW, M, and T) and issues the VCP prompt character (VCP>) for a response. You can enter any of the characters or commands listed in Tables 2-5 and 2-6 and the VCP program responds as indicated in the tables. A carriage return is used to terminate a VCP entry.

After a response to an inquiry, you can change the data contained in that register or memory location by entering new data. For example (operator inputs are underlined and <cr> indicates a carriage return):

        A 001234 *4321*<cr>
        A 004321

Data input is terminated by a carriage return. If during an input, the program cannot interpret a character, the program displays the characters "!?", then starts a new line with the VCP prompt. Entry errors may be corrected by backspacing over them and entering the correct information. While entering data, you can abort the input by entering a rub-out (DEL). The loader commands, %B, %L, and %W can also be aborted by a rub-out. When entering data into a register, leading zeros can be omitted. If you type a question mark, the VCP will output a "help" file that summarizes acceptable command entries.

MINI-CARTRIDGE TAPE

Device: HP 264x Terminal

Interface: HP 12005A Asynchronous Serial Interface

Default
Parameters*: 000020

Format: Reads absolute binary file, writes 4k absolute binary block.

Loader: Transmits special escape sequence to invoke a read of a record and does checksum of the data. When writing to tape, a block number is used to specify which 4k-word memory area is to be dumped to tape (0 = 0 to 4k.

If a file number is specified then the program will issue a find file command; if not, the tape is read from where it stands. When writing to the tape, the program will not write a file mark; this allows sequential blocks to be written in a series. There are only two units (0 and 1) on the terminal; if a larger unit number is specified, the result will be unpredictable.

More than 32k words may be loaded into a system from a single cartridge tape.

PROM MODULE

Device: PROM (2k x 8 bits)

Interface: HP 12008A PROM Storage Module

Default
Parameters*: 000022

Format: Count-Partial-Data
Count = number of 64k byte blocks.
Partial = number of words of partial 64k byte block.
Data = 16-bit words, one word per location until Count and Partial are satisfied.

Loader: Uses STC-LIA process to transfer data. The PROM cannot be written to nor does it use the block number of unit number.

DISC DRIVE

Device: HP 9895, CS/80 and SS/80 Disc Drive, cartridge tape drive of the 7908/11/12/14 Disc Drive, or 2434A/84A/B integrated disc using HP-IB interface.

Interface: HP 12009A HP-IB Interface

Default
Parameters*: 002027

Format: Count-Partial-Data
Count† = number of 64k byte blocks.
Count† = number of 64k byte blocks.
Data = 16-bit words, one word per loction until Count and Partial are satisfied.

Loader: Uses HP-IB protocol to communicate with the disc. The load sequence is:

1. Device clear
2. Status check
3. Read/write 32 words via DMA
4. Status check

---

* See Figure 2-2 for loader command formats.
† The Count and Partial values are stored in memory locations 00000 and 00001, respectively.

Figure 2-1. Loading Device Parameters and Media Formats

DISC DRIVE (VIA DISC INTERFACE)

Device: HP 2434A/84A/B Internal fixed/micro-floppy disc drive.

Interface: HP 12022A Disc Interface.

Default
Parameters*: 000032

Format: Same as Disc Drive via HP-IB, above.

Loader: Standard I/O for commands to Interface, and DMA for data.

MAGNETIC TAPE

Device: HP 7970/7974/7978/7979/7980 Magnetic Tape Drive

Interface: HP 12009A HP-IB Interface

Default
Parameters*: 004027

Format: Memory Image file
Count-Partial-Data
Count = number of 64k byte blocks.
Partial = number of words of partial 64k byte block.
Data = 256 byte records read until EOF or until Count and Partial are satisfied.

Loader: Uses HP-IB protocol to communicate with the disc. The load sequence is:

       1. Device ID
       2. Status clear
       3. Rewind/file forward (if file specified)
       4. Read/write
       5. Status check

COMPUTER NETWORK

Device: HP 1000 Computer.

Interface: HP 12007B/12044A HDLC Interface.

Default
Parameters*: 000024

Format: Reads absolute binary or memory image files, writes a 32k memory image file.

Loader: Standard handshake using HP distributed system protocol. Block number and unit number are not used.

---

* See Figure 2-2 for loader command formats.
† The Count and Partial values are stored in memory locations 00000 and 00001, respectively.

Figure 2-1. Loading Device Parameters and Media Formats (Continued)

Table 2-5. VCP Characters and Associated Registers

CHARACTER ENTERED	RESPONSE †	MEANING
A*	xxxxx	A-Register contents
B*	xxxxx	B-Register contents
E*	x	E-Register contents
G*	x000xx	Global Register (GR) contents and status (bit 15 = 0 if enabled, 1 if disabled)
I*	x	Interrupt system status (0=off, 1=on)
M*	0xxxxx	Memory address (pointer for T and Ln command)
O*	x	O-Register contents
P*	xxxxx	Program execution address
Q*	xxxxx	C- and Q-Register contents (C is bit 15)
RS	xxxxx	Switch register contents
T	0xxxxx xxxxx	Memory contents pointed to by M
V	xxxxx	Violation register (memory protect)
X*	xxxxx	X-Register contents
Y*	xxxxx	Y-Register contents
Z*	xxxxx	Z-Register contents
RC	xxxxx	Central Interrupt Register contents
RD**	xxxxx xxxxx	Data for I/O diagnose modes 1 and 2
RF**	xxxxx	I/O flags: Flags 20 thru 24, and Flag 30 (1=flag set; 0=flag clear)
RI**	xxxxx	Interrupt mask register
RP	xxxx xxxx xxx	Parity violation register contents
RS	xxxxx	Switch Register
RW*	xxxxx	Working map set (WMAP)
R20**	xxxxx	DMA self-configuration register
R21**	xxxxx	DMA control register
R22**	xxxxx	DMA address register
R23**	xxxxx	DMA count register
R24**	xxxxx	I/O scratch register
R25**	xxxxx	I/O scratch register
R26**	xxxxx	I/O scratch register
R30**	xxxxx	I/O card data register
R31**	xxxxx	Optional I/O card register
R32**	xxxxx	Optional I/O card register
?	xxxxx	Output Help file

† x = octal data.

* Registers that are maintained in the VCP save area of boot RAM.

** Applies only to the I/O card whose select code equals the contents of the Global Register.

NOTE: When a register's contents are changed by the user, the new value is returned; if the VCP does not accept a change, the VCP prompt is returned.

Table 2-6. VCP Commands

COMMAND*	MEANING
%B	Load and go (boot). Execute a specified loader routine and start program execution at completion of load. See Figure 2-2 for format.
%C	Clear memory. Set all memory to zero and perform a preset.
%E	Execute. Start execution of program at location P=2 (A-Register equals -1 (all ones) and B-Register equals 0).
%L	Load. Similiar to %B except do not start execution. See Figure 2-2 for format. (%L followed by %R is equivalent to %B).
%M	Memory test. Execute destructive extended memory test. Tests addressing logic. The test will optionally loop on error. Returns amount of memory found. If an error is found, the error type, error address, and the data written and read is displayed.
%P	Preset. Generate a control reset (CRS) signal on the backplane to initialize all cards
%R	Run. Set all registers to the appropriate values in the save area and start execution at address specified by the P-Register.
%S	Parity error set. Places a parity error in addressable memory to test the parity interrupt handler and to verify proper functioning of the parity error interrupt logic.
%T	Test. Initiate the self-test and return to VCP (memory is sustained but the I/O system is reset).
%W	Write. Write to the selected device. (See Figure 2-2 for format.) When writing to a disc drive, the Count and Partial values defined in Figure 2-1 must be in memory locations 00000 and 00001.
D	Decrement. Decrement memory pointer and display the contents of the M- and T-Registers. Valid only after T.
Ln	List. List n blocks of eight memory locations starting with location pointed to by the M-Register.
N	Next. Same as D except increment the pointer. Valid only after T.
RMxx	List the 32 map registers in the DMS map set specified by xx.
RMxxPyy	Show the value of register yy in map set xx. If a number is input after this command,the register is changed to the new value.
?	Output Help file.
* Must be followed by a carriage return.	

Extended Memory Test (%M)

The extended memory test (VCP %M command) is a three-to-four-minute test that reads all the memory found in the system to check for parity errors, tests the functionality of the address logic, and checks for stuck bits in the data.

CAUTION

The extended memory test (%M) is a destructive memory test. For this reason, you will be asked twice whether you really want to execute the test after you issue the %M command. The only valid responses to the questions are Y(yes) or N(no). If any other responses are given, the question will be repeated. If N, control will pass back to the VCP prompt. If Y, the test will go to the next step.

After verifying that you want to execute the destructive test, you are asked if you want to loop if an error is found. The default is N, no looping. If the response is Y, looping will be activated. After an error is displayed, you have the option of continuing with testing or returning to VCP.

When command execution is initiated, the amount of memory in the system is determined and displayed on the VCP console. All of the memory is read, and if a parity error occurs, an error message is displayed. Note that in this section of the test, the data-written that is displayed when an error occurs is invalid.

Error messages are displayed for parity errors, addressing errors, or pattern test errors at the time they occur. If looping has been activated, the testing is repeated.

If looping has not been activated, you will be asked if you want to continue. If you enter N, control will pass to the VCP prompt. If you enter Y, testing will continue if the failure occurred before the pattern test. If you are in the pattern test, a Y reply will cause testing to begin on the next pattern.

If there are no errors detected, a message indicating successful completion of the extended memory test will be displayed. The current value of some of the registers will then be displayed and control will be passed to the VCP prompt.

Address Test

The address test executes next, to determine if the addressing logic is functioning properly. A message will notify you when pass 1 and pass 2 are being executed. Each location will then be read back and verified to contain the unique pattern that was written into it. If the patterns don't match, an addressing error will occur. A message will also indicate successful completion of the test.

Pattern Test for Stuck Bits

Five patterns are run to test for stuck bits, one 32k-word block at a time. If the data pattern written does not match the data pattern read, a pattern test error occurs.

When all of the 32k-word blocks have been tested successfully with all five data patterns, a message is displayed indicating that the pattern has completed successfully.

The following is a sample error message displayed when an error is detected in the extended memory test:

Address Test Failure

Page number with error = 000044
Offset into page with error = 000000
Logical address with error = 010000
Data Written = 010000 *
Data Read   = 000002

* Not valid before the first "Addr. Test in Progress – Pass 1" message is displayed.

Parity Error Set Routine (%S)

The parity error set routine (VCP %S command) allows you to place a parity error somewhere in addressable memory in order to test the parity error interrupt handler in the extended memory test and the VCP pre-test. It also can be used to quickly verify that the parity error interrupt logic is functioning properly.

Throughout this routine, when you are prompted for a number, you must respond with a valid octal number; otherwise, the prompt is repeated or the invalid input will be ignored.

When %S is entered at the VCP prompt, a message is displayed indicating what this command will do. VCP will prompt you for the page number in physical memory where you want to place the error. If a valid octal number is not entered in the first digit of the number, the question will be repeated. If there is an invalid number in any other digit location, the number is ignored.

When a valid page number is entered, you will be prompted for the offset address of the location to be changed. The logical address where the error is placed will then be displayed.

After the requested location has been read and the contents displayed, you are prompted for new data to be written to that location. This new data should have one or more bits toggled to cause a single or double-bit error. You will be asked whether the data just entered is okay. If you enter N, the new data entered will not be written and the old data will be displayed again, so that another value can be entered.

If you enter Y, the parity error sense is reversed, the new data is written, parity sense is restored, and parity error interrupts are turned on.

To verify that the parity interrupt circuitry is working, you must read the location you put in by using the %M command, the L command with WMAP set properly, or manually using the M and T-Registers.

To clear the parity error, the memory location with the error must be written to, without changing the parity sense. This can be accomplished by using the %T, %C, or other standard VCP commands. See Table 2-6.

Loader Commands

The loader commands can be entered via the VCP in either of two ways:

1. Allow the parameter default values (given in Figure 2-1) to be used; or

2. Specify all necessary parameters.

The VCP loader command format is shown in Figure 2-2. The loader command error codes and their meanings are listed in Table 2-7.

LOADER COMMAND FORMAT

%B/L/W dv fffffbusc text

where:

dv	=	device type as follows:

DC = disc (cartridge or flexible) via HP-IB
CT = cartridge tape (HP 264x)
RM = PROM card
DS = DS computer network Link
MT = magnetic tape via HP-IB
DI = disc via HP 12022A Card

fffff	=	file number (octal 0 to 77777 only)
b	=	4k-word memory block number when writing to cartridge tape; HP-IB bus address of disc drive; or non-HP-IB drive address; otherwise, use 0. For the HP 2437A/87A internal disc drives, this is 0 for the first fixed drive, 1 for the second, and 3 for the micro-floppy drives.
u	=	unit number (0 to 7) only if used on device. For the HP 7906 Disc Drive, the unit number is the head number. For CS/80 Disc Drive that includes cartridge tape drive, unit 0 = disc drive and unit 1 = cartridge tape drive.
sc	=	select code of interface card to be used.
text	=	file name, or ASCII string to be passed to the program after it is loaded. This is only available with the %B and %L commands.

Note: See Figure 2-1 for default parameters for each loading device.

Note that spaces cannot be used in the command entry. The following formats are all acceptable:

%Bdvtext	Device parameters are defaulted; text cannot start with a number.
%Bdffbusc	No text passed.
%Bdvffbusctext	Text passed.

EXAMPLES:

%BDC	Load and start execution of the default program on disc. (Disc parameters defaulted to 002027; see Figure 2-1).
%BDC30	Load and start execution of the default program on the disc at select code 30 and default other parameters.
%LDC27025	Load (but don't execute) and override parameter default values:

file number 2 (i.e., the third file)
HP-IB bus address 7
unit 0
select code 25

%WDC27025	Same as above except write to file 2.

Figure 2-2. Loader Command Format

Table 2-7. VCP Loader Command Errors

ERROR CODE	MEANING	ERROR CODE	MEANING
		Magnetic Tape Loader Errors	
0	Unrecognizable load/bootstring.		
2	Select code less than 20 octal.	510	Time out during initialization/read ID.
3	No card with the select code you specified.	511	Time out when issuing end/select unit.
		512	Mag tape off line.
		513	No write ring.
Cartridge Tape Loader Errors		514	Time out during End command.
		515	Time out waiting for rewind completion.
		517	Time out waiting for DMA transfer.
110	File forward error. Status in B-Register.	520	Parity error during DMA transfer.
111	Checksum error.	521	Time out doing a PHI flush.
112	No data before EOF (end of file).	522	Time out waiting for DSJ.
120	Write error. Status in B-Register.	523	Bad DSJ response.
		525	Time out waiting for Mag Tape Not Busy.
		530	Time out after issuing a command.
PROM Module Loader Errors		531	Parallel Poll time out after issuing a command
		535	Bad status after read/write command.
211	End of programs.	550	No data transfer (read only).
212	Bad format.	560	No mag tape ID.
213	System larger than 32k must start on boundary.		
214	Write not allowed to ROM.	**HP 12022A Disc Interface Loader Errors**	
		610	Time out after SDH (sector drive head) for read/write.
DS/1000 Loader Errors		611	Time out after cylinder high.
		612	Time out after cylinder low.
		613	Time out after sector.
310	Time out after CLC 0. Check select code specified.	614	Time out after sector count.
		615	Time out after read/write command.
311	Checksum error. P file not absolute binary.	616	Time out after DMA read/write transfer.
312	Time out after download request.	617	Parity error during transfer.
313	Time out after file number.	620	Fixed disc not ready.
314	Bad transfer (Central generated). Status in B-Register.	630	Time out after request status register.
		631	Time out after read status register.
315	Time out after buffer request.	632	Time out after waiting for not busy.
316	Time out after count echo.	633	Time out after request error register.
317	Time out waiting for data.	634	Time out after read error register.
320	Time out after VCP mode requests a DS write.	635	Status error: A-Register = status register; B-Register = error register.
321	Central will not accept data. Status in B-Register.	650	Time out after SDH register for restore.
325	Data block out of sequence.	651	Time out after restore.
		660	Disc not defined.
Disc Loader Errors (via HP-IB)		**Other**	
		1024/ 1025	Possible meanings:
411	Time out reading disc type. Check HP-IB address.		1. Booting from CS/30 disc that has been push button restored from CTD tape or booting diagnostics directly from the tape. The CTD tape may not have certified/formatted before data was stored to it.
412	Time out UDC (Universal Device Code) or reading status. Check disc.		
413	Status error. Status in B-Register.		
414	Time out during file mask.		
415	Time out during seek.		2. Booting from a CTD tape in ASAVE format.
416	Time out during read or write command.		
417	Time out during DMA of data.		3. Booting from the CS/80 disc was not successful. Bootex may be corrupted.
420	Parity error during DMA transfer.		
421	Time out during FIFO flush.		
422	Time out during DSJ (Device Specified Jump) command.		4. Faulty tape control board in the CS/80 disc.
423	Bad DSJ return. Returned value in B-Register.		
460	Disc not identifiable. Disc ID in B-Register.		5. Incorrect VCP file number in the runstring.

VCP User Considerations

When using the VCP to debug a program, you should be aware of the following conditions:

1. The VCP program uses an interface card or port A of the on-board I/O and modifies the characteristics of that interface. When the VCP program exits, it sets register 24 on the interface to all "1"'s to allow software detection of a VCP interaction and, thus, reinitialization of an operation. (This also causes an interrupt if the interrupt system is enabled.) The VCP will leave the interface in output mode with both Flag 30 and Control 30 set.

2. The status of the interrupt system (STC 4 (on) or CLC 4 (off)) is not indicated and will remain unchanged unless %P is executed to preset the computer.

3. Memory protect is indicated by the sign bit of RW (WMAP-Register) and may be modified.

3

Programming Information

This chapter describes the software data formats and the base set machine-language instruction coding (including single and double-precision floating point, virtual memory, high-level language support instruction set (LIS), and operating system instruction set (OSI) required to operate the computer and its associated input/output system. Refer to the "Dynamic Mapping System" and "Code and Data Separation" chapters for information on machine-language coding.

Data Formats

As shown in Figure 3-1, the basic data format is a 16-bit word in which bit positions are numbered from 0 through 15 in order of increasing significance. Bit position 15 of the data format is used for the sign bit; a logic 0 in this position indicates a positive number and a logic 1 in this position indicates a negative number. The data is assumed to be a whole number and the binary point is therefore assumed to be to the right of the number.

The basic word can also be divided into two 8-bit bytes or combined to form a 32-bit double word. The byte format is used for character-oriented input/output devices; packing two bytes of data into one 16-bit word is accomplished by software drivers or by byte-packing hardware in the I/O Master. In I/O operations, the higher-order byte (byte 1) is the first to be transferred.

The double-integer format is used for extended arithmetic in conjunction with the extended arithmetic instructions. Bit position 15 of the most-significant word is the sign bit and the binary point is assumed to be to the right of the least significant word. The integer value is expressed by the remaining 31 bits.

Two floating point formats are shown in Figure 3-1. The single-precision format is used with single-precision floating point instructions included in the standard base set of instructions. The double-precision format is used with double-precision floating point instructions. Bit position 15 of the most-significant word is the mantissa sign and bit position 0 of the least-significant word is the exponent sign. Bits 1 through 7 of the least significant word express the exponent and the remaining bits express the mantissa. A single-precision floating point number is made up of a 23-bit mantissa (fraction) and sign and a 7-bit exponent and sign, thus providing six significant decimal digits in the mantissa. A double-precision floating point number is made up of a 55-bit mantissa and a 7-bit exponent and sign, thus providing 16 significant decimal digits in the mantissa. If either the mantissa or the exponent is negative, that part must be stored in two's complement form. The number must be in the approximate range of 10^{-38} to 10^{+38}. When loaded into the accumulators, the A-Register contains the most-significant word and the B-Register contains the least-significant word.

Figure 3-1 also illustrates the octal notation for both single-length (16-bit) and double-length (32-bit) words. Each group of three bits, beginning at the right, is combined to form an octal digit. A single-length (16-bit) word can therefore be fully expressed by six octal digits and a double-length (32-bit) word can be fully expressed by 11 octal digits. Octal notation is not shown for byte or floating-point formats, since bytes normally represent characters and floating-point numbers are given in decimal form.

The range of representable numbers for single-word data is +32,767 to -32,768 (decimal) or +77,777 to -100,000 (octal). The range of representable numbers for double-word integer data is +2,147,483,647 to -2,147,483,648 (decimal) or +17,777,777,777 to -20,000,000,000 (octal).

Addressing

Paging

The computer memory is logically divided into pages of 1,024 words each. A page is defined as the largest block of memory that can be directly addressed by the address bits of a single-length memory reference instruction. These memory reference instructions use 10 bits (bits 0 through 9) to specify a memory address; thus, the page size is 1,024 locations (2000 octal). Octal addresses for each page, up to a maximum memory size of 32k, are listed in Table 3-1.

DATA FORMATS

- - - > INCREASING MEMORY - - - >

SINGLE INTEGER

Sign Bit — Least significant data bit

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 — Binary point

PACKED BYTE FORMAT

Byte 0   Byte 1

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

DOUBLE INTEGER

Sign Bit — Binary point

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0   15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Integer
31 bits

SINGLE PRECISION FLOATING POINT

Mantissa sign — Exponent sign

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0   15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Binary Point   Mantissa 23 bits   Exponent 7 bits

DOUBLE PRECISION FLOATING POINT (OPTIONAL)

Mantissa sign — Exponent sign

15 14 13   1 0   15 14   2 1 0   15 14   2 1 0   15 14   8 7 6 5 4 3 2 1 0

Binary Point   Mantissa 55 bits   Exponent 7 bits

OCTAL NOTATION

WORD FORMAT

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

8^5 8^4 8^3 8^2 8^1 8^0

INTEGER DOUBLE WORD

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0   15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

8^{10} 8^9 8^8 8^7 8^6 8^5 8^4 8^3 8^2 8^1 8^0

Figure 3-1. Data Formats and Octal Notation

L8600-37

Provision is made to directly address one of two pages: page zero (the base page consisting of locations 00000 through 01777) and the current page (the page in which the instruction itself is located). Memory reference instructions reserve bit 10 to specify one or the other of these two pages. To address locations on any other page, indirect addressing is used as described in the following paragraphs. Page references are specified by bit 10 as follows:

1. Logic 0 = Page Zero (Z)

2. Logic 1 = Current Page (C)

Table 3-1. Memory Paging

MEMORY SIZE	PAGE	OCTAL ADDRESSES		
	0	00000	to	01777
	1	02000	to	03777
	2	04000	to	05777
	3	06000	to	07777
	4	10000	to	11777
	5	12000	to	13777
	6	14000	to	15777
	7	16000	to	17777
	8	20000	to	21777
	9	22000	to	23777
	10	24000	to	25777
	11	26000	to	27777
	12	30000	to	31777
	13	32000	to	33777
	14	34000	to	35777
16K ↓	15	36000	to	37777
	16	40000	to	41777
	17	42000	to	43777
	18	44000	to	45777
	19	46000	to	47777
	20	50000	to	51777
	21	52000	to	53777
	22	54000	to	55777
	23	56000	to	57777
	24	60000	to	61777
	25	62000	to	63777
	26	64000	to	65777
	27	66000	to	67777
	28	70000	to	71777
	29	72000	to	73777
	30	74000	to	75777
	31	76000	to	77777

Direct and Indirect Addressing

All memory reference instructions reserve bit 15 to specify either direct or indirect addressing. For single-length memory reference instructions, bit 15 of the instruction word is used; for extended arithmetic memory reference instructions, bit 15 of the address word is used. Indirect addressing uses the address part of the instruction to access another word in memory, which is taken as the new memory reference for the same instruction. This new address word is a full 16 bits long: 15 address bits plus another direct/indirect bit. The 15-bit length of the address permits access to any location in the Logical address space. If bit 15 again specifies indirect addressing, still another address is obtained; thus, multistep indirect addressing may be done to any number of levels. The first address obtained that does not specify another indirect level becomes the effective address for the instruction. Direct or indirect addressing is specified by bit 15 as follows:

1. Logic 0 = Direct (D)

2. Logic 1 = Indirect (I)

After three or more levels of indirect addressing, interrupts are checked and, if an interrupt is pending, the instruction will be interrupted and restarted when the interrupt service routine is done.

Memory Mapping

Memory mapping is a standard feature of the A400 computer and is used to access all locations of main memory. Refer to the Dynamic Mapping System chapter for a description of memory mapping.

Virtual Memory Area

Under Virtual Memory Area (VMA) operation, a program may access two separate data areas, one being the 32k word logical address space, and the other being a virtual address space of up to 16M words. The virtual address space may be either memory-resident or disc-resident, and up to 1M words per program may reside in memory. This is accomplished through mapping pages of the logical address space to the virtual address space.

Code and Data Separation

When Code and Data Separation (CDS) is enabled, a program's address space is partitioned into two separate address spaces: a code space and a data space of up to 31k words each. Opcodes and the operand pointers that follow the opcode reside in code space, and variables and constants reside in data space. CDS instructions are provided that remap the code segment to other physical pages in memory, thus providing large program support. A program's code size may be up to 128 segments (each having 31k words of code), which may be either memory-resident or disc-resident. The optional HP 92078A package for the RTE-A operating system provides software support for CDS.

Base-Relative Addressing

Under CDS, special hardware is used to access memory locations relative to a base register called the Q-Register. When a memory address is in the range 2 through 1023, the Q-Register value is added to produce an effective address in the data space. When CDS is enabled, code may not reside on the base page, which means that jump instructions may not jump to the base page.

Reserved Memory Locations

The first 64 memory locations of the physical base page (octal addresses 00000 through 00077) are reserved as listed in Table 3-2. The first two locations are reserved as addresses for the two 16-bit accumulators (the A- and B-Registers). If options or input/output devices corresponding to locations 00020 through 00077 are not included in the system configuration, these locations can be used for programming purposes. The last 64 locations of the physical base page (1700 to 1777) are reserved for use by the Virtual Control Panel program for the string area.

Table 3-2. Reserved Memory Locations

MEMORY LOCATION	PURPOSE
00000	A-Register address.
00001	B-Register address.
00002-00003	Reserved.
00004	Power-fail interrupt.
00005	Memory parity interrupt.
00006	Time base generator interrupt.
00007	Memory protect interrupt.
00010	Unimplemented instruction interrupt.
00011	Reserved.
00012	Virtual Area Memory Interrupt.
00013	CDS Segment Interrupt.
00014-00017	Reserved.
00020-00077	Interrupt locations corresponding to interface card selection codes.
01700-01777	VCP program string area.

Nonexistent Memory

Nonexistent memory is defined as those locations not physically implemented in the machine. Any attempt to write into a nonexistent memory location will be ignored (no operation). Any attempt to read from a nonexistent memory location will return an all-ones word (177777 octal); no parity error occurs. If the nonexistent memory is protected, a memory protect interrupt will be generated.

Base Set Instruction Formats

The base set of instructions are classified according to format. The six formats used are illustrated in Figure 3-2 and described in the following paragraphs except for the DMS and CDS instructions, which are described in Sections IV and V. In all cases where a single bit is used to select one of two cases (for example, D/I), the choice is made by coding a logic 0 or logic 1, respectively.

Memory Reference Instructions

This class of instructions, which combines an instruction code and a memory address into one 16-bit word, is used to execute some function involving data in a specific memory location. Examples are storing, retrieving, and combining memory data to and from the accumulators (A- and B-Registers) or causing the program to jump to a specified location in memory.

The memory cell referenced (that is, the absolute address) is determined by a combination of 10 memory address bits (0 through 9) in the instruction word and 5 bits (10 through 14) assumed from the current contents of the M-Register. This means that memory reference instructions can directly address any word in the current page; additionally, if the instruction is given in some location other than the base page (page zero), bit 10 (Z/C) of the instruction doubles the addressing range to 2,048 locations by allowing the selection of either page zero or the current page. (This causes bits 10 through 14 of the address contained in the M-Register to be set to zero instead of assuming the current contents of the M-Register.) This feature provides a convenient linkage between all pages of memory, since page zero can be reached directly from any other page.

With CDS enabled, this feature becomes even more powerful as the base register is added to all base page references (addresses from 2 to 1777 octal, or MRG instructions with Z/C=0). This means that each single-word instruction has direct access to data on the current page, or data up to 1k word relative to the base register.
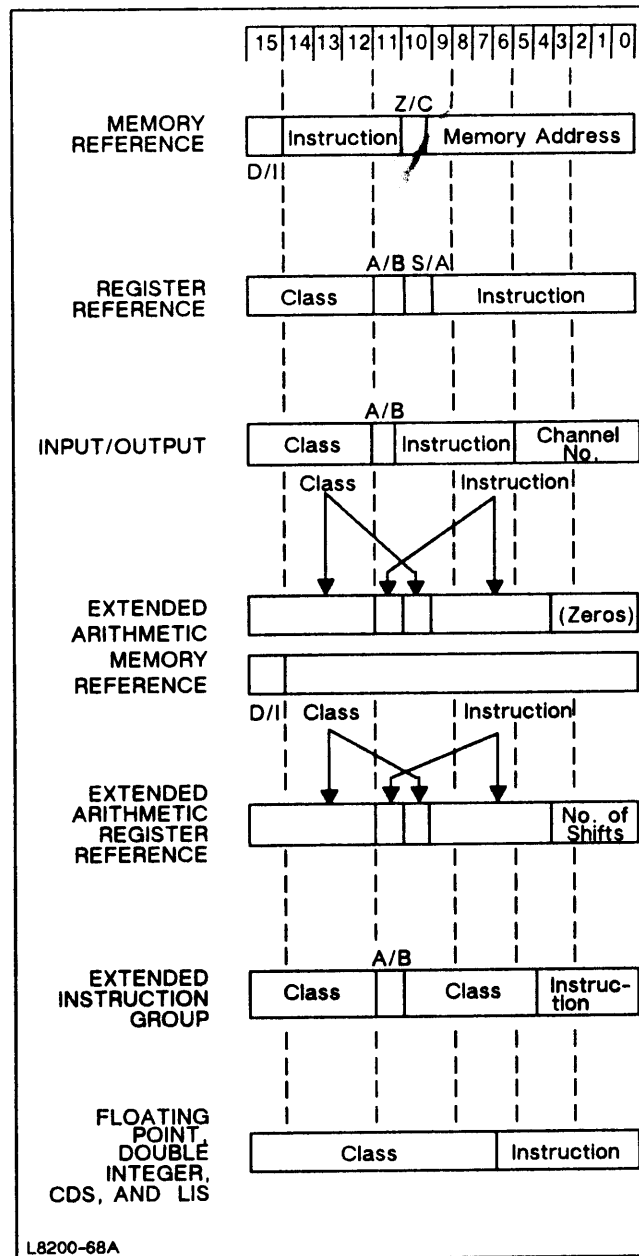
Figure 3-2. Base Set Instruction Formats

As discussed under previously, bit 15 is used to specify direct or indirect memory addressing. Note also that since the A- and B-Registers are addressable, any single-word memory reference instruction can apply to either of these Registers as well as to memory cells. For example, an ADA 0001 instruction adds the contents of the B-Register (address 0001) to the contents currently held in the A-Register; specify page zero for these operations since the addresses of the A- and B-Registers are on page zero.

Register Reference Instructions

In general, the register reference instructions manipulate bits in the A-Register, B-Register, and E-Register; there is no reference to memory. This group includes 39 basic instructions which may be combined to form a one-word multiple instruction that can operate in various ways on the contents of the A-, B-, and E-Registers. These 39 instructions are divided into two subgroups; the shift/rotate group (SRG) and the alter/skip group (ASG). The appropriate subgroup is specified by bit 10 (S/A). Typical operations are clear and/or complement a register, conditional skips, and register increment.

Input/Output Instructions

The input/output instructions use bits 6 through 11 for a variety of I/O instructions and bits 0 through 5 to apply the instructions to a specific I/O channel (if the Global Register is disabled) or to an I/O card register. This provides the means of controlling all peripherals connected to the I/O channels and for transferring data to and from these peripherals. Included also in this group are instructions to control the interrupt system, overflow bit, and computer halt.

Extended Arithmetic Memory Reference Instructions

As the single-word memory reference instruction described previously, the extended arithmetic memory reference instructions include an instruction code and a memory address. In this case, however, two words are required. The first word specifies the extended arithmetic class (bits 12 through 15 and 10) and the instruction code (bits 4 through 9 and 11); bits 0 through 3 are not needed and are coded with zeros. The second word specifies the memory address of the operand. Since the full 15 bits are used for the address, this type of instruction may directly address any location in memory. If the CDS mode is enabled and the reference is to the base page, the base (Q) register will be added to the second word before referencing memory. As with all memory reference instructions, bit 15 is used to specify direct or indirect addressing. Operations performed by this class of instructions are integer multiply and divide (using double-length product and dividend) and double load and double store.

Extended Arithmetic Register Reference Instructions

This class of instructions provides long shifts and rotates on the combined contents of the A- and B-Registers. Bits 12 through 15 and 10 identify the instruction class; bits 4 through 15 and 10 identify the instruction class; bits 4 through 9 and 11 specify the direction and type of shift; and bits 0 through 3 control the number of shifts, which can range from 1 to 16 places.

Extended Instructions

The extended instructions include index register instructions, bit and byte manipulation instructions, and move and compare instructions. Instructions comprising the extended instruction group are one, two, or three words in length. The first word is always the instruction code; operand addresses are given in the words following the instruction code or in the A- and B-Registers. The operand addresses are 15 bits long, with bit 15 (most-significant bit) generally indicating direct or indirect addressing.

Floating Point Instructions

The floating point instructions allow addition, subtraction, multiplication, and division of 32 bit floating point quantities. Two conversion routines are provided for transforming numerical integer representations to/from floating point representations. The A400 adds double precision (64-bit) floating point instructions as well as all routines to convert from single and double integer to single and double precision floating point, and vice versa.

Double Integer Instructions

The double integer instructions allow arithmetic and test operations on 32-bit quantities. Bits 15 through 7 identify the instruction class, and bits 6 through 0 specify the instruction code. Double integer values contained in the A- and B-Registers have the most significant bits in the A-Register.

Language Instruction Set

The language instruction set performs several frequently used high-level language operations, including parameter passing, array address calculations, and floating point conversion, packing, rounding and normalizing. Bits 15 through 7 identify the instruction class, and bits 6 through 0 specify the instruction code.

Virtual Memory Instructions

The virtual memory instructions perform accesses to virtual memory and the extended memory area, which are extensions of logical memory.

Operating System Instructions

The operating system instructions provide instructions for ascertaining the CPU and firmware identification, and instructions for interrupt conditions.

CDS Instructions

The A400 includes the CDS instruction set, which includes instructions for examining and modifying the base (Q) register, bounds (Z) register, and CDS-mode (C) register. This set also includes instructions for transferring control between subroutines (which may or may not be memory-resident).

All instructions that reference multi-word data (double integer, single and double precision floating point) as well as instructions using sequential addressing (DMS move instructions, .SETP and SFB) will have the base register added to the initial address if the instruction is base relative and CDS mode is enabled. Subsequent memory references are then executed sequentially.

Instructions that leave an address in a register upon completion (for example, LBT, .ZFER, .SETP, MW00) will contain an address revolved for base relativity, incremented by the proper count.

Base Set Instruction Coding

Machine language coding for the base set of instructions are provided in following paragraphs. Definitions for these instructions are grouped according to the instruction type: memory reference, register reference, input/output, extended arithmetic memory reference, and extended arithmetic register reference.

Directly above each definition is a diagram showing the machine language coding for that instruction. The darker shaded ▓ bits code the instruction type and the lighter shaded ▭ bits code the specified instruction. Unshaded bits are further defined in the introduction to each instruction type. The mnemonic code and instruction name are included above each diagram.

In all cases where an additional bit is used to specify a secondary function (D/I,Z/C, or H/C), the choice is made by coding a logic 0 or logic 1, respectively. In other words, a logic 0 codes D (direct addressing), Z (zero page, or H (hold flag); a logic 1 codes I (indirect addressing), C (current page), or C(clear flag).

Memory Reference Instructions

The following 14 memory reference instructions execute a function involving data in memory. Bits 0 through 9 specify the affected memory location on a given memory page or, if indirect addressing is specified, the next address to be referenced. Indirect addressing may be continued to any number of levels; when bit 15 (D/I) is a logical 0 (specifying direct addressing), that location will be taken as the effective address. The A- and B-Registers may be addressed as locations 00000 and 00001 (octal), respectively.

If bit 10 (Z/C) is a logic 1, the memory address is on the current page. If bit 10 is a logic 0, the memory address depends on whether CDS mode is enabled. If CDS mode is enabled, the base (Q) register will be added to bits 0 through 9 to provide the memory address. If CDS mode is not enabled, the memory address is on the base page (page 0). If the A- and B-Register is addressed, bit 10 must be a logic 0 to specify page zero, unless the current page is page zero.

ADA　　　　　　　　　　　　　　　　　　**ADD TO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	1 0 0	0 Z/C			
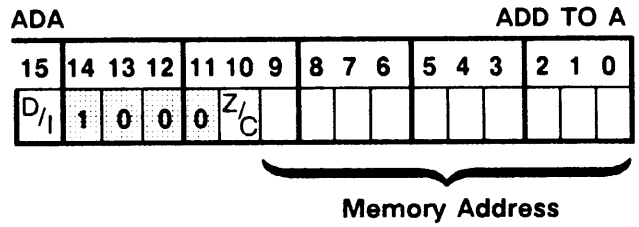
Memory Address

Adds the contents of the addressed memory location to the contents of the A-Register. The sum remains in the A-Register and the contents of the memory cell are unaltered. The result of this addition may set the extend bit or the overflow bit. (Extend and overflow examples are illustrated in the appendix.)

ADB　　　　　　　　　　　　　　　　　　**ADD TO B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	1 0 0	1 Z/C			
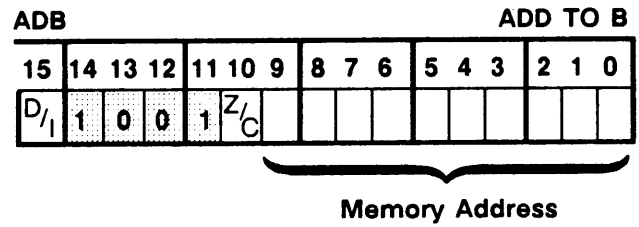
Memory Address

Adds the contents of the addressed memory location to the contents of the B-Register. The sum remains in the B-Register and the contents of the memory cell are unaltered. The result of this addition may set the extend bit or the overflow bit. (Extend and overflow examples are illustrated in the appendix.)

AND　　　　　　　　　　　　　　　　　　**"ADD" TO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	0 0 1	0 Z/C			
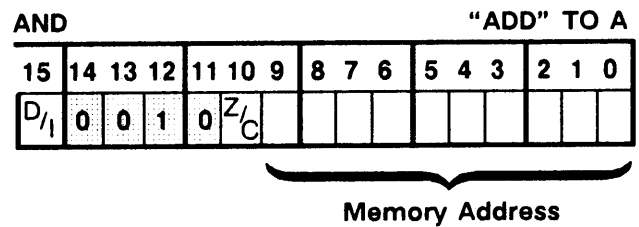
Memory Address

Combines the contents of the addressed memory location and the contents of the A-Register by performing a logical "AND" operation. The contents of the memory cell are unaltered.
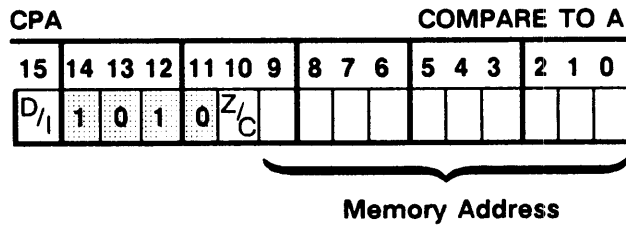
CPA   **COMPARE TO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	1 0 1	0 Z/C			

Memory Address

Compares the contents of the addressed memory location with the contents of the A-Register. If the two 16-bit words are not identical, the next instruction is skipped; that is, the P-Register advances two counts instead of one count. If the two words are identical, the next sequential instruction is executed (do if true). Neither the A-Register contents nor memory cell contents are altered.
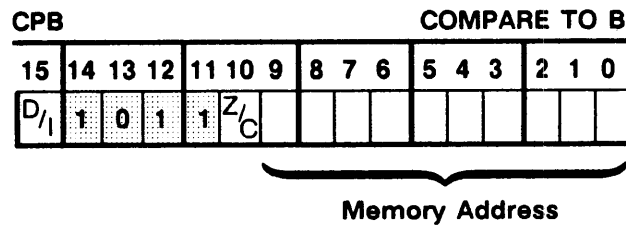
CPB   **COMPARE TO B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	1 0 1	1 Z/C			

Memory Address

Compares the contents of the addressed memory location with the contents of the B-Register. If the two 16-bit words are not identical, the next instruction is skipped; that is, the P-Register advances two counts instead of one count. If the two words are identical, the next sequential instruction is executed (do if true). Neither the B-Register contents nor memory cell contents are altered.
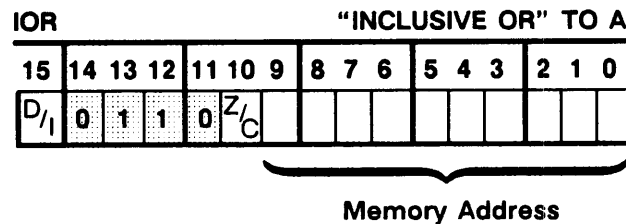
IOR   **"INCLUSIVE OR" TO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	0 1 1	0 Z/C			

Memory Address

Combines the contents of the addressed memory location and the contents of the A-Register by performing a logical "inclusive OR" operation. The contents of the memory cell are unaltered.

ISZ   **INCREMENT AND SKIP IF ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	0 1 1	1 Z_C			
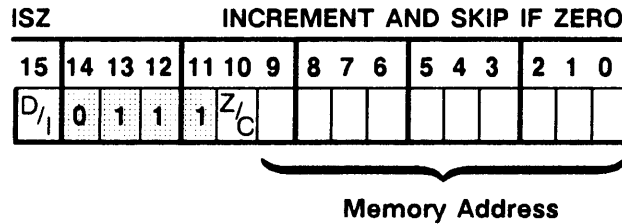
Memory Address

Adds one to the contents of the addressed memory location. If the result of this operation is zero (memory contents incremented from 177777 to 000000), the next instruction is skipped; that is, the P-Register is advanced two counts instead of one count. If the result of this operation is not zero, the next sequential instruction is executed. In either case, the incremented value is written back into the memory cell. Current page, direct addressing with this instruction will produce undefined results if CDS is enabled.

JMP   **JUMP**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	0 1 0	1 Z_C			
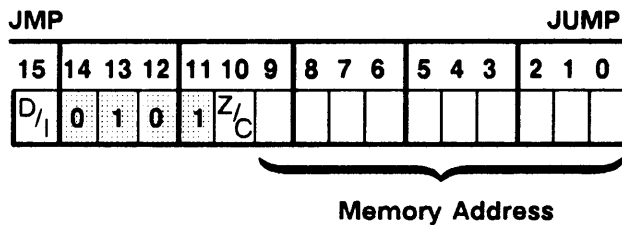
Memory Address

Transfers control to the addressed memory location. That is, a JMP causes the P-Register count to set according to the memory address portion of the JMP instruction so that the next instruction will be read from that location.

JSB   **JUMP TO SUBROUTINE**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	0 0 1	1 Z_C			
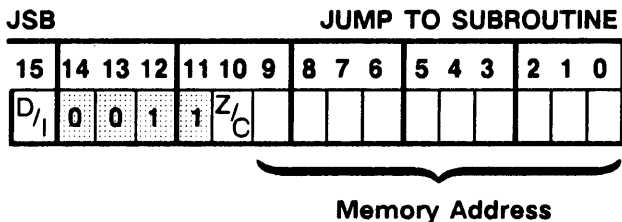
Memory Address

This instruction, executed in location P (P-Register count), causes the computer control to jump unconditionally to the memory location (m) specified by the memory address portion of the JSB instruction. The contents of the P-Register plus one (return address) is stored in memory location m, and the next instruction to be executed will be that contained in the next sequential memory location (m + 1). A return to the main program sequence at P + 1 will be effected by a JMP indirect through location m. This instruction has undetermined results if executed while CDS is enabled.

LDA   **LOAD A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	1 1 0	0 Z/C			
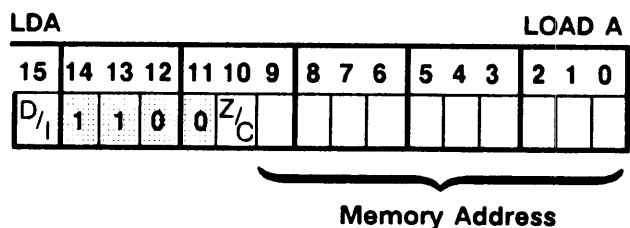
Memory Address

Loads the contents of the addressed memory location into the A-Register. The contents of the memory cell are unaltered.

LDB   **LOAD B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	1 1 0	1 Z/C			
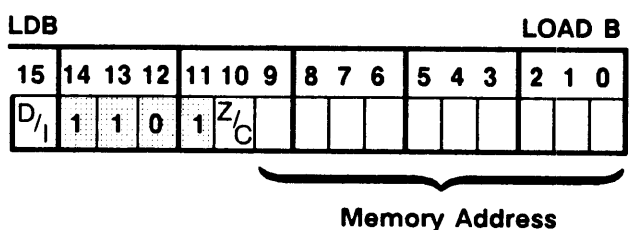
Memory Address

Loads the contents of the addressed memory location into the B-Register. The contents of the memory cell are unaltered.

STA   **STORE A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	1 1 1	0 Z/C			
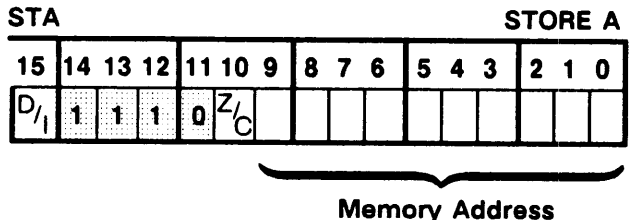
Memory Address

Stores the contents of the A-Register in the addressed memory location. The previous contents of the memory cell are list; the A-Register contents are unaltered. Current page, direct addressing with this instruction will produce undefined results if CDS is enabled.

STB   **STORE B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
D/I	1 1 1	1 Z/C			

Memory Address

Stores the contents of the B-Register in the addressed memory location. The previous contents of the memory cell are lost; the B-Register contents are unaltered. Current page, direct addressing with this instruction will produce undefined results if CDS is enabled.

XOR   **"EXCLUSIVE OR" TO A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	0	1	0	0	Z/C										
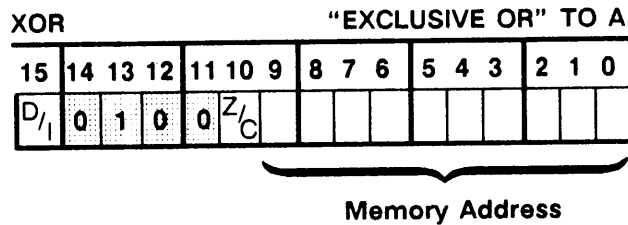
Memory Address

Combines the contents of the addressed memory location and the contents of the A-Register by performing a logical "exclusive OR" operation. The contents of the memory cell are unaltered.

Register Reference Instructions

The 39 register reference instructions execute functions on data contained in the A-Register, B-Register, and E-Register. These instructions are divided into two groups: the shift/rotate group (SRG) and the alter/skip group (ASG). In each group, several instructions may be combined into one word. Since the two groups perform separate and distinct functions, instructions from the two groups cannot be mixed. Unshaded bits in the coding diagrams are available for combining other instructions from the same group. The ASG and SRG instructions are not affected by the state of CDS.

Shift/Rotate Group. The 20 instructions in the shift/rotate group (SRG) are defined first; this group is specified by setting bit 10 to a logic 0. A comparison of the various shift/rotate functions are illustrated in Figure 3-3. Rules for combining instructions in this group are as follows (refer to Table 3-3):

1. Only one instruction can be chosen from each of the two multiple-choice columns.

2. References can be made to either the A-Register or B-Register, but not both.

3. Sequence of execution is from left to right.

4. In machine code, use zeros to exclude unwanted operations.

5. Code a logic 1 in bit position 9 to enable shifts or rotates in the first position; code a logic 1 in bit position 4 to enable shifts or rotates in the second position.

6. The extend bit is not affected unless specifically stated. However, if a "rotate-with-E" instruction (ELA, ELB, ERA, or ERB) is coded but disabled by a logic 0 in bit position 9 and/or position 4, the E-Register will be updated even though the A- or B-Register contents are not affected; to avoid this situation code a "no operation" (four zeros) in the first and/or second positions (3 zeros for ALS/BLS).

Table 3-3. Shift/Rotate Group Combining Guide

{ ALS ARS RAL RAR ALR ALF ERA ELA }	[,CLE]	[,SLA] ,	{ ALS ARS RAL RAR ALR ALF ERA ELA }
{ BLS BRS RBL RBR BLR BLF ERB ELB }	[,CLE]	[,SLB] ,	{ BLS BRS RBL RBR BLR BLF ERB ELB }
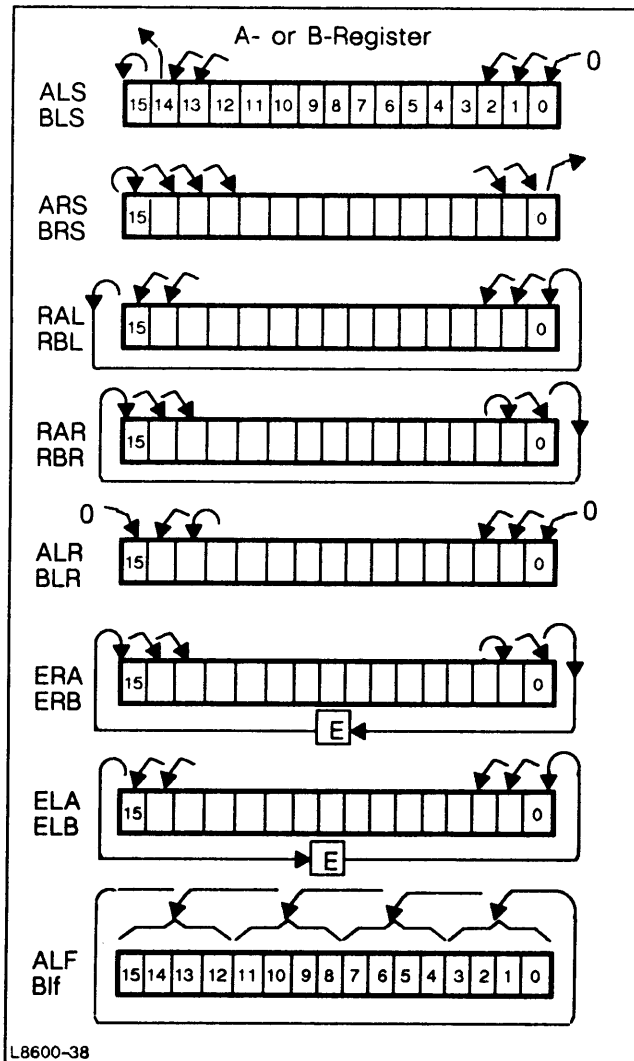
Figure 3-3. Shift and Rotate Functions
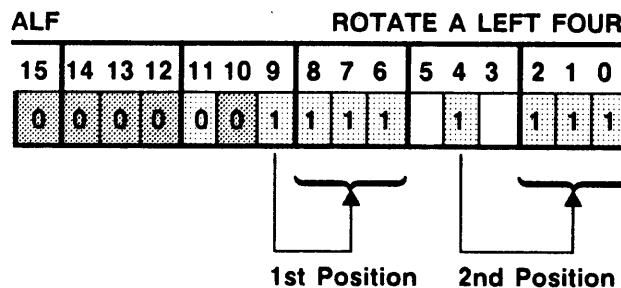
ALF   **ROTATE A LEFT FOUR**

1st Position   2nd Position

Rotates the A-Register contents (all 16 bits) left four places. Bits 15, 14, 13, and 12 rotate around to bit positions 3, 2, 1, and 0, respectively. Equivalent to four successive RAL instructions.

ALR   **A LEFT SHIFT, CLEAR SIGN**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0 1	1 0 0	1	1 0 0

1st Position   2nd Position

Shifts the A-Register contents left one place and clears sign bit 15.

ALS   **A LEFT SHIFT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0 1	0 0 0	1	0 0 0

1st Position   2nd Position

Arithmetically shifts the A-Register contents left one place, 15 magnitude bits only; bit 15 (sign) is not affected. The bit shifted out of bit position 14 is lost; a logic 0 replaces vacated bit position 0.

ARS   **A RIGHT SHIFT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0 1	0 0 1	1	0 0 1

1st Position   2nd Position

Arithmetically shifts the A-Register contents right one place, 15 magnitude bits only; bit 15 (sign) is not affected. A copy of the sign bit is shifted into bit position 14; the bit shifted out of bit position 0 is lost.

BLF   **ROTATE B LEFT FOUR**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0 1	1 1 1	1	1 1 1

1st Position   2nd Position

Rotates the B-Register contents (all 16 bits) left four places. Bits 15, 14, 13, and 12 rotate around to bit positions 3, 2, 1, and 0, respectively. Equivalent to four successive RBL instructions.

BLR   **B LEFT SHIFT, CLEAR SIGN**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0 1	1 0 0	1	1 0 0

1st Position   2nd Position

Shifts the B-Register contents left one place and clears sign bit 15.

BLS   **B LEFT SHIFT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0 1	0 0 0	1	0 0 0

1st Position   2nd Position

Arithmetically shifts the B-Register contents left one place, 15 magnitude bits only; bit 15 (sign) is not affected. The bit shifted out of bit position 14 is lost; a logic 0 replaces vacated bit position 0.

BRS   **B RIGHT SHIFT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0 1	0 0 1	1	0 0 1

1st Position   2nd Position

Arithmetically shifts the B-Register contents right one place, 15 magnitude bits only; bit 15 (sign) is not affected. A copy of the sign bit is shifted into bit position 14; the bit shifted out of bit position 0 is lost.

CLE   **CLEAR E**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0		1	

Clears the E-Register; that is, the extend bit becomes a logic 0.

ELA   **ROTATE E LEFT WITH A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0 1	1 1 0	1	1 1 0

1st Position   2nd Position

Rotates the E-Register content left with the A-Register contents (one place). The E-Register content rotates into bit position 0; bit 15 rotates into the E-Register.

ELB   **ROTATE E LEFT WITH B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0 1	1 1 0	1	1 1 0

1st Position   2nd Position
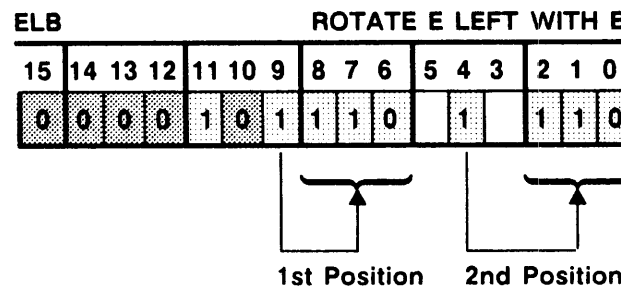
Rotates the E-Register content left with the B-Register contents (one place). The E-Register content rotates into bit position 0; bit 15 rotates into the E-Register.

ERA   **ROTATE E RIGHT WITH A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0 1	1 0 1	1	1 0 1

1st Position   2nd Position
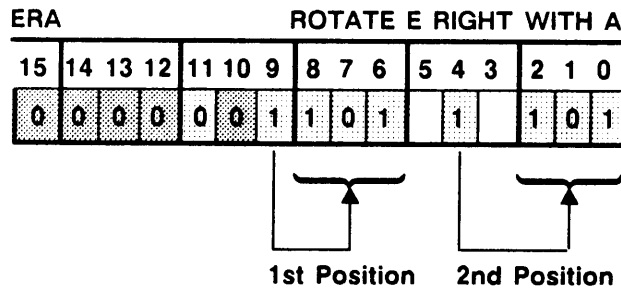
Rotates the E-Register content right with the A-Register contents (one place). The E-Register content rotates into bit position 15; bit 0 rotates into the E-Register.

ERB   **ROTATE E RIGHT WITH B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0 1	1 0 1	1	1 0 1

1st Position   2nd Position
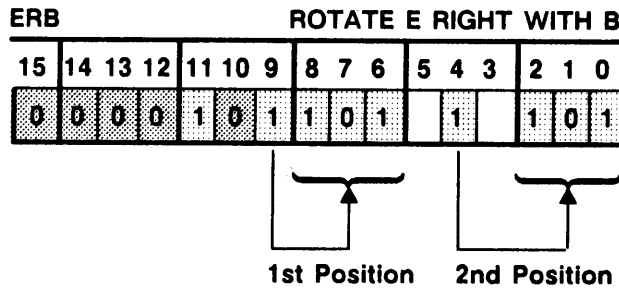
Rotates the E-Register content right with the B-Register contents (one place). The E-Register content rotates into bit position 15; bit 0 rotates into the E-Register.

NOP   **NO OPERATION**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
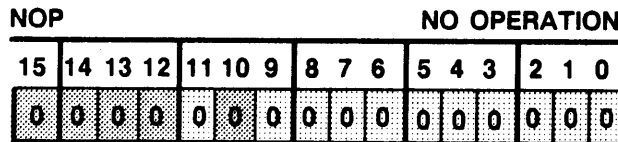
This all-zeros instruction causes a no-operation cycle.

RAL   **ROTATE A LEFT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0 1	0 1 0	1	0 1 0

1st Position   2nd Position

Rotates the A-Register contents left one place (all 16 bits). Bit 15 rotates into bit position 0.

RAR   **ROTATE A RIGHT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0 1	0 1 1	1	0 1 1

1st Position   2nd Position

Rotates the A-Register contents right one place (all 16 bits). Bit 0 rotates into bit position 15.

RBL   **ROTATE B LEFT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0 1	0 1 0	1	0 1 0

1st Position   2nd Position

Rotates the B-Register contents left one place (all 16 bits). Bit 15 rotates into bit position 0.

RBR   **ROTATE B RIGHT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0 1	0 1 1	1	0 1 1

1st Position   2nd Position

Rotates the B-Register contents right one place (all 16 bits). Bit 0 rotates into bit position 15.

SLA   **SKIP IF LSB OF A IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 0		1	

Skips the next instruction if the least-significant bit (bit 0) of the A-Register is a logic 0.

SLB   **SKIP IF LSB OF B IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 0		1	

Skips the next instruction if the least-significant bit (bit 0) of the B-Register is a logic 0.

Alter/Skip Group. The 19 instructions of the alter/skip group (ASG) are defined next. This group is specified by setting bit 10 to a logic 1. Rules for combining instructions are as follows (refer to Table 3-4).

Table 3-4. Alter/Skip Group Combining Guide

{ CLA CMA CCA }	[,SEZ]	{ CLE CME CCE }	[,SSA] [,SLA], [,INA] [,SZA] [,RSS]
{ CLB CMB CCB }	[,SEZ]	{ CLE CME CCE }	[,SSB] [,SLB] [,INB] [,SZB] [,RSS]

1. Only one instruction can be chosen from each of the two multiple-choice columns.

2. References can be made to either the A-Register or B-Register, but not both.

3. Sequence of execution is from left to right.

4. If two or more skip functions are combined, the skip function will occur if either or both conditions are met. One exception exists: refer to the RSS instruction.

5. In machine code, use zeros to exclude unwanted instructions.

CCA   **CLEAR AND COMPLEMENT A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 1 1	1		

Clears and complements the A-Register contents; that is, the contents of the A-Register become 177777 (octal). This is the two's complement form of −1.

CCB   **CLEAR AND COMPLEMENT B**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1								

Clears and complements the B-Register contents; that is, the contents of the B-Register become 177777 (octal). This is the two's complement form of –1.

CCE   **CLEAR AND COMPLEMENT E**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0		1			1	1						

Clears and complements the E-Register content (extend bit); that is, the extend bit becomes a logic 1.

CLA   **CLEAR A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1								

Clears the A-Register; that is, the contents of the A-Register becomes 000000 (octal).

CLB   **CLEAR B**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1								

Clears the B-Register; that is, the contents of the B-Register become 000000 (octal).

CLE   **CLEAR E**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0		1			0	1						

Clears the E-Register; that is, the extend bit becomes a logic 0.

CMA   **COMPLEMENT A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0								

Complements the A-Register contents (one's complement).

CMB COMPLEMENT B

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 1 1	0		

Complements the B-Register contents (one's complement).

CME COMPLEMENT E

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1	1 0		

Complements the E-Register content (extend bit).

INA INCREMENT A

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 1			1

Increments the A-Register by one. The overflow bit will be set if an increment of the largest positive number (077777 octal) is made. The extend bit will be set if an all-ones word (177777 octal) is incremented.

INB INCREMENT B

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 1			1

Increments the B-Register by one. The overflow bit will be set if an increment of the largest positive number (077777 octal) is made. The extend bit will be set if an all-ones word (177777 octal) is incremented.

RSS REVERSE SKIP SENSE

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1			1

Skip occurs for any of the following skip instructions, if present, when the non-zero condition is met. An RSS without a skip instruction in the word causes an unconditional skip. If a word with RSS also includes both SSA and SLA (or SSB and SLB), bits 15 and 0 must both be logic 1's for a skip to occur; in all other cases, a skip occurs if one or more skip conditions are met.

SEZ   **SKIP IF E IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1		1	

Skips the next instruction if the E-Register content (extend bit) is a logic 0.

SLA   **SKIP IF LSB OF A IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 1			1	

Skips the next instruction if the least-significant bit (bit 0) of the A-Register is a logic 0; that is, skips if an even number is in the A-Register.

SLB   **SKIP IF LSB OF B IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 1			1	

Skips the next instruction if the least-significant bit (bit 0) of the B-Register is a logic 0; that is, skips if an even number is in the B-Register.

SSA   **SKIP IF SIGN OF A IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 1		1	

Skips the next instruction if the sign bit (bit 15) of the A-Register is a logic 0; that is, skips if a positive number is in the A-Register.

SSB   **SKIP IF SIGN OF B IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 1		1	

Skips the next instruction if the sign bit (bit 15) of the B-Register is a logic 0; that is, skips if a positive number is in the B-Register.

SZA   **SKIP IF A IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	0 1			1

Skips the next instruction if the A-Register contents are zero (16 zeros).

SZB   **SKIP IF B IS ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0 0 0	1 1			1

Skips the next instruction if the B-Register contents are zero (16 zeros).

Input/Output Instructions

The following input/output instructions provide the capability of setting, clearing or testing the flag and control bits associated with DMA, programmed I/O, interrupts, memory protect, time base generator, parity error, Global Register, and overflow. I/O instructions with select codes of seven or less have various functions. (Refer to Table 5-3 for further information regarding specific select-code functions.) I/O instructions permit data transfer between the A- and B-Registers and either specific I/O devices or between registers associated with memory protect, parity error, or interrupts. The various registers and I/O devices are addressed by means of their register numbers and select codes.

Bit 11, where relevant, specifies the A- or B-Register or distinguishes between set control and clear control; otherwise, bit 11 may be a logic 0 or a logic 1 without affecting the instruction (although the assembler will assign zeros in this case). In those instructions where bit position 9 includes the letters H/C, the programmer has the choice of holding (logic 0) or clearing (logic 1) the device flag after executing the instruction. (Exception: the H/C bit associated within instructions SOC and SOS holds or clears the overflow bit instead of the device flag.) Note that this H/C option is not supported on some of the I/O instructions with select code less than 10 octal.

Bits 8, 7, and 6, specify the appropriate I/O instruction. When the Global Register is enabled, bits 5 through 0 apply the instruction to a register on the I/O card whose select code is in the Global Register.

NOTE

Execution of I/O instructions is inhibited when the memory protect
feature is enabled.

The following instruction descriptions assume that the Global Register is disabled and, therefore, the instructions are addressed to a select code. The extension of I/O instructions are not affected by the state of CDS.

CLC — CLEAR CONTROL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	H/C	1	1	1						

Select Code or Register Number
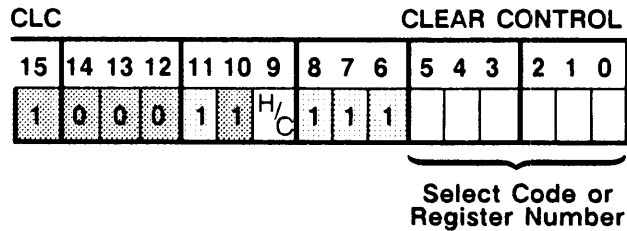
Clears the control bit (Control 30) of the selected I/O channel or function. This turns off the specific device channel and prevents it from interrupting. A CLC 00 instruction clears the control bits from select code 06 upward, effectively turning off all I/O devices.

CLF — CLEAR FLAG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0		1	1	0	0	1						

Select Code or Register Number
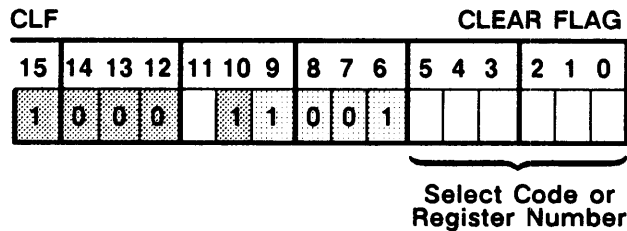
Clears the flag (Flag 30) of the selected I/O channel or function. A CLF 00 instruction disables the interrupt system for the time base generator and all interface cards; this does not affect the status of the individual channel flags.

CLO — CLEAR OVERFLOW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1

Clears the overflow bit.

HLT — HALT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0		1	H/C	0	0	0						

Select Code or Register Number
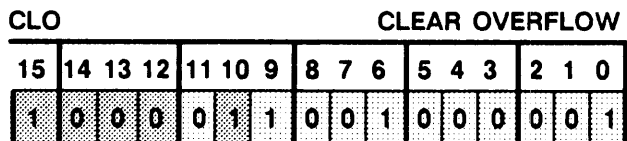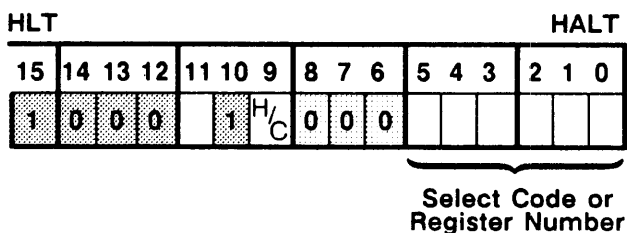
Halts the computer, holds or clears the flag of the selected I/O channel, and invokes the virtual control panel program. The HLT instruction will be contained in the T-Register, which is displayed on the VCP when the VCP program starts executing. The P-Register (also displayed) will contain the HLT location plus one. Note that if break is not enabled on any I/O card, the HLT instruction has no effect.
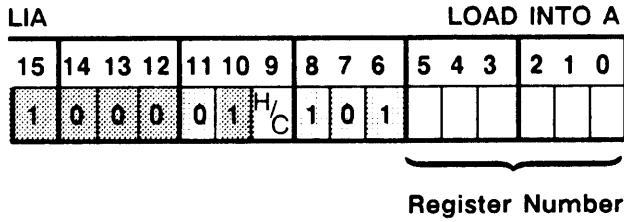
LIA   **LOAD INTO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 1 H/C	1 0 1		

Register Number

Loads the contents of the addressed I/O buffer or special function register into the A-Register.

LIB   **LOAD INTO B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 1 H/C	1 0 1		

Register Number

Loads the contents of the addressed I/O buffer or special function register into the B-Register.

MIA   **MERGE INTO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 1 H/C	1 0 0		

Register Number

By executing a logical "inclusive OR" function, merges the contents of the addressed I/O buffer or special function register into the A-Register.

MIB   **MERGE INTO B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 1 H/C	1 0 0		

Register Number

By executing a logical "inclusive OR" function, merges the contents of the addressed I/O buffer or special function register into the B-Register.

OTA   **OUTPUT A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 1 H/C	1 1 0		

Register Number
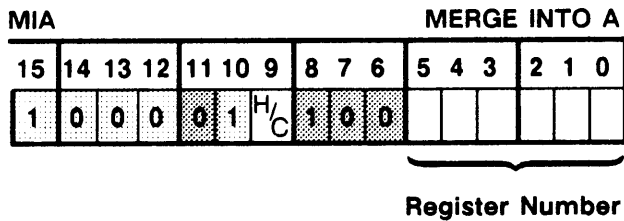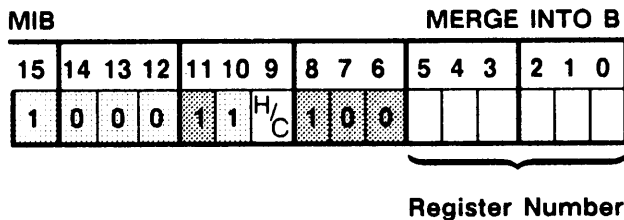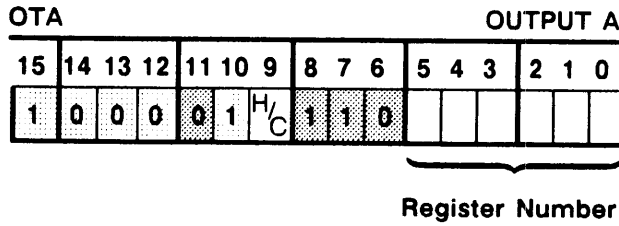
Outputs the contents of the A-Register to the addressed I/O buffer or special function register. The contents of the A-Register are not altered.

OTB   **OUTPUT B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 1 H/C	1 1 0		

Register Number
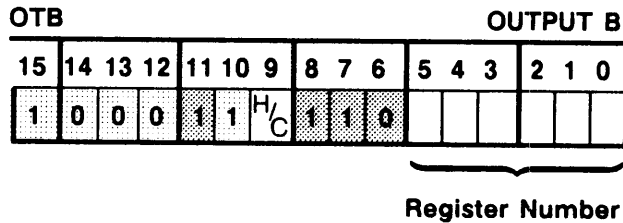
Outputs the contents of the B-Register to the addressed I/O buffer or special function register. The contents of the B-Register are not altered.

SFC   **SKIP IF FLAG CLEAR**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 0	1 0		

Select Code or
Register Number
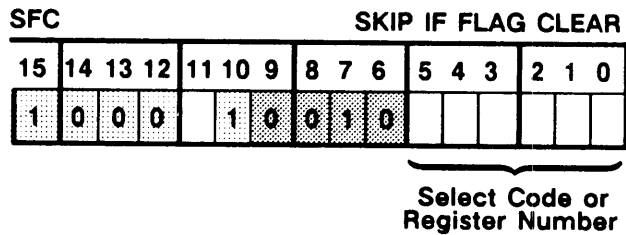
Skips the next programmed instruction if the flag (Flag 30) of the selected channel is clear (device busy).

SFS   **SKIP IF FLAG SET**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 0	1 1		

Select Code or
Register Number
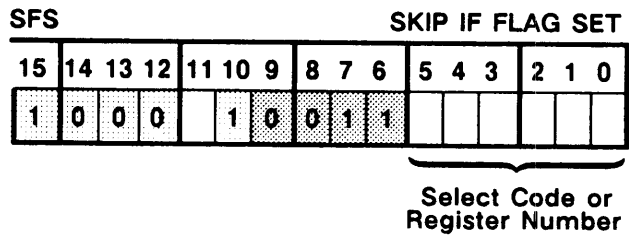
Skips the next programmed instruction if the flag (Flag 30) of the selected channel is set (device ready).

SOC   **SKIP IF OVERFLOW CLEAR**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 1 H/C	0 1 0	0 0 0	0 0 1

Skips the next programmed instruction if the overflow bit is clear. Use the H/C (bit 9) to either hold or clear the overflow bit following the completion of this instruction (whether the skip is taken or not).

SOS   **SKIP IF OVERFLOW SET**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 1 H/C	0 1 1	0 0 0	0 0 1

Skips the next programmed instruction if the overflow bit is set. Use the H/C bit (bit 9) to either hold or clear the overflow bit following the completion of this instruction (whether the skip is taken or not).

STC   **SET CONTROL**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 1 H/C	1 1 1		

Select Code or Register Number

Sets the control bit (Control 30) of the selected I/O channel or function.

STF   **SET FLAG**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0	0 0 1		

Select Code or Register Number

Sets the flag (Flat 30) of the selected I/O channel or function. An STF 00 instruction enables the interrupt system for the time base generator and all interface cards.

STO   SET OVERFLOW

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 1 0	0 0 1	0 0 0	0 0 1

Sets the overflow bit.

Extended Arithmetic Memory Reference Instructions

The four extended arithmetic memory reference instructions provide for integer multiply and divide and for loading and storing double-length words to and from the A- and B-Registers. The complete instruction requires two words: one for the instruction code and one for the address. When stored in memory, the instruction word is the first to be fetched; the address word is in the next sequential location.

Since 15 bits are available for the address, these instructions can directly address any location in memory. As for all memory reference instructions, indirect addressing to any number of levels may also be used. A logic 0 in bit position 15 specifies direct addressing; a logic 1 specifies indirect addressing.

DIV   DIVIDE

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 0	1 0 0	0 0 0	0 0 0
D/I					

Memory Address

Divides a double-word integer in the combined B- and A-Registers by a 16-bit integer in the addressed memory location. The result is a 16-bit integer quotient in the A-Register and a 16-bit integer remainder in the B-Register. Overflow can result from an attempt to divide by zero, or from an attempt to divide by a number too small for the dividend. In the former case (divide by zero), the division will not be attempted and the B- and A-Register contents will be unchanged except that a negative quantity will be made positive. In the latter case (divisor too small), the execution will be attempted with unpredictable results left in the B- and A-Registers. If there is no divide error, the overflow bit is cleared.

DLD — DOUBLE LOAD

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 0	0 1 0	0 0 0	0 0 0
D/I					
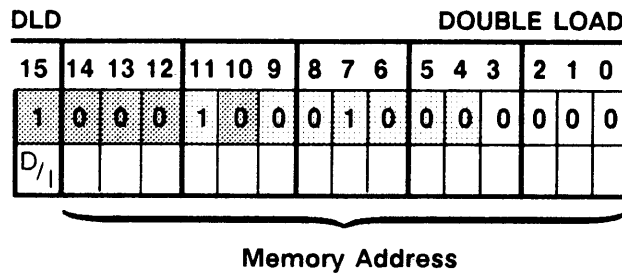
Memory Address

Loads the contents of addressed memory location m (and m+1) into the A- and B-Registers, respectively. If m is base relative and CDS mode is enabled, the base register will be added to m and the references will come from m+Q and m+Q+1 (even if m+1 is not base relative).

DST — DOUBLE STORE

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 0	0 1 0	0 0 0	0 0 0
D/I					
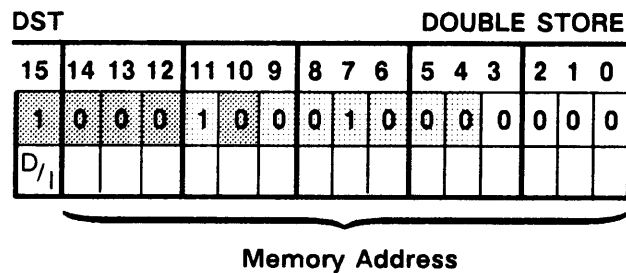
Memory Address

Stores the double-word quantity in the A- and B-Registers into addressed memory locations m (and m+1), respectively. If m is base relative and CDS mode is enabled, the base register will be added to m and the references will come from m+Q and m+Q+1 (even if m+1 is not base relative).

MPY — MULTIPLY

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 0	0 1 0	0 0 0	0 0 0
D/I					
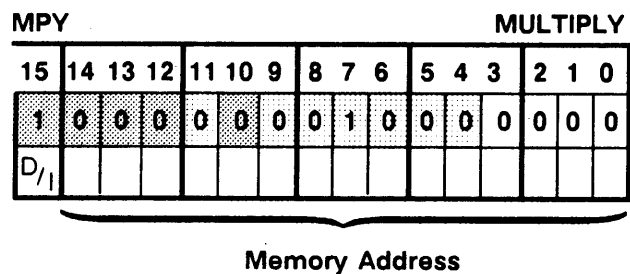
Memory Address

Multiplies a 16-bit integer in the A-Register by a 16-bit integer in the addressed memory location. The resulting double-length integer product resides in the B- and A-Registers, with the B-Register containing the sign bit and the most-significant 15 bits of the quantity. The A-Register may be used as an operand (that is, memory address 0), resulting in an arithmetic square. The instruction clears the overflow bit.

Extended Arithmetic Register Reference Instructions

The six extended arithmetic register reference instructions provide various types of shifting operations on the combined contents of the B- and A-Registers. The B-Register is considered to be to the left (most-significant word) and the A-Register is considered to be to the right (least-significant word). An example of each type of shift operation is illustrated in Figure 3-4.

The complete instruction is given in one word and includes four bits (unshaded) to specify the number of shifts (1 to 16). Viewing these four bits as a binary-coded number enables the number of shifts to be easily understood; that is, binary-coded 1 = 1 shift, binary-coded 2 = 2 shifts ... binary-coded 15 = 15 shifts. The maximum number of 16 shifts is coded with four zeros, which essentially exchanges the contents of the B- and A-Registers.

The extend bit is not affected by any of the following instructions. Except for the arithmetic shifts, overflow also is not affected.

The execution of extended arithmetic register reference instructions is not affected by the state of CDS.

ASL   ARITHMETIC SHIFT LEFT

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 0	0 0 0	0 1	

**Number
of Shifts**

Arithmetically shifts the combined contents of the B- and A-Registers left n places. The value of n may be any number from 1 through 16. Zeros are filled into vacated low-order positions of the A-Register. The sign bit is not affected, and data bits are lost out of bit position 14 of the B-Register. If any one of the lost bits is a significant data bit ("1" for positive numbers, "0" for negative numbers), the overflow bit will be set; otherwise, overflow will be cleared during execution. See ASL example in Figure 3-4. Note that two additional shifts in this example would cause an error by losing a significant '1'.

ASR   ARITHMETIC SHIFT RIGHT

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	0 0 0	0 1	

**Number
of Shifts**

Arithmetically shifts the combined contents of the B- and A-Registers right n places. The value of n may be any number from 1 through 16. The sign bit is unchanged and is extended into bit positions vacated by the right shift. Data bits shifted out of the least-significant end of the A-Register are lost. Overflow cannot occur because the instruction clears the overflow bit.

	B-REGISTER	A-REGISTER

ASR 5
(Arithmetic Shift Right 5 places)

Bits lost

1 011 000 101 000 101 | 0 101 101 011 100 111

1 111 110 110 001 010 | 0 010 101 011 010 111

Extended sign

ASL 5
(Arithmetic Shift Left 5 places)

Bits lost

0 000 000 111 101 000 | 1 101 101 000 110 111

0 011 110 100 011 011 | 0 100 011 011 100 000

Zeros filled

LSR 5
(Logical Shift Right 5 places)

Bits lost

1 011 000 101 000 101 | 0 101 101 011 100 111

Zeros Filled

0 000 010 110 001 010 | 0 010 101 011 010 111

LSL 5
(Logical Shift Left 5 places)

Bits lost

0 101 000 111 101 000 | 1 101 101 000 110 111

0 011 110 100 011 011 | 0 100 011 011 100 000

RRR 8
(Rotate Right 8 places)

0 101 110 111 000 010 | 0 100 010 110 000 111

1 000 011 101 011 101 | 1 100 001 001 000 101

RRL 7
(Rotate Left 7 places)

0 110 011 101 111 000 | 0 110 011 010 000 111

1 011 110 000 110 011 | 0 100 001 110 110 011

L8600-39

Figure 3-4. Examples of Double-Word Shifts and Rotates

LSL   LOGICAL SHIFT LEFT

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	0 0 0	0 0 0	1 0	

Number of Shifts

Logically shifts the combined contents of the B- and A-Registers left n places. The value of n may be any number from 1 through 16. Zeros are filled into vacated low-order bit positions of the A-Register; data bits are lost out of the high-order bit positions of the B-Register.

LSR   LOGICAL SHIFT RIGHT

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	0 0 0	1 0	

Number of Shifts

Logically shifts the combined contents of the B- and A-Registers right n places. The value of n may be any number from 1 through 16. Zeros are filled into vacated high-order bit positions of the B-Register; data bits are lost out of the low-order bit positions of the A-Register.
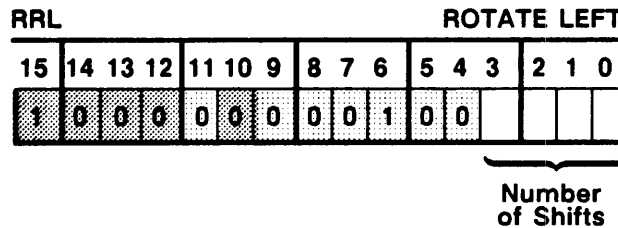
RRL   ROTATE LEFT

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 0	0 0 1	0 0	

Number of Shifts

Rotates the combined contents of the B- and A-Registers left n places. The value of n may be any number from 1 through 16. No bits are lost or filled in. Data bits shifted out of the high-order end of the B-Register are rotated around to enter the low-order end of the A-Register.
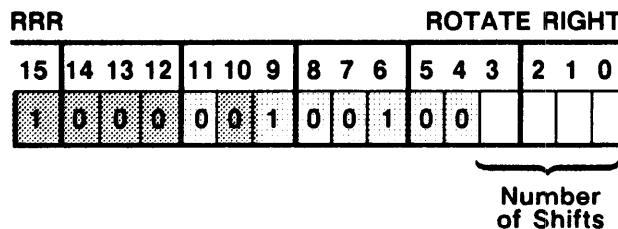
RRR   ROTATE RIGHT

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	0 0 1	0 0	

Number of Shifts

Rotates the combined contents of the B- and A-Registers right n places. The value of n may be any number from 1 through 16. No bits are lost or filled in. Data bits shifted out of the high-order end of the A-Register are rotated around to enter the high-order end of the B-Register.

Extended Instruction Group

Index/Register Instructions. The index registers (X and Y) are two 16-bit registers accessible by the following instructions:

ADX   **ADD MEMORY TO X**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	0	1	1	0
D/I															
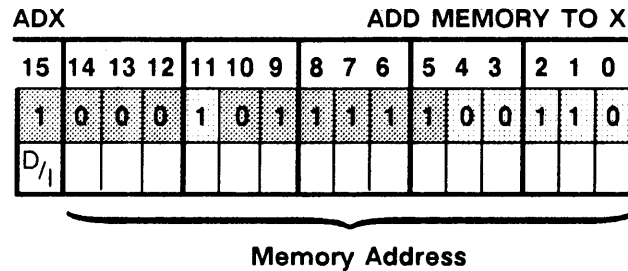
Memory Address

Adds the contents of the addressed memory location to the contents of the X-Register. The sum remains in the X-Register and the contents of the memory cell are unaltered. The result of this addition may set the extend bit or the overflow bit. The A- and B-Registers can be referenced as memory locations 0 and 1, respectively.

ADY   **ADD MEMORY TO Y**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	1	1	1	0
D/I															
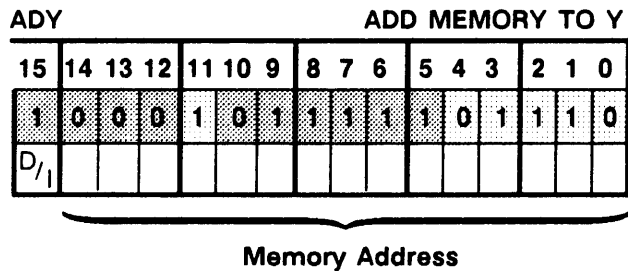
Memory Address

Adds the contents of the addressed memory location to the contents of the Y-Register. The sum remains in the Y-Register and the contents of the memory cell are unaltered. The result of this addition may set the extend bit or the overflow bit. The A- and B-Registers can be referenced as memory locations 0 and 1, respectively.

CAX   **COPY A TO X**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1
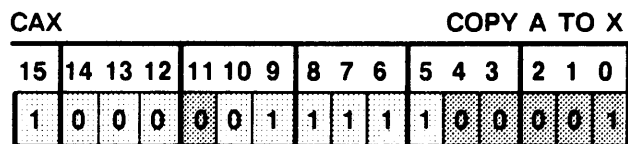
Copies the contents of the A-Register into the X-Register. The contents of the A-Register are unaltered.

CAY COPY A TO Y

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	1	0	1	0	0	1

Copies the contents of the A-Register into the Y-Register. The contents of the A-Register are unaltered.

CBX COPY B TO X

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	0	0	0	1

Copies the contents of the B-Register into the X-Register. The contents of the B-Register are unaltered.

CBY COPY B TO Y

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	1	0	0	1

Copies the contents of the B-Register into the Y-Register. The contents of the B-Register are unaltered.

CXA COPY X TO A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	1	0	0	1	0	0

Copies the contents of the X-Register into the A-Register. The contents of the X-Register are unaltered.

CXB COPY X TO B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	0	1	0	0

Copies the contents of the X-Register into the B-Register. The contents of the X-Register are unaltered.

CYA   **COPY Y TO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	1 0 1	1 0 0

Copies the contents of the Y-Register into the A-Register. The contents of the Y-Register are unaltered.

CYB   **COPY Y TO B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 0 1	1 0 0

Copies the contents of the Y-Register into the B-Register. The contents of the Y-Register are unaltered.

DSX   **DECREMENT X AND SKIP IF ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 0	0 0 1

Subtracts one from the contents of the X-Register. If the result of this operation is zero (X-Register decremented from 000001 to 000000), the next instruction is skipped; that is, the P-Register count is advanced two counts instead of one count. If the result is not zero, the next sequential instruction is executed.

DSY   **DECREMENT Y AND SKIP IF ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 1	0 0 1

Subtracts one from the contents of the Y-Register. If the result of this operation is zero (Y-Register decremented from 000001 to 000000), the next instruction is skipped; that is, the P-Register count is advanced two counts instead of one count. If the result is not zero, the next sequential instruction is executed.

ISX   **INCREMENT X AND SKIP IF ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 0	0 0 0

Adds one to the contents of the X-Register. If the result of this operation is zero (X-Register rolls over to 000000 from 177777), the next instruction is skipped; that is, the P-Register count is advanced two counts instead of one count. If the result is not zero, the next sequential instruction is executed.

ISY   **INCREMENT Y AND SKIP IF ZERO**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 1	0 0 0

Adds one to the contents of the Y-Register. If the result of this operation is zero (Y-Register rolls over to 000000 from 177777), the next instruction is skipped; that is, the P-Register count is advanced two counts instead of one count. If the result is not zero, the next sequential instruction is executed.

LAX   **LOAD A INDEXED BY X**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	1 0 0	0 1 0
D/I					

Operand Address

Loads the A-Register with the contents indicated by the effective address, which is computed by adding the contents of the X-Register to the operand address. The effective address is loaded into the M-Register; the X-Register and memory contents are not altered. Indirect addressing is resolved before indexing; bit 15 of the effective address is ignored. If CDS mode is enabled, the operand address is resolved for base relativity and the base register will be added before indexing. The index value can be positive or negative.

LAY   **LOAD A INDEXED BY Y**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0
D/I															

Operand Address

Loads the A-Register with the contents indicated by the effective address, which is computed by adding the contents of the Y-Register to the operand address. The effective address is loaded into the M-Register; the Y-Register and memory contents are not altered; Indirect addressing is resolved before indexing; bit 15 of the effective address is ignored. If CDS mode is enabled, the operand address is resolved for base relativity and the base register will be added before indexing. The index value can be positive or negative.

LBX   **LOAD B INDEXED BY X**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	0	0	1	0
D/I															

Operand Address

Loads the B-Register with the contents indicated by the effective address, which is computed by adding the contents of the X-Register to the operand address. The effective address is loaded into the M-Register; the X-Register and memory contents are not altered. Indirect addressing is resolved before indexing; bit 15 of the effective address is ignored. If CDS mode is enabled, the operand address is resolved for base relativity and the base register will be added before indexing. The index value can be positive or negative.

LBY   LOAD B INDEXED BY Y

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 0 1	0 1 0
D/I					

Operand Address

Loads the B-Register with the contents indicated by the effective address, which is computed by adding the contents of the Y-Register to the operand address. The effective address is loaded into the M-Register; the X-Register and memory contents are not altered. Indirect addressing is resolved before indexing; bit 15 of the effective address is ignored. If CDS mode is enabled, the operand address is resolved for base relativity and the base register will be added before indexing. The index value can be positive or negative.

LDX   LOAD X FROM MEMORY

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 0 0	1 0 1
D/I					

Operand Address

Loads the contents of the addressed memory location into the X-Register. The A- and B-Registers may be addressed as locations 00000 and 00001, respectively; however, if it is desired to load from the A- or B-Register, copy instruction CAX or CBX should be used since they are more efficient.

LDY   LOAD Y FROM MEMORY

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 0 1	1 0 1
D/I					

Operand Address
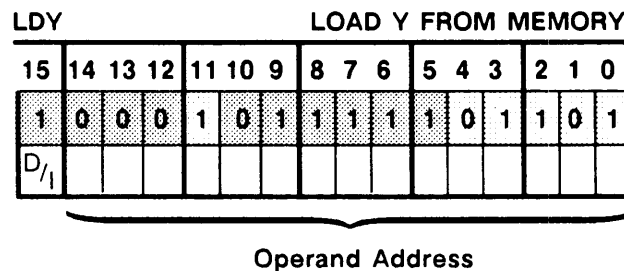
Loads the contents of the addressed memory location into the Y-Register. The A- and B-Registers may be addressed as locations 00000 and 00001, respectively; however, if it is desired to load from the A- or B-Register, copy instruction CAY or CBY should be used because they are more efficient.
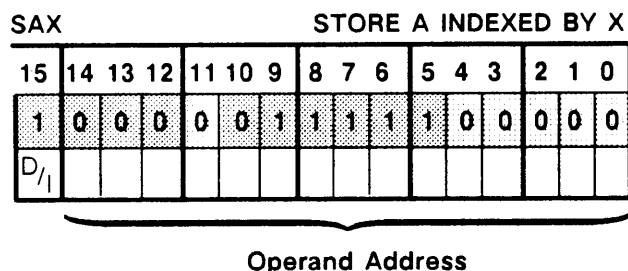
SAX **STORE A INDEXED BY X**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
D/I															

Operand Address

Stores the contents of the A-Register into the location indicated by the effective address, which is computed by adding the contents of the X-Register to the operand address. The effective address is loaded into the M-Register; the A- and X-Register contents are not altered. Indirect addressing is resolved before indexing; bit 15 of the effective address is ignored. If CDS mode is enabled, the operand address is resolved for base relativity and the base register will be added before indexing. The index value can be positive or negative.

SAY **STORE A INDEXED BY Y**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0
D/I															
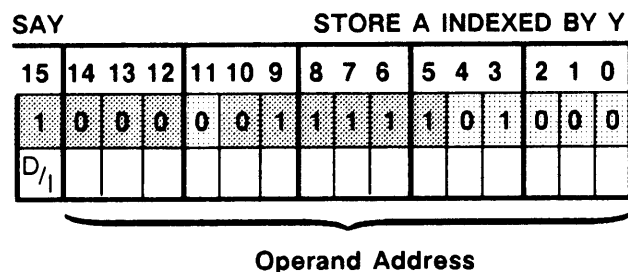
Operand Address

Stores the contents of the A-Register into the location indicated by the effective address, which is computed by adding the contents of the Y-Register to the operand address. The effective address is loaded into the M-Register; the A- and Y-Register contents are not altered. Indirect addressing is resolved before indexing; bit 15 of the effective address is ignored. If CDS mode is enabled, the operand address is resolved for base relativity and the base register will be added before indexing. The index value can be positive or negative.

SBX **STORE B INDEXED BY X**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	0	0	0	0
D/I															
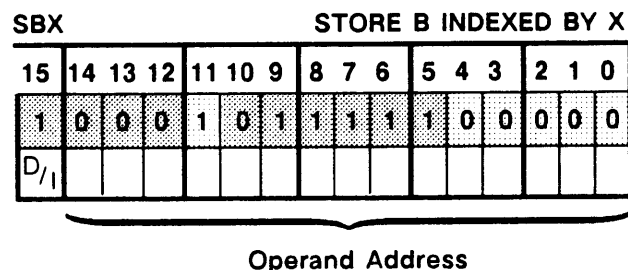
Operand Address

Stores the contents of the B-Register into the location indicated by the effective address, which is computed by adding the contents of the X-Register to the operand address. The effective address is loaded into the M-Register; the B- and X-Register contents are not altered. Indirect addressing is resolved before indexing; bit 15 of the effective address is ignored. If CDS mode is enabled, the operand address is resolved for base relativity and the base register will be added before indexing. The index value can be positive or negative.

SBY   **STORE B INDEXED BY Y**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0
D/I															
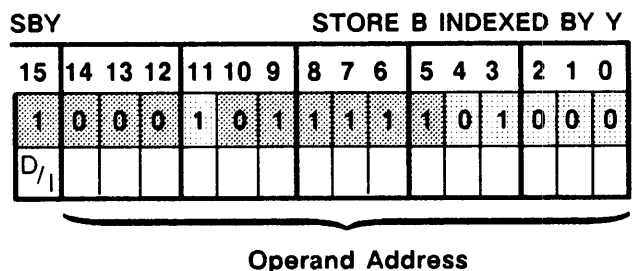
Operand Address

Stores the contents of the B-Register into the location indicated by the effective address, which is computed by adding the contents of the Y-Register to the operand address. The effective address is loaded into the M-Register; the B- and Y-Register contents are not altered. Indirect addressing is resolved before indexing; bit 15 of the effective address is ignored. If CDS mode is enabled, the operand address is resolved for base relativity and the base register will be added before indexing. The index value can be positive or negative.

STX   **STORE X TO MEMORY**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	0	0	0	1	1
D/I															
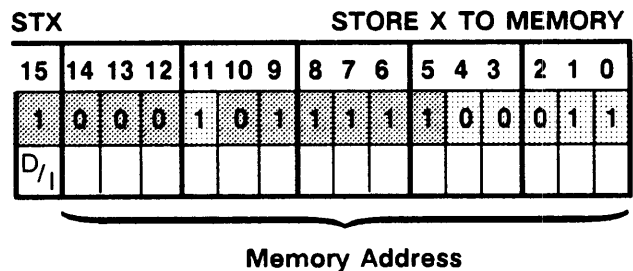
Memory Address

Stores the contents of the X-Register into the addressed memory location. The A- and B-Registers may be addressed as locations 00000 and 00001, respectively. The X-Register contents are not altered.

STY   **STORE Y TO MEMORY**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 0 1	0 1 1
D/I					

Memory Address

Stores the contents of the Y-Register into the addressed memory location. The A- and B-Registers may be addressed as locations 00000 and 00001, respectively. The Y-Register contents are not altered.

XAX   **EXCHANGE A AND X**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	1 0 0	1 1 1

Exchanges the contents of the A- and X-Registers.

XAY   **EXCHANGE A AND Y**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	1 0 1	1 1 1

Exchanges the contents of the A- and Y-Registers.

XBX   **EXCHANGE B AND X**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 0 0	1 1 1

Exchanges the contents of the B- and X-Registers.

XBY   **EXCHANGE B AND Y**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 0 1	1 1 1

Exchanges the contents of the B- and Y-Registers.

Jump Instructions. The following four jump instructions allow a program to either jump to or exit from a subroutine.
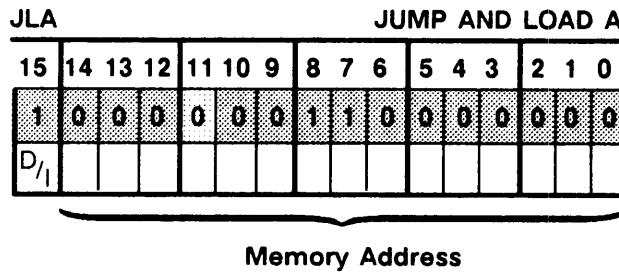
JLA   JUMP AND LOAD A

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 0	1 1 0	0 0 0	0 0 0
D/I					

Memory Address

This instruction, executed in location P, causes computer control to jump unconditionally to the memory location specified by the second word of the instruction. The contents of the program counter plus two are stored in the A-Register. A return to the main program will be effected by a JMP indirect through location 00000 (the A-Register).
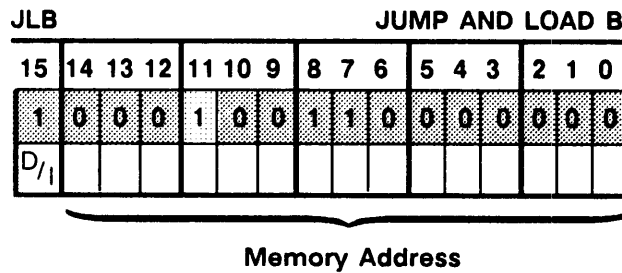
JLB   JUMP AND LOAD B

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 0	1 1 0	0 0 0	0 0 0
D/I					

Memory Address

This instruction, executed in location P, causes computer control to jump unconditionally to the memory location specified by the second word of the instruction. The contents of the program counter plus two are stored in the A-Register. A return to the main program will be effected by a JMP indirect through location 00001 (the B-Register).

JLY   JUMP AND LOAD Y

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 0	0 1 0
D/I					
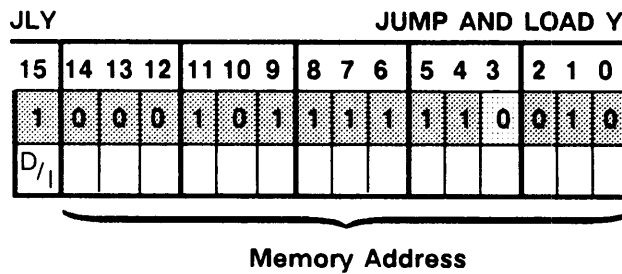
Memory Address

This instruction is designed for entering a subroutine. The instruction, executed in location P, causes computer control to jump unconditionally to the memory location specified in the memory address. Indirect addressing may be specified. The contents of the P-Register plus two (return address) are loaded into the Y-Register. A return to the main program sequence at P + 2 may be effected by a JPY instruction (described next).

JPY   **JUMP INDEXED BY Y**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 1	0 1 0
D/I					
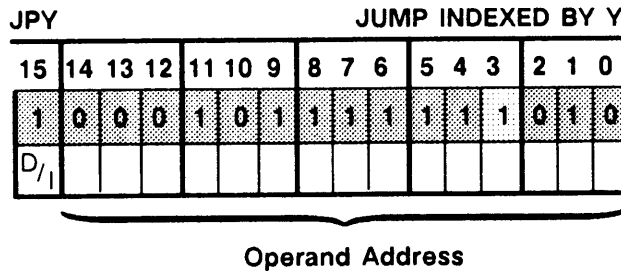
Operand Address

Transfers control to the effective address, which is computed by adding the contents of the Y-Register to the operand address. Indirect addressing is not allowed. The effective address is loaded into the P-Register; the Y-Register contents are not altered. Memory protect checks are performed on all references to memory (read, write, fetch), except references to memory locations 0 and 1 (A and B).

Byte Manipulation Instructions. A byte address is defined as two times the word address plus zero or one, depending on whether the byte is in the high-order position (bits 8 through 15) or low-order position (bits 0 through 7) of the word containing it. If the byte of interest is in bit positions 8 through 15 of memory location 100, for example, then the address of that byte is 2* 100 + 0, or 200; the address of the low-order byte in the same location is 201 (2* 100 + 1). Because of the way byte addresses are defined, 16 bits are required to cover all possible byte addresses in the 32k-word Logical address space (memory goes to 4M bytes). Hence, for byte addressing, bit 15 does not indicate indirect addressing. Memory references to byte addresses on base page (4-3777) with CDS mode enabled will have 2*Q (byte base register) added to the base relative address.

Byte addresses 000 through 003 reference bytes in the A- and B-Registers. These addresses will not cause memory violations. The user should, however, be careful in referencing these byte addresses: for example, storing into byte address 002 or 003 would destroy the byte address originally contained in the B-Register.

CBT   **COMPARE BYTES**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	1	0	1	1	0
D/I															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Return if array 1 = array 2

Return if array 1 < array 2

Return if array 1 > array 2

Compares the bytes in string 1 with those in string 2. This is a three-word instruction where:

Word 1 = Instruction code,

Word 2 = Address of word containing the string count, and

Word 3 = All-zeros word reserved for use by microcode.

The operand addresses are in the A- and B-Registers. The A-Register contains the first byte address of string 1 and the B-Register contains the first byte address of string 1.

The number of bytes to be compared is given in the memory location addressed by Word 2 of the instruction; the number of bytes to be compared is restricted to a positive integer greater than zero. The strings are compared one byte at a time; the ith byte in string f1 is compared with the ith byte in string 2. The comparison is performed arithmetically; that is, each byte is treated as a positive number. If all bytes in string 1 are identical with all bytes in string 2, the "equal" exit is taken. As soon as two bytes are compared and found to be different, the "less than" or "greater than" exit is taken, depending on whether the byte in string 1 is less than or greater than the byte in string 2. The three ways this instruction exits are as follows:

1. No skip if string 1 is equal to string 2; the P-Register advances one count from Word 3 of the instruction. The A-Register contains its original value incremented by the count stored in the address specified in Word 2.

2. Skips one word if string 1 is less than string 2; the P-Register advances two counts from Word 3 of the instruction. The A-Register contains the address of the byte in string 1 where the comparison stopped.

3. Skips two words if string 1 is greater than string 2; the P-Register advances three counts from Word 3 of the instruction. The A-Register contains the address of the byte in string 1 where the comparison stopped.

For all three exits, the B-Register will contain its original value incremented by the count stored in the address specified in Word 2. This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-Registers. During the interrupt, the remaining count is stored in Word 3 of the instruction. This instruction has undefined results if executed with CDS mode enabled.

LBT   **LOAD BYTE**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	1	0	0	1	1

This one-word instruction loads into the A-Register the byte whose address is contained in the B-Register. The byte is right-justified with leading zeros in the left byte. The B-Register is incremented by one.

MBT   **MOVE BYTES**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	1	0	1	0	1
D/I															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Moves bytes in a left-to-right manner; that is, the byte having the lowest address from the source is moved first. This is a three word instruction where:

Word 1 = Instruction code,

Word 2 = Address of word containing the byte count, and

Word 3 = All-zeros word reserved for use by microcode.

The operand addresses are in the A- and B-Registers. The A-Register contains the first byte address source and the B-Register contains the first byte address destination.

The number of bytes to be moved is given by a 16-bit positive integer greater than zero addressed by Word 2 of the instruction. The byte address in the A- and B-Registers are incremented as each byte is being moved. Thus, at the end of the operation, the A- and B-Registers are incremented by the number of bytes moved. Wraparound of the byte address would result from a carry out of bit position 15; therefore, if the destination became 000, 001, 002, or 003, the next byte would be moved into the A- or B-Register and destroy the proper byte addresses for the move operation. For each byte move, a memory protect check is performed.

This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-Registers. During the interrupt, the remaining count is stored in Word 3 of the instruction. This instruction has undefined results if executed with CDS mode enabled.

SBT   **STORE BYTE**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 0	1 0 0

Stores the A-Register low-order (right) byte in the byte address contained in the B-Register. The B-Register is incremented by one. A memory protect check is performed before the byte is stored. The left byte in the A-Register does not have to be zeros. The other byte in the same word of the stored byte is not altered.

SFB   **SCAN FOR BYTE**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 0	1 1 1

This is a one word instruction with the operands in the A- and B-Registers. The A-Register contains a termination byte (high-order byte) and a test byte (low-order byte). The B-Register contains the first byte address of the string to be scanned.

A string of bytes is scanned starting at the byte address given in the B-Register. Scanning terminates when a byte in the string matches either the test byte or the termination byte in the A-Register. The manner in which the instruction exits depends on which byte is matched first. If a byte in the string matches the test byte, the instruction will not skip upon exit; the B-Register will contain the address of the byte matching the test byte. If a byte in the string matches the termination byte, the instruction will skip one word upon exit; the B-Register will contain the address of the byte matching the termination byte *plus one*.

The scanning operation will not continue indefinitely even if neither the termination byte nor test byte exists in memory. These bytes are in the A-Register with byte addresses 000 and 001, respectively. Thus, if no match is made by the time the B-Register points to the last byte in memory, the B-Register will roll over to zero and the next test will match the termination byte in the A-Register with itself.

This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-Registers.

Bit Manipulation Instruction. The following three instructions allow any number of bits in a specified memory location to be cleared, set, or tested.

CBS   **CLEAR BITS**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0
D/I															
D/I															

Memory Address
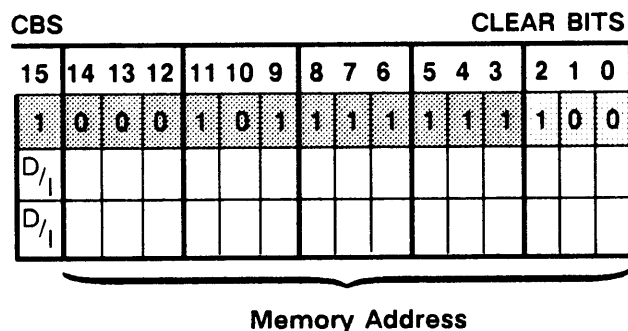
Clears bits in the addressed location. This is a three-word instruction where:

Word 1 = Instruction code,

Word 2 = Address of a 16-bit mask, and

Word 3 = Address of word where bits are to be cleared.

The bits to be cleared correspond to logic 1's in the mask. The bits corresponding to logic 0's in the mask are not affected. A memory protect check is performed prior to modifying the word in memory.

SBS   **SET BITS**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	1	1	0	1	1
D/I															
D/I															

Memory Address
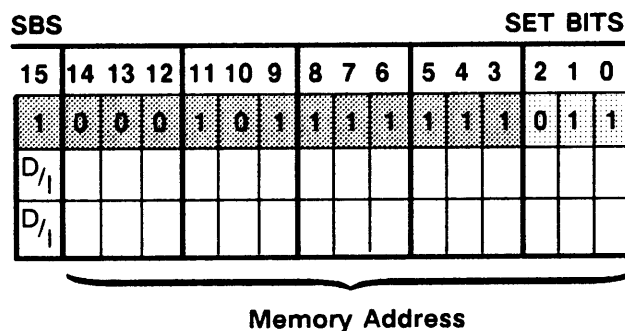
Sets bits in the addressed location. This is a three-word instruction where:

Word 1 = Instruction code,

Word 2 = Address of a 16-bit mask, and

Word 3 = Address of word where bits are to be set.

The bits to be set correspond to logic 1's in the mask. The bits corresponding to logic 0's in the mask are not affected. A memory protect check is performed prior to modifying the word in memory.

TBS   **TEST BITS**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	1	1	1	0	1
D/I															
D/I															

Memory Address

Tests (compares) bits in the addressed location. This is a three-word instruction where:

Word 1 = Instruction code,

Word 2 = Address of a 16-bit mask, and

Word 3 = Address of word in which bits are to be tested.

The bits in the addressed memory word corresponding to logic 1's in the mask are tested. If all the bits tested are 1's, the instruction will not skip; otherwise the instruction will skip one word (that is, the P-Register will advance two counts from Word 3 of the instruction).

Word Manipulation Instructions. The following instructions facilitate the comparing and moving of word arrays.

CMW   **COMPARE WORDS**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	0
D/I															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Return if array 1 = array 2
Return if array 1 < array 2
Return if array 1 > array 2

Compares the words in array 1 with those in array 2. This is a three-word instruction where:

Word 1 = Instruction code,

Word 2 = Address of word containing the word count, and

Word 3 = All-zeros word reserved for use by microcode.

The operand addresses are in the A- and B-Registers. The A-Register contains the first word address of array 1 and the B-Register contains the first word address of array 2. Bit 15 of the addresses in the A- and B-Registers are ignored; that is, no indirect addressing allowed.

The number of words to be compared is given in the memory location addressed by Word 2 of the instruction; the number of words to be compared is restricted to a positive integer greater than zero. The arrays are compared one word at a time; the ith word in array 1 is compared with the ith word in array 2. This comparison is performed arithmetically; that is, each word is considered a two's complement number. If all words in array 1 are equal to all words in array 2, the "equal" exit is taken. As soon as two words are compared and found to be different, the "less than" or "greater than" exit is taken, depending on whether the word in array 1 is less than or greater than the word in array 2. The three ways this instruction exits are as follows:

1. No skip if array 1 is equal to array 2; the P-Register advances one count from Word 3 of the instruction. The A-Register contains its original value incremented by the word count stored in the address specified in Word 2.

2. Skips one word if array 1 is less than array 2; the P-Register advances two counts from Word 3 of the instruction. The A-Register contains the address of the word in array 1 where the comparison stopped.

3. Skips two words if array 1 is less than array 2; the P-Register advances three counts from Word 3 of the instruction. The A-Register contains the address of the word in array 1 where the comparison stopped.

For all three exits, the B-Register will contain its original value incremented by the word count stored in the address specified in Word 2. This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-Registers. During the interrupt, the remaining count is stored in Word 3 of the instruction. This instruction has undefined results if executed with CDS mode enabled.

MVW   **MOVE WORDS**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	1 1 1	1 1 1
D/I					
0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0

Moves words in a left-to-right manner; that is, the word having the lowest address in the source is moved first. This is a three-word instruction where:

Word 1 = Instruction code,

Word 2 = Address of word containing the count, and

Word 3 = All-zeros word reserved for use by microcode.

The operand addresses are in the A- and B-Registers. The A-Register contains the first word address source and the B-Register contains the first word address destination. The number of words to be moved is a 16-bit positive integer greater than zero addressed by Word 2 of the instruction. The word addresses in the A- and B-Registers are incremented as each word is being moved. Thus, at the end of the operation, the A- and B-Registers are incremented by the number of words moved.

Wraparound of the word address would result from a carry into bit position 15 (that is, at 32676). If the destination address became 000 or 001, the next word would be moved into the A- or B-Register and destroy the proper word addresses for the move operation. For each word move, a memory protect check is performed.

This instruction is interruptible. The interrupt routine is expected to save and restore the contents of the A- and B-Registers. During the interrupt, the remaining count is stored in Word 3 of the instruction. This instruction has undefined results if executed with CDS mode enabled.

Floating Point Instructions

The floating point instructions allow addition, subtraction, multiplication, and division of single precision floating point quantities, and conversion of quantities from floating point format to integer format or vice versa. The A400 has additional instructions to convert single precision floating point quantities to double integer and vice versa. Data formats are shown in Figure 3-1. Except for zero, all floating point operands must be normalized (that is, sign of mantissa differs from most significant bit of mantissa).

The execution times of the floating point instructions are specified below in Table 3-5. These instructions are noninterruptible except during indirect address resolution; any attempted interrupt is held off for the full execution time of the currently active floating point instruction. However, DMA operation is not held off.

Single Precision Operations. Overflow for single precision operations occurs if the result lies outside the range of representable single precision floating point numbers $[-2^{127},(1-2^{-23})2^{127}]$. In such a case, the overflow flag is set and the result $(1-2^{-23})2^{127}$ is returned to the A- and B-Registers. Underflow occurs if the result lies inside the range $[-2^{-129}(1+2^{-22}),-2^{-129}]$. In such a case, the overflow flag is set and the result 0 is returned to the A- and B-Registers.

FAD   FLOATING POINT ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
D/I															

Memory Address
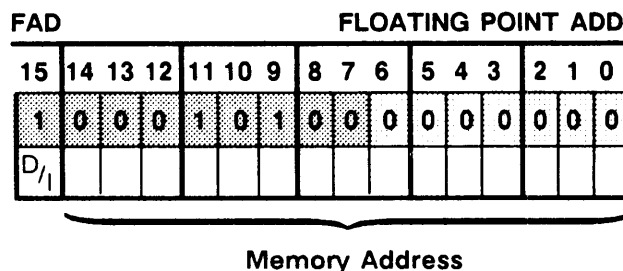
Adds the floating point quantity in the A- and B-Registers to the floating point quantity in the specified memory locations. The floating point result is returned to the A- and B-Registers.

FSB   **FLOATING POINT SUBTRACT**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0
D/I															

$\underbrace{}$

Memory Address
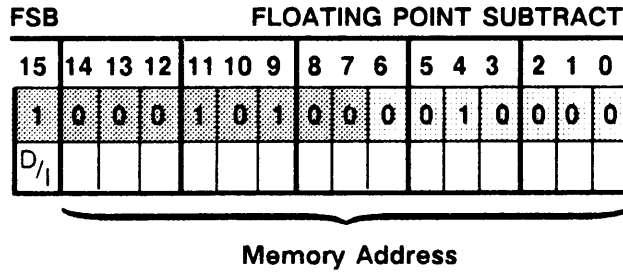
Subtracts the floating point quantity in the specified memory locations from the floating point quantity in the A- and B-Registers. The floating point result is returned to the A- and B-Registers.

FMP   **FLOATING POINT MULTIPLY**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0
D/I															

$\underbrace{}$

Memory Address
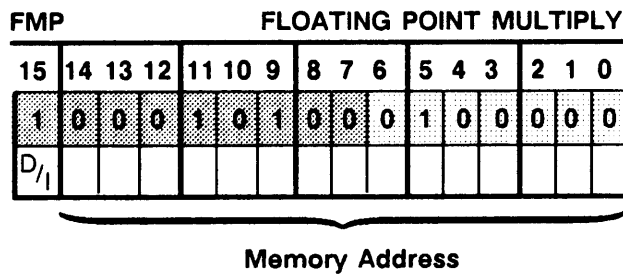
Multiplies the floating point quantity in the A- and B-Registers by the floating point quantity in the specified memory locations. The floating point result is returned to the A- and B-Registers.

FDV   **FLOATING POINT DIVIDE**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0
D/I															

$\underbrace{}$

Memory Address
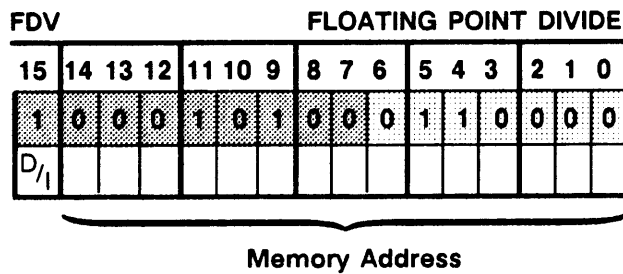
Divides the floating point quantity in the A- and B-Registers by the floating point quantity in the specified memory locations. The floating point result is returned by the A- and B-Registers.

FIX

FLOATING POINT TO SINGLE INTEGER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0

Converts the floating point quantity in the A- and B-Registers to single integer format. The integer result is returned to the A-Register. If the magnitude of the floating point number is <1, regardless of sign, the integer 0 is returned. If the magnitude of the exponent of the floating point number is ≥16, regardless of sign, the integer 32767 (077777 octal) is returned as the result and the overflow flag is set.

FLT

SINGLE INTEGER TO FLOATING POINT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	1	0	1	0	0	0	0

Converts the single integer quantity in the A-Register to single precision floating point format. The floating point result is returned to the A- and B-Registers.

.FIXD*

FLOATING POINT TO DOUBLE INTEGER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0

Converts the floating point quantity in the A- and B-Registers to double integer format. The integer result is returned to the A- and B-Registers. (The A-Register contains the most-significant word and the B-Register contains the least-significant word.) If the magnitude of the floating point number is <1, regardless of sign, the integer 0 is returned. If the magnitude of the floating point number is ≥32, regardless of sign, the integer $2^{23}-1$ is returned as the result and the overflow flag is set.

.FLTD*

DOUBLE INTEGER TO FLOATING POINT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0

Converts the double integer quantity in the A- and B-Registers to single precision floating point format. The floating point result is returned to the A- and B-Registers.

*Refer to the "Assembly Language" paragraph of this chapter.

Language Instruction Set

The Language Instruction Set consists of seventeen instructions that perform certain frequently used high-level language operations including parameter passing, array address calculations, and floating point conversion, packing, rounding and normalization operations.

For multiple-word instructions, indirect addressing to any number of levels is permitted for the word(s) indicated as a memory address. A logic 0 in bit position 15 specifies direct addressing; a logic 1 specified indirect addressing.

The following paragraphs provide machine language coding and definitions for the Language Instruction Set. Data formats are shown in Figure 3-1. For a more detailed description of instructions in the Language Instruction Set, refer to the *Relocatable Library Reference Manual*, HP part no. 92077-90037.

.ZFER*   **TRANSFER EIGHT WORDS**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	1	1	1	1	1
D/I															
D/I															
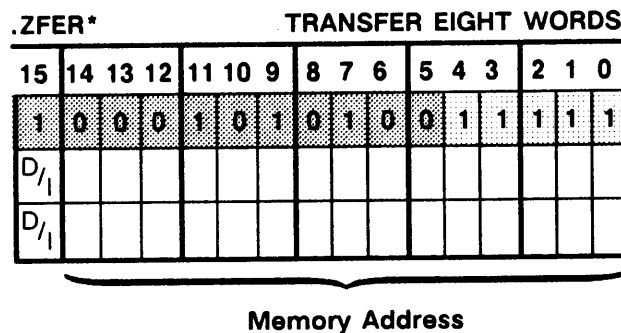
Memory Address

Transfers eight consecutive words from one memory location to another. The source address +8 is returned to the A-Register; the destination address +8 is returned to the B-Register. This is a three-word instruction where:

Word 1 = Instruction.

Word 2 = Destination address.

Word 3 = Source address.

Wraparound of either address produces undefined results. Under CDS, the source and/or destination address may be adjusted for base relativity.

*Refer to the "Assembly Language" paragraph of this chapter.

TRANSFER COMPLEX OR DOUBLE FLOATING POINT

.CFER*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	1	1	0	0	1
D/I															
D/I															
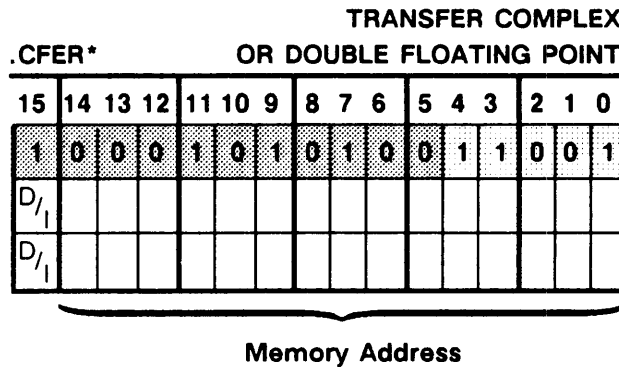
Memory Address

Transfers a double precision floating point quantity (four consecutive words) from one memory location to another. The source address +4 is returned to the A-Register; the destination address +4 is returned to the B-Register. This is a three-word instruction where:

Word 1 = Instruction.

Word 2 = Destination address.

Word 3 = Source address.

Wraparound of either address produces undefined results. Under CDS, the source and/or destination address may be adjusted for base relativity.

TRANSFER THREE CONSECUTIVE WORDS

.DFER*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	0	0	1	0	1
D/I															
D/I															

Memory Address

Transfers three consecutive words from one memory location to another. The source address +3 is returned to the A-Register; the destination address +3 is returned to the B-Register. This is a three-word instruction where:

Word 1 = Instruction.

Word 2 = Destination address.

Word 3 = Source address.

Wraparound of either address produces undefined results. Under CDS, the source and/or destination address may be adjusted for base relativity.

*Refer to the "Assembly Language" paragraph of this chapter.

TRANSFER PARAMETER ADDRESSES

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 1 0	1 0 0

Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point. There must be exactly two words between the subroutine entry point and the .ENTP instruction. A true address is determined by eliminating all indirect references. The true return address is returned to the A-Register. Used for privileged or re-entrant subroutines. This instruction has undefined results if executed with CDS mode enabled.

NOTE

When calling a subroutine that uses .ENTR, do not pass a parameter that is located in the subroutine's .ENTR parameter area. This can cause indeterminate results. For example:

WRONG	RIGHT
jsb sub	jsb sub
def *+2	def *+2
def parm	def parm
.	.
.	parm nop
.	.
.	.
parm nop	parm1 nop
sub nop	sub nop
jsb .entr	jsb .entr
def parm	def parm1

TRANSFER PARAMETER ADDRESSES

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 1 0	0 1 1

Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point. A true address is determined by eliminating all indirect references. No more than three levels of indirect addressing are allowed per parameter. This instruction has undefined results if executed with CDS mode enabled.

*Refer to the "Assembly Language" paragraph of this chapter.

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 1 0	0 0 0

Transfers three consecutive words from one memory location to another. The A-Register must contain the source address and the B-Register must contain the destination address. The source address +3 is returned to the A-Register; the destination address +3 is returned to the B-Register. Wraparound of either address produces undefined results. Under CDS, the source and/or destination addresses may be adjusted for base relativity.

.SETP*   SET A TABLE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	1	0	1	1	1
0	Address where Count is given													

Sets a table of increasing numbers in consecutive memory locations. The A-Register must contain the initial number and the B-Register must contain the initial memory address (direct only); the succeeding memory location must give the address where the number of memory locations (count $\geq$ 0) is given. Entries in the table are established by incrementing the initial address and number by one (1) for each successive entry until the last number, initial number, the initial address+COUNT and the A-Register equals the initial value+COUNT. Wraparound will produce undefined results. This instruction is interruptible. On return the B-Register equals the initial address +COUNT. Under CDS, the memory addresses may be adjusted for base relativity.

NOTE

If the initial address +COUNT −1 results in an address which is beyond the end of logical memory, addresses within the base page are destroyed.

COMPLEMENT AND NORMALIZE
SINGLE FLOATING POINT
..FCM*

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 1 1	0 1 0

*Refer to the "Assembly Language" paragraph of this chapter.

Complements and normalizes in place a packed single precision floating point quantity located in the A- and B-Registers. The result is returned to the A- and B-Registers.

SINGLE FLOATING POINT TO DOUBLE FLOATING POINT

.BLE*

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 0 0	1 1 1
D/I					
D/I					
D/I					

Memory Address
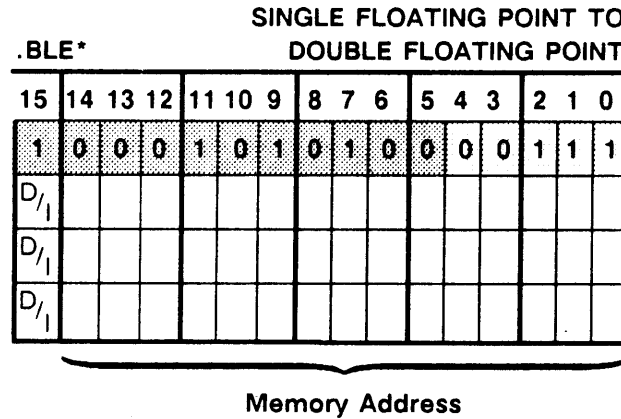
Converts the single precision floating point quantity in specified memory locations to a double-precision floating point quantity. The result is returned to other specified memory locations. This is a four-word instruction where:

Word 1 = Instruction code.

Word 2 = Return address.

Word 3 = Address of result.

Word 4 = Address of operand.

UNPACK FLOATING POINT QUANTITY

.FLUN*

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 1 0	1 1 0
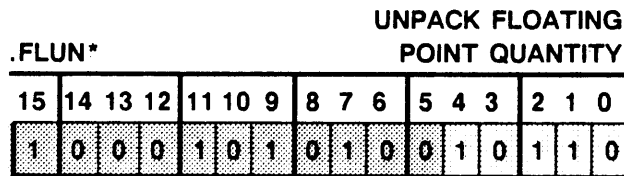
Unpacks a floating point quantity. The lower part of the floating point quantity must be in the B-Register. The exponent is returned to the A-Register, the lower part of the mantissa is returned to the B-Register.
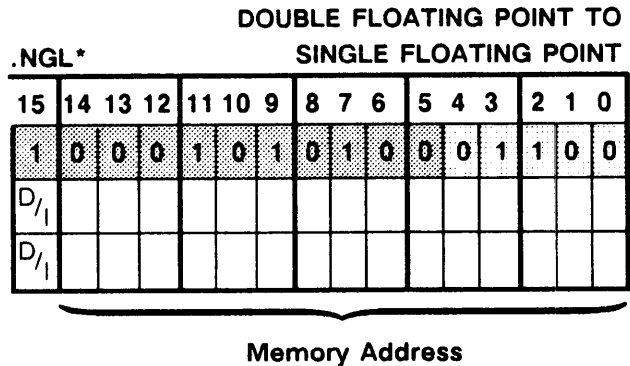
*Refer to the "Assembly Language" paragraph of this chapter.

DOUBLE FLOATING POINT TO SINGLE FLOATING POINT

.NGL*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	0	1	1	0	0
D/I															
D/I															

Memory Address

Converts the double precision floating point quantity in the specified memory locations to a single precision floating point quantity. The result is placed in the A- and B-Registers. Overflow is cleared unless, during execution, rounding results in overflow or underflow of the exponent, in which case overflow is set and the result is truncated to the greatest positive number. This is a three-word instruction where:

Word 1 = Instruction code.

Word 2 = Return address.

Word 3 = Address of operand.

NORMALIZE FLOATING POINT QUANTITY

.PACK*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0
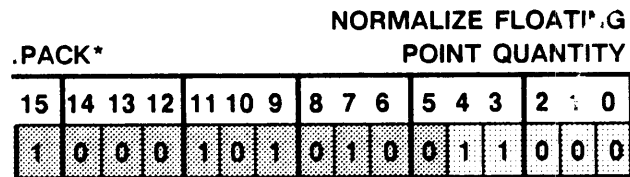
Converts the signed mantissa of a floating point quantity into a normalized format. The floating point quantity must be in the A- and B-Registers. The succeeding instruction must reserve one word of memory for temporary storage of the exponent. The first word of the two-word floating point result is returned to the A-Register; the second word, to the B-Register.

X TIMES 2 TO THE POWER N

.PWR2*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	1	0	1	0	1
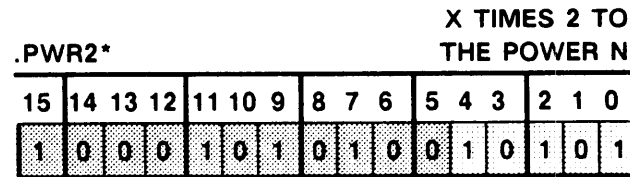
Calculates for floating point x and integer n: $y = x * 2^n$. The floating point quantity must be in the A- and B-Registers; the succeeding instruction must define integer n. The first word of the two-word floating point result is returned to the A-Register; the second word, to the B-Register.

*Refer to the "Assembly Language" paragraph of this chapter.

..TCM* NEGATE DOUBLE FLOATING POINT

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 1 1	0 1 1
D/I					
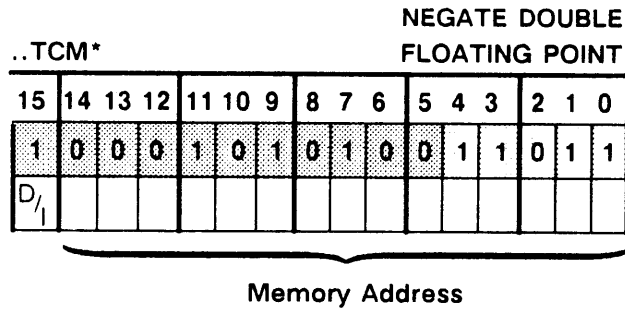
Memory Address

Negates a packed double precision floating point quantity located in the specified memory locations. The result is returned to the same specified memory locations.
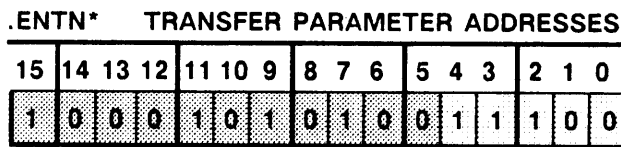
.ENTN* TRANSFER PARAMETER ADDRESSES

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 1 1	1 0 0

Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point. The return address stored in the SUB entry point references the word following the last parameter DEF in the calling routine. A true address is determined by eliminating all indirect references. This instruction has undefined results if executed with CDS mode enabled.
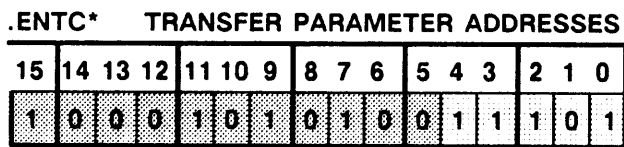
.ENTC* TRANSFER PARAMETER ADDRESSES

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 0	0 1 1	1 0 1

Transfers the true addresses of parameters from a calling sequence into a subroutine; adjusts return address to the true return point. The return address stored in the SUB entry point references the word following the last parameter DEF in the calling routine. There must be exactly two words between the subroutine entry point and the .ENTC instruction. A true address is determined by eliminating all indirect references. The true return address is returned to the A-Register. Used for privileged or re-entrant subroutines. This instruction has undefined results if executed with CDS mode enabled.

*Refer to the "Assembly Language" paragraph of this chapter.

.CPM*   SINGLE INTEGER ARITHMETIC COMPARE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	1	1	1	1	0
D/I															
D/I															

Return if operand 1 = operand 2
Return if operand 1 < operand 2
Return if operand 1 > operand 2

Arithmetically compares operands addressed by second and third word. Does not skip if operands are equal; however, skips one instruction if the first operand is less than the second, or skips two instructions if the first operand is greater than the second.

Double Integer Instructions

The double integer instructions allow arithmetic and test operations on 32-bit integer quantities. The data format for double integer values is shown in Figure 3-1. Double integer values contained in the (A,B) registers have the most significant bits in the A-Register. Values stored in memory require two locations. The operand address in a double integer instruction points to the first memory location, which contains the most significant bits. Double integer instructions clear the overflow register upon entry, and will set the O-Register if an overflow occurs. The E-Register is never cleared by a double integer instruction.

.DAD*   DOUBLE INTEGER ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0
D/I															

Memory Address

Performs the double integer operation:

(A,B) = (A,B) + <OPND>

The contents of <OPND> are unaltered. In the event of overflow, the overflow bit is set and the returned

*Refer to the "Assembly Language" paragraph of this chapter.

result contains the lower 32 bits of the actual sum, in unsigned form. The extend bit will be set if an unsigned carry out of the A-Register occurs.
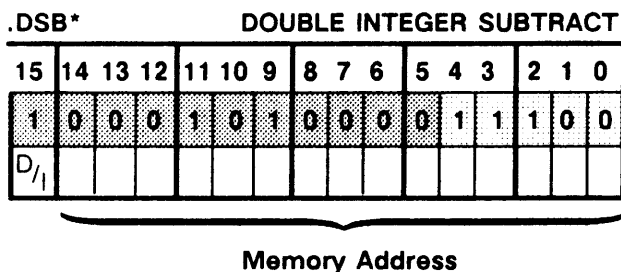
.DSB*				DOUBLE INTEGER SUBTRACT											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	1	1	1	0	0
D/I															

Memory Address

Performs the double integer operation:

$$(A,B) = (A,B) - <OPND>$$

The contents of <OPND> are unaltered. In the event of overflow, the overflow bit is set and the returned result contains the lower 32 bits of the actual difference, in unsigned form. The extend bit will be set if an unsigned borrow out of the A-Register occurs.
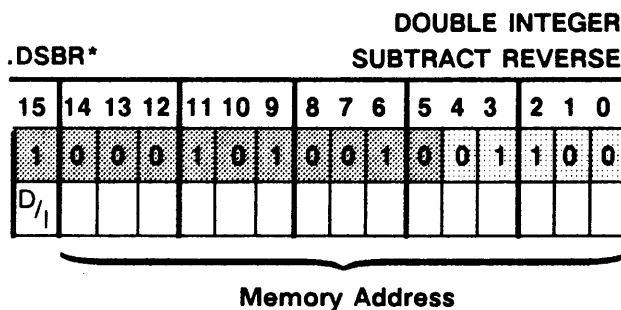
.DSBR*				DOUBLE INTEGER SUBTRACT REVERSE											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	1	0	0	1	1	0	0
D/I															

Memory Address

Performs the double integer operation:

$$(A,B) = <OPND> - (A,B)$$

The contents of <OPND> are unaltered. In the event of overflow, the overflow bit is set and the returned result contains the lower 32 bits of the actual difference, in unsigned form. The extend bit will be set if an unsigned borrow occurs.

*Refer to the "Assembly Language" paragraph of this chapter.
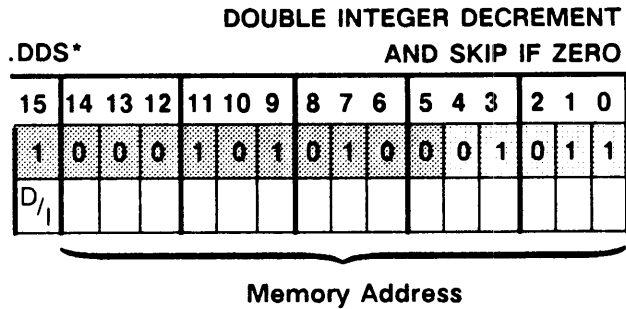
DOUBLE INTEGER DECREMENT
.DDS* — AND SKIP IF ZERO

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	0	1	0	1	1
D/I															

Memory Address

Performs the double integer operation:

<OPND> = <OPND> - 1

If the new value of <OPND> equals zero, the next instruction will be skipped. The value in <OPND> is treated as an unsigned number, and a borrow out of the <OPND> is ignored.

.DNG* — DOUBLE INTEGER NEGATE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	0	0	0	1	1
D/I															

Memory Address
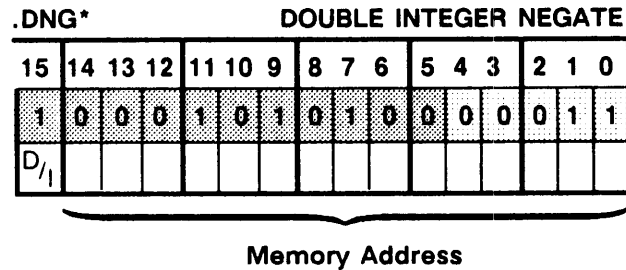
Performs the double integer operation:

(A,B) = - (A,B)

An input value of (100000,000000) is left unchanged and overflow is set. An input value of zero will cause the extend bit to be set.

.DCO* — DOUBLE INTEGER COMPARE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0
D/I															

Memory Address
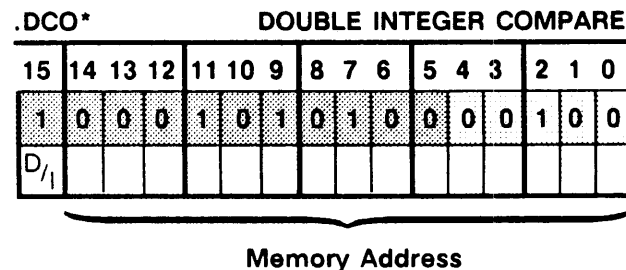
Compares the double integers (A,B) and <OPND>

If (A,B) = <OPND> Return to P+2
If (A,B) < <OPND> Return to P+3
If (A,B) > <OPND> Return to P+4

where P is the address of the .DCO instruction. The value of both double integers and the overflow bit are unaltered.

*Refer to the "Assembly Language" paragraph of this chapter.

.DIN*   DOUBLE INTEGER INCREMENT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0

Performs the double integer operation:

$$(A,B) = (A,B) + 1$$

An input value of (077777, 177777) will return a result of (100000, 000000) and set overflow. An input value of (177777, 177777) will return a result of zero and cause the extend bit to be set.

.DDE*   DOUBLE INTEGER DECREMENT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	1

Performs the double integer operation:

$$(A,B) = (A,B) - 1$$

An input value of (100000, 000000) will return a result of (077777, 177777) and set overflow. An input value of zero will return a result of (177777, 177777) and cause the extend bit to be set.

DOUBLE INTEGER INCREMENT
.DIS*   AND SKIP IF ZERO

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	0	0	1	0	1	0
D/I															

Memory Address

Performs the double integer operation:

$$<OPND> = <OPND> + 1$$

If the new value of <OPND> equals zero, the next instruction will be skipped. The value in <OPND> is treated as an unsigned number, and a carry out of the <OPND> is ignored.

*Refer to the "Assembly Language" paragraph of this chapter.

.DDI* **DOUBLE INTEGER DIVIDE**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
D/I															

Memory Address

Performs the double integer operation:

(A,B) = (A,B) ÷ <OPND>

The contents of <OPND> are unaltered. If overflow or divide by zero occurs, the result (077777,177777) is returned and overflow is set.

.DDIR* **DOUBLE INTEGER DIVIDE REVERSE**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	1	0	1	1	1	0	0
D/I															

Memory Address

Performs the double integer operation:

(A,B) = <OPND> ÷ (A,B)

The contents of <OPND> are unaltered. If overflow or divide by zero occurs, the result (077777,177777) is returned and overflow is set.

.DMP* **DOUBLE INTEGER MULTIPLY**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	1	0	1	1	0	0
D/I															

Memory Address

Performs the double integer operation:

(A,B) = (A,B) x <OPND>

The contents of <OPND> are unaltered. If overflow occurs, the result (077777,177777) is returned and overflow is set.

*Refer to the "Assembly Language" paragraph of this chapter.

Virtual Memory Instructions

The Virtual Memory Instructions perform accesses to virtual memory and the extended area, which are extensions of logical memory. If an addressed data item is in physical memory, the instructions perform the required mapping, including modification of map registers and entry of the appropriate page numbers into the user's logical address space. If an addressed data item is not in physical memory, a fault is generated to a macrocode routine which swaps the data from the disc into physical memory and then restarts the VMA instruction. The fault sequence generated depends on whether the CDS mode is enabled. If CDS mode is disabled, a JSB,I through memory location 04 in the user map is effected. Memory location 04 is expected to contain the address of the entry point of the VMA fault-handler in the user space (indirect addressing is not allowed). If CDS mode is enabled, an interrupt is generated to trap cell 12 octal in the system map. As the VAM fault interrupt is the lowest priority interrupt, any other pending interrupts will be serviced first.

NOTE

VMA always maps the page that the requested VMA address is on in addition to the next page, ensuring that entire data items up to 1k words in size are mapped-in. The exception to this is .PMAP, which only maps-in the requested page.

.PMAP*   MAP SPECIFIED PAGE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0

| Error return |
| Normal return |

On entry, the A-Register is loaded with the number of the user-map register to be altered and the B-Register is loaded with the page ID, which are the parameters passed to the routine. If an attempt is made to map in the last+1 page, that PMR is mapped read and write protected. When no error occurs, a normal return occurs to the second word after the instruction; mapping is complete; and the contents of the A- and B-Registers are incremented. If a fault occurs and the sign bit is set in the A-Register, an error return to the word that follows the instruction occurs. If a fault occurs, and the sign bit is not set in the A-Register, a normal fault sequence is generated. The O-,X-, and Y-Registers are undefined. The E-Register is set if an attempt was made to map the last+1page; otherwise it is cleared.

*Refer to the "Assembly Language" paragraph of this chapter.

The .PMAP instruction uses the last user page (31) of memory and then maps that logical page read and write protected. After a .PMAP call, memory references to address >75777 octal will cause memory protect violations.

.IMAP*   16-BIT SUBSCRIPT MAPPING

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0

D/I

Word 2 = DEF dope vector

Word 3 = Subscript N

⋮

Word N+2 = Subscript 1

Performs a subscript calculation and maps the result into logical memory. Each of the subscripts and dimensions are 16-bit integers. However, the calculation uses 32-bit adds and multiplies. The subscripts are sign-extended to 32 bits. The subscript words cannot address the A- or B-Register.

Word 2 points to a table that specifies in order the number of dimensions, dimension sizes, the number of words per element, and a two-word offset.

On a normal return, the A-, X-, Y-, E- and O-Registers are undefined and the B-Register contains the logical address.

.IRES*   16-BIT SUBSCRIPT RESOLUTION

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0

D/I

Word 2 = DEF dope vector

Word 3 = Subscript N

⋮

Word N+2 = Subscript 1

Performs a subscript calculation. Each of the subscripts and dimensions are 16-bit integers. However, the calculation uses 32-bit adds and multiplies. The subscripts are sign-extended to 32 bits. The subscript words cannot address the A- or B-Register.

Word 2 points to a table that specifies in order the number of dimensions, dimension sizes, the number of words per element, and a two-word offset.

*Refer to the "Assembly Language" paragraph of this chapter.

On a normal return, the A- and B-Registers contain the address of the array element in double-integer format (most significant word in the A-Register).

.JMAP*   32-BIT SUBSCRIPT MAPPING

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0

D/I	
Word 2 = DEF dope vector	
Word 3 = Subscript N	
Word N+2 = Subscript 1	

Performs a subscript calculation and maps the result into logical memory. Each of the subscripts and dimensions are 16-bit integers, and the calculation uses 32-bit adds and multiplies. The subscripts are sign-extended to 32 bits. The subscript words cannot address the A- or B-Register.

Word 2 points to a table that specifies the number of dimensions, dimension sizes, the number of words per element, and a two-word offset.

On a normal return, the A-, X-, Y-, E-, and O-Registers are undefined and the B-Register contains the logical address.

.JRES*   32-BIT SUBSCRIPT RESOLUTION

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1

D/I	
Word 2 = DEF dope vector	
Word 3 = Subscript N	
Word N+2 = Subscript 1	

Performs a subscript calculation. Each of the subscripts and dimensions are 16-bit integers, and the calculation uses 32-bit adds and multiplies. The subscript words cannot address the A- or B-Register.

Word 2 points to a table that specifies the number of dimensions, dimension sizes, the number of words per element, and a two-word offset.

On a normal return, the A- and B-Registers contain the address of the array element in double-integer format (most significant word in the A-Register).

*Refer to the "Assembly Language" paragraph of this chapter.
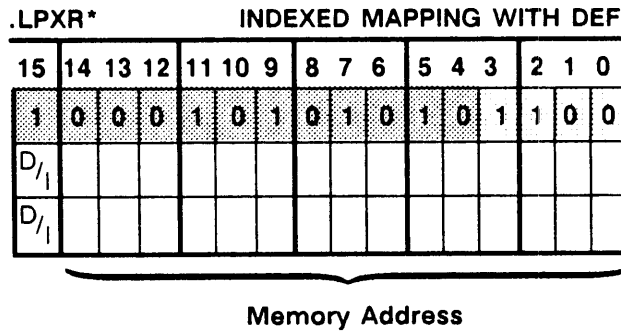
.LPXR*   INDEXED MAPPING WITH DEF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	1	1	0	0
D/I															
D/I															

Memory Address

On entry, the pointer specified by the second instruction word is resolved, and the double word it points to is loaded into the A- and B-Registers. The offset specified in the third instruction word is resolved, and the double word it points to is added to the contents of the A- and B-Registers. The result is treated as a 26-bit VMA pointer and is mapped. On exit, the B-Register contains the logical address of the data item, and the A-, X-, Y-, E-, and O-Registers are undefined. The offset word cannot refer to the A- or B-Register.

.LPX*   INDEXED MAPPING WITH REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	1	1	0	1
D/I															

Memory Address
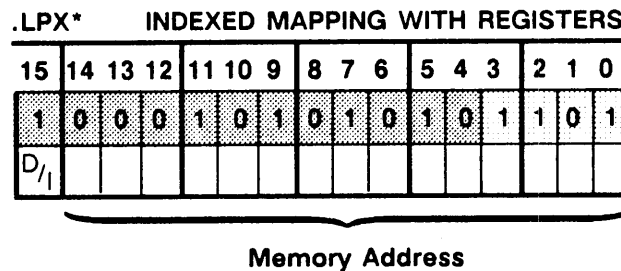
On entry, the second instruction word either directly or indirectly points to a double integer in memory, which is to be added to the double integer in the A- and B-Registers to form a double-word VMA pointer. If bit 15 of the A-Register is set, the B-Register contains the address of a data item presently residing in logical memory and the .LPX instruction does nothing; otherwise, the data item is mapped. On exit, the B-Register contains the logical address of the data item, and the A-, X-, Y-, E-, and O-Registers are undefined.

*Refer to the "Assembly Language" paragraph of this chapter.

.LBPR* MAPPING WITH DEF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	1	1	1	0
D/I															

Memory Address

On entry, the pointer specified by the second instruction word is resolved and the double word it points to is loaded into the A- and B-Registers. This value is treated as a 26-bit VMA pointer and is mapped. On exit, the B-Register contains the logical address of the data item, and the A-, X-, Y-, E-, and O-Registers are undefined.

.LBP* MAPPING WITH REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1

On entry, the 26-bit VMA pointer is contained in the A-Register (most significant word) and B-Register; if bit 15 of the A-Register is set, the B-Register contains the address of a data item presently residing in logical memory; otherwise, the data item is mapped. On exit, the B-Register contains the logical address of the data item, and the A-, X-, Y-, E-, and O-Registers are undefined.

Operating System Instruction Set

The operating system instructions provide instructions for ascertaining the CPU and firmware identification, and instructions for interrupt conditions.

.CPUID* PROCESSOR IDENTIFICATION

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0

The A-Register is loaded with a number that identifies the type of processor installed in the computer system, where:

Octal 2 = A600
Octal 3 = A700
Octal 4 = A900
Octal 5 = A600+
Octal 7 = A400

*Refer to the "Assembly Language" paragraph of this chapter.

.FWID*   **FIRMWARE IDENTIFICATION**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 1	0 0 0	0 0 1

On exit, the A-Register contains a number that identifies the revision code of the firmware.

.WFI*   **WAIT FOR INTERRUPT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 1	0 0 0	0 1 0

This instruction is equivalent to a JMP* except that the processor does not perform memory accesses, which would decrease the effective bandwidth of the memory backplane. This instruction is interruptible.

.SIP*   **SKIP IF INTERRUPT PENDING**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 1 1	0 0 0	0 1 1

The processor skips if an I/O interrupt is pending (INTRQ- is asserted on the A-series backplane), which is independent of the Level 2 and Level 3 interrupt masks.

Execution Times

Table 3-5 lists the execution times required for the various base set instructions. Table 3-6 lists the execution times required for double precision floating point instructions.

Double-Precision Floating Point Instructions

The double-precision floating point instructions are standard in the base set and provide for add, subtract, multiply and divide operations on a double-precision value, as well as instructions that convert double-precision floating point values to or from single and double integer fixed values.

Overflow for double precision operations occurs if the result lies outside the range of representable double precision floating point numbers $[-2^{127},(1-2^{-55})2^{-127}]$. In such a case, the overflow flag is set and $(1-2^{-55})2^{127}$ is returned as the result. Underflow occurs if the result lies inside the range $[-2^{-127}(1+2^{-54}),2^{-127}]$. In such a case, the overflow flag is set and 0 is returned as the result.

*Refer to the "Assembly Language" paragraph of this chapter.

.TADD*   DOUBLE FLOATING
                                              POINT ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
D/I															
D/I															
D/I															

Memory Address
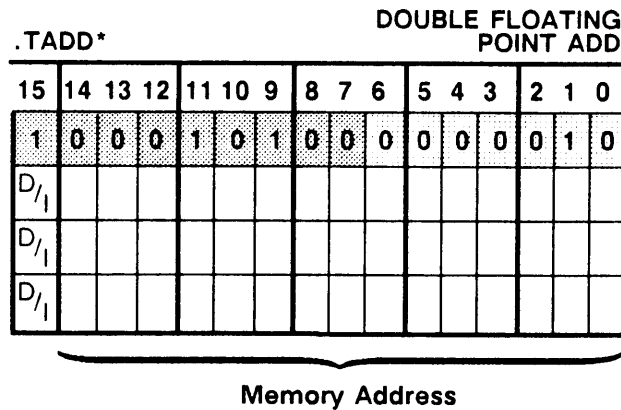
Adds two double precision floating point quantities (augend plus addend). This is a four-word instruction where:

Word 1 = Instruction code.
Word 2 = Address of result.
Word 3 = Address of augend.
Word 4 = Address of addend.

.TDIV*   DOUBLE FLOATING
                                             POINT DIVIDE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0
D/I															
D/I															
D/I															

Memory Address
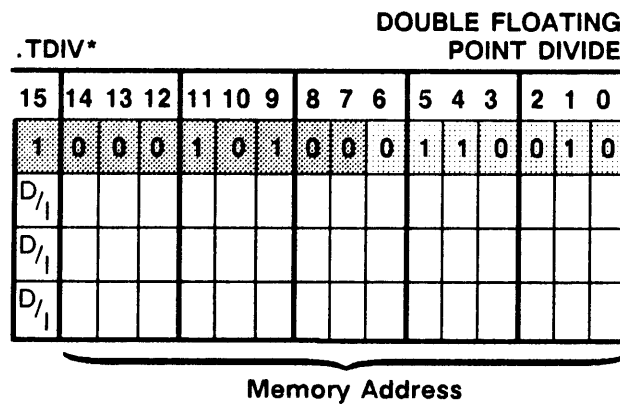
Divides one double precision floating point quantity by another (dividend by divisor). This is a four-word instruction where:

Word 1 = Instruction code.
Word 2 = Address of result.
Word 3 = Address of dividend.
Word 4 = Address of divisor.

*Refer to the "Assembly Language" paragraph of this chapter.

.TFTD* **DOUBLE INTEGER TO DOUBLE FLOATING POINT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 0 1	0 1 0	1 1 0
D/I					

Memory Address
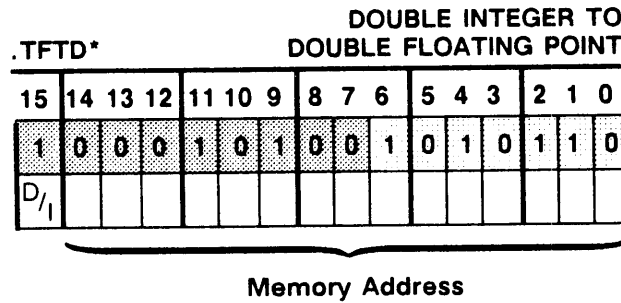
Converts the double integer quantity in the A- and B-Registers to double precision floating point format. The floating point result is returned to the specified memory locations.

.TFTS* **SINGLE INTEGER TO DOUBLE FLOATING POINT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 0 1	0 1 0	0 1 0
D/I					

Memory Address
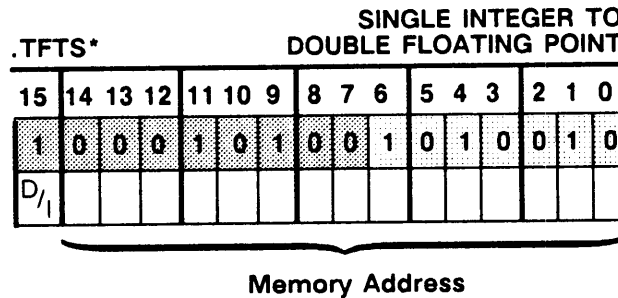
Converts the single integer quantity in the A-Register to double precision floating point format. The floating point result is returned to the specified memory locations.

.TFXD* **DOUBLE FLOATING POINT TO DOUBLE INTEGER**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	0 0 1	0 0 0	1 1 0
D/I					

Memory Address
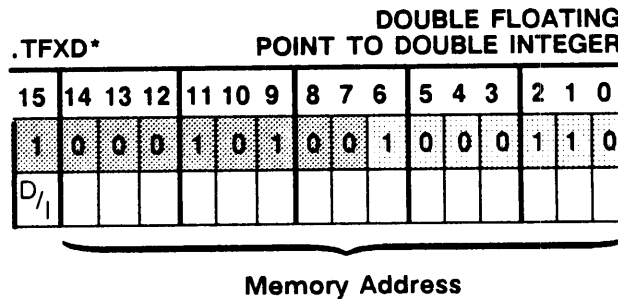
Converts the double precision floating point quantity in the specified memory locations to double integer format. The integer result is returned to the A- and B-Registers. (The A-Register contains the most-significant word and the B-Register contains the least-significant word.) If the magnitude of the floating point number is <1, regardless of sign, 0 is returned as the result. If the magnitude of the exponent of the floating point number is ≥ 32, regardless of sign, the integer $2^{31}-1$ is returned as the result and the overflow flag is set.

*Refer to the "Assembly Language" paragraph of this chapter.
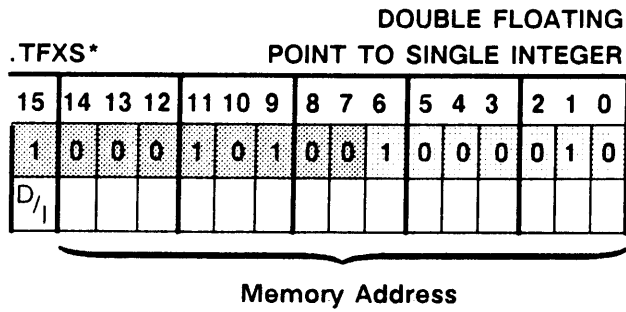
.TFXS* — DOUBLE FLOATING POINT TO SINGLE INTEGER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0
D/I															

Memory Address

Converts the double precision floating point quantity in the specified memory locations to single integer format. The integer result is returned to the A-Register. If the magnitude of the floating point number is <1, regardless of sign, 0 is returned as the result. If the magnitude of the exponent of the floating point number is ≥ 16, regardless of sign, the integer $2^{15}-1$ is returned as the result and the overflow flag is set.

.TMPY* — DOUBLE FLOATING POINT MULTIPLY

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0
D/I															
D/I															
D/I															

Memory Address
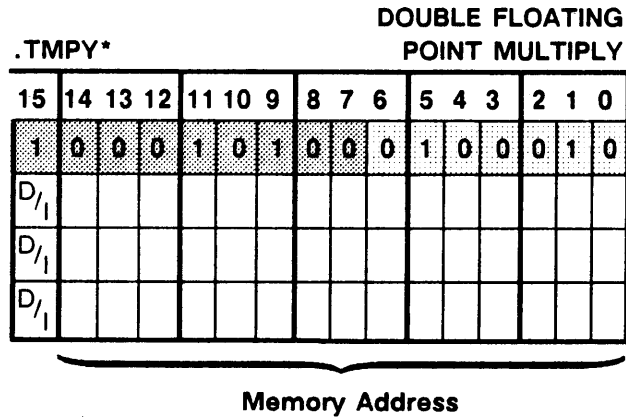
Multiplies one double precision floating point quantity by another (multiplicand by multiplier). This is a four-word instruction where:

Word 1 = Instruction code.

Word 2 = Address of result.

Word 3 = Address of multiplicand.

Word 4 = Address of multiplier.

*Refer to the "Assembly Language" paragraph of this chapter.

.TSUB*

DOUBLE FLOATING
POINT SUBTRACT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0
D/I															
D/I															
D/I															

Memory Address
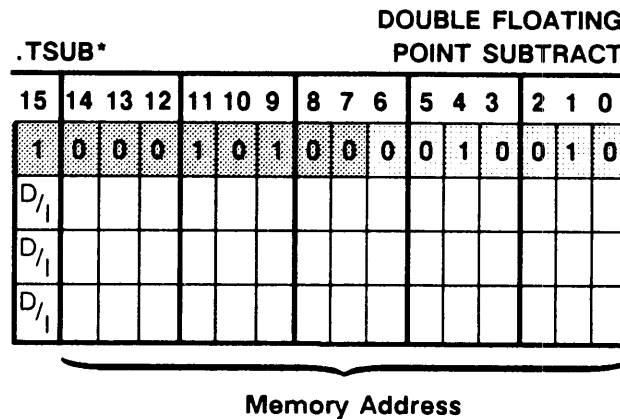
Subtracts one double precision floating point quantity from another (minuend from subtrahend). This is a four-word instruction where:

Word 1 = Instruction code.
Word 2 = Address of result.
Word 3 = Address of minuend.
Word 4 = Address of subtrahend.

Assembly Language

New instructions not recognized by the HP Macroassembler require different handling in HP Assembly Language programming. These instructions are asterisked in the preceding paragraphs and must be used in the form: JSB x where x is the instruction. (The instruction, x, must be declared as an external at the beginning of the assembly language program.) Most of these instructions correspond to library subroutines** and must be implemented into HP RTE systems (as described in the following paragraph) to enable their execution in firmware instead of software.

*Refer to the "Assembly Language" paragraph of this chapter.
**Refer to the *Relocatable Library Reference Manual*, HP part no. 92077-90037

RTE Implementation

New instructions are implemented in an RTE-A system by changing library entry points during the parameter input phase of system generation. (Refer to the appropriate RTE manual for the system generation procedures.) Using the list of entry point opcodes given in Table 3-7, make the entry changes as indicated below:

```
LPMR,RP,105700
SMPR,RP,105701
    .
    .
    .
.ADQB,RP,105413
```

Alternatively, entry points may be changed by loading (via LINK) a "replacement" module when user programs are loaded. Opcode replacement modules RPL60 through RPL63 are included in the RTE-A system.

Table 3-5. Typical Base Set Instruction Execution Times

INSTRUCTION	EXECUTION TIME (μsec)
Memory Reference Group	
(Direct)	
LDA/B, STA/B	1.0
ADA/B, IOR, XOR, AND	1.0
CPA/B	1.5
ISZ without skip	1.5
with skip	1.75
JSB	1.5
JMP	0.75
(One Indirect)	
LDA/B, STA/B	1.5
ADA/B, IOR, XOR, AND	1.5
CPA/B	2.0
ISZ	2.0
JSB	1.5
JMP	1.5
(Each Additional Indirect)	0.5
Alter/Skip Group	0.75 to 2.25
Shift/Rotate Group	0.75 to 3.00
Extended Arithmetic Group	
DLD	2.5
DST	2.25
MPY	6.00
DIV	8.5 to 9.5
ASL	1.75 plus 0.50/shift
ASR, LSL, LSR, RRL, RRR	1.75 plus 0.25/shift
Input/Output Group	
HLT	18.75
By select code:	
SC0: CLF	4.5
STF	5.75
SFC, SFS	
without skip	4.25
with skip	4.75
LIA/B	6.65
OTA/B	6.00
CLC	13.65

Table 3-5. Typical Base Set Instruction Execution Times (Continued)

INSTRUCTION	EXECUTION TIME (μsec)
Input/Output Group (Continued)	
SC1: CLF,STF	2.0
SFC, SFS	
without skip	2.5
with skip	2.75
LIA/B	16.25
OTA/B	2.75
SC2: STF	4.75
CLF	4.00
SFC, SFS	
without skip	4.5
with skip	4.75
LIA/B	6.75
OTA/B	6.0
STC	3.5
SC3: LIA/B	6.75
OTA/B	6.00
SC4: SFC, SFS	
without skip	2.75
with skip	3.00
LIA/B	2.75
OTA/B	3.00
CLC	3.5
STC	3.5
SC5: SFC, SFS	
without skip	3.00
with skip	3.50
STF	2.50
CLF	2.75
LIA/B	3.00
CLC, STC	2.75
SC6: SFC, SFS	
without skip	3.50
with skip	3.75
STF, CLF	5.00
CLC	4.5
STC	3.00
SC7: STC	3.00
LIA/B	3.25

Table 3-5. Typical Base Set Instruction Execution Times (Continued)

INSTRUCTION	EXECUTION TIME (μsec)
Input/Output Group (Continued)	
SC20 and up:	
CLC, CLF, STC, and STF	3.50
SFC, SFS	
without skip	3.50
with skip	5.25
LIA/B	6.00
MIA/B	6.00
OTA/B	5.25
Extended Instruction Group	
(Index Register Instructions)	
ADX, ADY	1.75
CXA, CXB, CYA, CYB	0.75
DSX, DSY	1.25
XAX, XBX, XAY, XBY	1.25
STX, STY	1.75
LDX, LDY	1.75
CAX, CBX, CAY, CBY	0.75
LAX, LBX, LAY, LBY	2.25
SAX, SBX, SAY, SBY	2.25
ISX, ISY	1.25
JLY, JPY	1.75
Per each indirect address level	0.50
JLA, JLB	1.75
(Bit Manipulation Instructions)	
CBS, SBS, TBS	3.5
(Word Manipulation Instructions)	
MVW	3.75 plus 2.33/byte
CMW	3.75 plus 1.25/word
(Byte Manipulation Instructions)	
LBT	1.75
SBT	2.50
MBT, CBT	4.25 plus 2.33/byte
SFB (Exit on location of byte	1.50 plus 1.50/byte

Table 3-5. Typical Base Set Instruction Execution Times (Continued)

INSTRUCTION	EXECUTION TIME (μsec)
Floating Point Group	
(Single Precision)	
FLT	1.75 to 6.50
FIX	1.50 to 6.50
FAD, FSB	7.75 to 26.00
FMP	13.775 to 25.25
Language Instruction Set	
.ENTR	4.75
.ENTP	5.25
.ENTN	3.5
.ENTC	4.25
per parameter (no indirect)	1.00
per indirect address level	1.00
.SETP (interruptible)	3.25
per table entry	0.50
.XFER	4.5
.DFER	5.75
.CFER	7.00
.ZFER	11.00
.CPM	3.50 to 4.25
..FCM	1.50 to 6.50
.NGL	5.9 to 5.9
.BLE	5.2 to 6.8
..TCM	7.0 to 8.0
.FLUN	1.6 to 1.6
.PACK	8.0 to 9.9
.PWR2	3.6 to 3.9
Double Integer Instructions	
.DAD	2.50 to 3.50
.DSB, DSBR	3.250 to 3.50
.DNG, DIN, DDE	1.75
.DCO	3.25 to 3.50
.DIS	2.75 to 3.25
.DDS	3.00
.DMP standard	16.75 to 27.00
.DDI standard	9.25 to 73.10
.DDIR standard	9.5 to 73.50

Table 3-5. Typical Base Set Instruction Execution Times (Continued)

INSTRUCTION	EXECUTION TIME (μsec)
Virtual Memory Instructions	
.LBP	6.25
.LBPR	7.00
.LPX	7.75
.LPXR	9.00
.IMAP (Basic)	9.75
Per parameter (standard)	9.75 to 14.75
.JMAP	9.75
Per parameter (standard)	11.00 to 26.00
.IRES (Basic)	5.0
Per parameter (standard)	9.75 to 14.75
.JRES (Basic)	5.5
Per parameter (standard)	11.00 to 26.00
.PMAP	5.00 to 6.00
Operating System Instructions	
.CPUID	1.25
.FWID	2.50
.SIP	1.50
with skip	1.36
.WFI	Until interrupted
Dynamic Mapping System Instruction Group	Refer to Chapter 4 for detailed descriptions and execution times.

Table 3-6. Double Precision Floating Point Execution Times

INSTRUCTION	EXECUTION TIME (μsec)
FIXD	7.0 to 8.2
FLTD	8.0 to 8.6
.TADD, .TSUB	23 to 26
.TDIV	68 to 72
.TFXS	8.0 to 9.5
.TFTS	9.5 to 10.4
.TFXD	9.5 to 10.7
.TFTD	10 to 11.1
.TMPY	57 to 59

Table 3-7. Instructions and Opcodes for RTE-A Implementation

INSTRUCTION MNEMONIC	OCTAL OPCODE	INSTRUCTION MNEMONIC	OCTAL OPCODE	INSTRUCTION MNEMONIC	OCTAL OPCODE	INSTRUCTION MNEMONIC	OCTAL OPCODE
LPMR	105700	JLY	105762	JLA	100600	.TFTD	105126
SPMR	105701	JPY	105772	JLB	104600	.TFXS	105102
LDMP	105702	LBT	105763	XLA	101724	.TFXD	105106
STMP	105703	SBT	105764	XLD	101724	.NGL	105214
LWD1	105704	MBT	105765	XLB	105724	.BLE	105207
LWD2	105705	CBT	105766	XSA	101725	.FLUN	105226
SWMP	105706	SFB	105767	XST	101725	.PACK	105230
SIMP	105707	SBS	105773	XSB	105725	.FIXD	105104
XJMP	105710	CBS	105774	XCA	101726	.FLTD	105124
XJCQ	105711	TBS	105775	XCB	105726	.PWR2	105225
XLA1	101724	MVW	105777	MWF	105732	.DMP	105054
XLA2	101421	CMW	105776	MWI	105730	.DDI	105074
XLB1	105724	.XFER	105220	MWW	105733	.DDIR	105134
XLB2	105721	.ENTR	105223	MBF	101732	.PCALI	105400
XSA1	101725	.ENTP	105224	MBI	101730	.PCALX	105401
XSA2	101722	.ENTN	105234	MBW	101733	.PCALV	105402
XSB1	105725	.ENTC	105235	.IMAP	105250	.PCALN	105404
XSB2	105722	.DFER	105205	.IRES	105244	.PCALR	105406
XCA1	101726	.CFER	105231	.JRES	105252	.EXIT	105417
XCA2	101723	.FCM	105232	.JMAP	105245	.EXIT1	105415
XCB1	105726	.ZFER	105237	.LPXR	105254	.EXIT2	105416
XCB2	105723	.SETP	105227	.LPX	105255	.SOSP	105406
MW00	105727	$SETP	105227	.LBPR	105256	.CCQA	101406
MW01	105730	.CPM	105236	.LBP	105257	.CCQB	105406
MW02	105731	.DNG	105203	.FAD	105000	.CACQ	101407
MW10	105732	.DCO	105204	.FAD	105000	.CBCQ	105407
MW11	105733	.DIN	105210	.FSB	105020	.CZA	101410
MW12	105734	.DDE	105211	FSB	105020	.CZB	105410
MW20	105735	.DIS	105212	.FMP	105040	.CAZ	101411
MW21	105736	.DDS	105213	FMP	105040	.CBZ	105411
MW22	105737	.DAD	105214	.FDV	105060	.CIQA	101412
MB00	101727	.DSB	105014	FDV	105060	.CIQB	105412
MB01	101730	.DSBR	105034	.FIX	105100	.ADQA	101413
MB02	101731	.PMAP	105240	FIX	105100	.ADQB	105413
MB10	101732	CXA	101744	IFIX	105100		
MB11	101733	CYA	101754	.FLT	105120		
MB12	101734	CXB	105744	FLT	105120		
MB20	101735	CYB	105754	FLOAT	105120		
MB21	101736	XAX	101747	.CPU	105300		
MB22	101737	XAY	101757	.FWID	105301		
SAX	101740	XBX	105747	.WFI	105302		
SAY	101750	XBY	105757	.SIP	105303		
SBX	105740	STX	105743	.DLD	104200		
SBY	105750	STY	105753	.DST	104400		
CAX	101741	LDX	105745	.MPY	100200		
CAY	101751	LDY	105755	.DIV	100400		
CBX	105741	ISX	105760	.TADD	105002		
CBY	105751	ISY	105770	.TSUB	105022		
LAX	101742	DSX	105761	.TMPY	105042		
LAY	101752	DSY	105771	.TDIV	105062		
LBX	105742	ADX	105746	..TCM	105233		
LBY	105752	ADY	105756	.TFTS	105122		

4

Dynamic Mapping System

The basic addressing space of the HP 1000 A400 computer is 32768 words, which is referred to as logical memory. The amount of memory actually installed in the computer system is referred to as physical memory. The Dynamic Mapping System (DMS) is standard logic in the HP 1000 A400 computer and provides an addressing capability for up to 16 million words of physical memory. The DMS allows logical memory to be mapped into physical memory through the use of dynamically alterable memory maps.

Memory Addressing

The basic memory addressing scheme provides for addressing 32 pages of logical memory, each page consisting of 1024 words. This memory is addressed through a 15-bit logical address bus as shown in Figure 4-1. The upper 5 bits of this bus provide the logical page address and the lower 10 bits provide the relative word offset within the page.
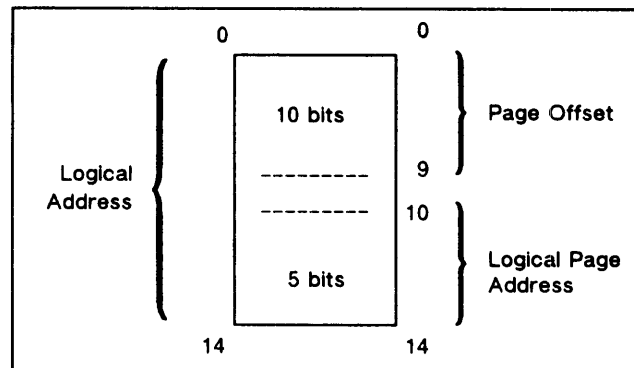
Figure 4-1. Basic Logical Memory Addressing Scheme

Also associated with any memory access is a 5-bit logical map number. The DMS converts the logical map number and the logical page address into a 14-bit physical page number, thereby allowing 16k (2^{14}) pages of physical memory to be addressed. This conversion is accomplished by having the 5-bit logical map number and the 5-bit logical page address access 1024 page mapping registers (PMRs), each of which is 16 bits wide. Each of these map registers contains the user-specified (privileged) 14-bit page address. This new page address is combined with the original 10-bit page offset to form a 24-bit memory address as shown in the Figure 4-2. The PMRs also contain two bits of memory protection information. Bit 15 indicates that the page is read-protected when privileged mode is disabled. Bit 14 indicates that the page is write-protected when privileged mode is disabled. Any attempt to read from a read-protected page will result in a read violation and the memory read will return an undefined result. Any attempt to write into a write-protected page will result in a write violation and the memory will not be altered. If a read or write violation occurs, the DMS signals the memory protect logic (part of the memory controller) that a violation has occurred, which causes the memory protect logic to generate an interrupt. As discussed in the "Interrupt System" chapter, memory protect violations are interrupted to select code 07.

The width of the PMRs is limited to a 16-bit word, of which two bits specify read/write protection, so the maximum width of the physical page address is 14 bits.
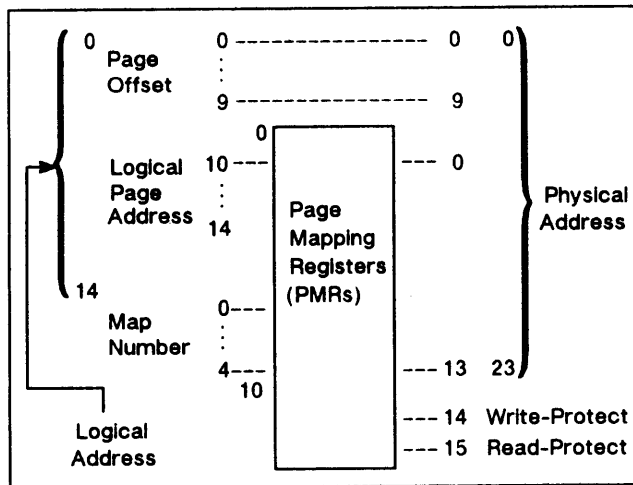
Figure 4-2. Expanded Memory Addressing Scheme

General Descriptions

Page Mapping Register Instructions

The page mapping register instructions allow the privileged user to alter the PMRs, each of which have the following format:

PAGE MAPPING REGISTER FORMAT

```
 0  ┐
 :  ├  physical page number
 :  │
13  ┘
14    -- write protect this page
15    -- read protect this page
```

The page mapping register instructions are:

LPMR – load a PMR indexed by A-Register from B-Register
SPMR – store a PMR indexed by A-Register to B-Register
LDMP – load a map from memory
STMP – store a map to memory

All of these instructions are privileged.

Working Map Instructions

The computer maintains three logical maps, cumulatively called the Working Map Set (WMAP). The working map instructions allow the system to alter the logical maps, and also to start a user program.

The Execute map is the map number used for instruction fetches and normal memory accesses. The data maps (DATA1 and DATA2) are the map numbers used in cross-map memory references. There are two data maps to allow the system to do cross-map moves from one area of memory to another without having to go through the system map. In addition, this feature allows the system to be able to quickly access one area of memory (such as a System Available Memory map) while being able to also access another (such as the user's map). Memory references to locations 0 or 1 in the Execute map are defined to access the A- or B-Registers, respectively. References to 0 or 1 in the data maps are defined to access physical memory locations.

The computer has an additional working map called the Code map. The Code map is defined as the Execute map that has been inclusively OR'd with 1, following which the original Execute map is redefined as the data map. The use of separate maps for both code and data occurs only when CDS mode is enabled, and effectively doubles the logical address space for user programs. The format of WMAP is as follows:

WMAP FORMAT

```
 0 ⎫
 ⋮ ⎬ Execute map number
 4 ⎭
 5 ⎫
 ⋮ ⎬ DATA1 map number
 9 ⎭
10 ⎫
 ⋮ ⎬ DATA2 map number
14 ⎭
15    -- memory protection enable
```

Upon servicing interrupts, the computer saves the currently executing WMAP in a register called IMAP, and loads WMAP with the following values:

1. The DATA1 map is set to the old Execute map.
2. The new Execute map is set to zero.
3. The DATA2 map contains an undefined value.
4. Memory protection is disabled.

The working map instructions are:

```
XJMP  -  cross jump
XJCQ  -  cross map jump (and load C and Q)
SWMP  -  store current WMAP into memory
SIMP  -  store current IMAP into memory
LWD1  -  load WMAP field DATA1 from memory
LWD2  -  load WMAP field DATA2 from memory
```

All of these instructions are privileged.

Cross-Map Instructions

While the working map instructions provide a way to load the working map set, the cross-map instructions provide a means to use them.

These instructions are non-privileged. For all of these instructions, indirect DEF references are done through the Execute map, while the final reference is done through the specified map. When Code and Data Separation (CDS) is enabled, any memory accesses involving the Execute map number are considered to be data accesses, and the base register hardware will add the base (Q) register value to memory addresses from 2 through 1023. Memory accesses involving the DATA1 or DATA2 map numbers are done with CDS disabled, so accesses to the base page will not have the base register added.

Abbreviations used are:

"0" – means logical Execute map
"1" – means logical DATA1 map
"2" – means logical DATA2 map

The cross map instructions are:

XLA1 – cross load A through the DATA1 map
XLB1 – cross load B through the DATA1 map
XLA2 – cross load A through the DATA2 map
XLB2 – cross load B through the DATA2 map
XSA1 – cross store A through the DATA1 map
XSB1 – cross store B through the DATA1 map
XSA2 – cross store A through the DATA2 map
XSB2 – cross store B through the DATA2 map
XCA1 – cross compare A through the DATA1map
XCB1 – cross compare B through the DATA1 map
XCA2 – cross compare A through the DATA2 map
XCB2 – cross compare B through the DATA2 map
MW00 – cross move words from Execute to Execute
MW01 – cross move words from Execute to DATA1
MW02 – cross move words from Execute to DATA2
MW10 – cross move words from DATA1 to Execute
MW11 – cross move words from DATA1 to DATA1
MW12 – cross move words from DATA1 to DATA2
MW20 – cross move words from DATA2 to Execute
MW21 – cross move words from DATA2 to DATA1
MW22 – cross move words from DATA2 to DATA2
MB00 – cross move bytes from Execute to Execute
MB01 – cross move bytes from Execute to DATA1
MB02 – cross move bytes from Execute to DATA2
MB10 – cross move bytes from DATA1 to Execute
MB11 – cross move bytes from DATA1 to DATA1
MB12 – cross move bytes from DATA1 to DATA2
MB20 – cross move bytes from DATA2 to Execute
MB21 – cross move bytes from DATA2 to DATA1
MB22 – cross move bytes from DATA2 to DATA2

If CDS mode is enabled, the base (Q) register will be added to base relative
addresses in the Execute map only. Cross map references to addresses in one of the
alternate maps are not checked for base relativity.

Detailed Descriptions

The following paragraphs provide machine language coding and definitions for the
DMS instructions. Note that all memory accesses are subject to the DMS memory
protection rules.

LPMR **LOAD PAGE MAPPING REGISTER**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 0 0	0 0 0

Loads the contents of the B-Register into the page mapping register (PMR) addressed by the contents of the A-Register. Any attempt to address a PMR outside the range of 0 to 1023 or to modify a PMR that is currently being accessed produces undefined results. The format for the PMR contents is: bit 15 = read protect; bit 14 = write protect; and bits 13 to 0 = physical page number. This instruction is privileged. After the operation, the A-Register is incremented.

SPMR **STORE PAGE MAPPING REGISTER**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 0 0	0 0 1

Loads the contents of the page mapping register (PMR) addressed by the value in the A-Register into the B-Register. Any attempt to address a PMR outside the range of 0 to 1023 produces undefined results. The format for the PMR contents is: bit 15 = read protect; bit 14 = write protect; and bits 13 to 0 = physical page number. This instruction is privileged. After the operation, the A-Register is incremented.

LDMP **LOAD A MAP**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 0 0	0 1 0
D/I					
D/I					

Loads the map number specified by Word 2 from the 32-word block of memory specified by Word 3, where:

Word 1 = Instruction code.
Word 2 = Pointer to Map number.
Word 3 = Pointer to Map image.

There are 32 maps of 32 PMRs each; the beginning PMR number of a map is related to the map number as follows:

PMR number = Map number x 32

Undefined results occur when a map number outside the range of 0 to 31 is addressed, when modification of a currently executing map is tried, or when the resolved address of the map image is outside the range of 2 to 77740 octal.

All memory references are done in the Execute map and may include the A- and B-Registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

STMP **STORE A MAP**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	1
D/I															
D/I															

Stores the map number specified by Word 2 to the 32-word block of memory specified by Word 3, where:

Word 1 = Instruction code.
Word 2 = Pointer to Map number.
Word 3 = Pointer to Map image.

There are 32 maps of 32 PMRs each; the beginning PMR number of a map is related to the map number as follows:

PMR number = Map number x 32

Undefined results occur when a map number outside the range of 0 to 31 is addressed or when the resolved address of the map image is outside the range of 2 to 77740 octal.

All memory references are done in the Execute map and may include the A- and B-Registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XJMP **CROSS MAP JUMP**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	0	1	0	0	0
D/I															
D/I															

Resolves indirect references, sets the program counter to the resolved address specified by Word 3, and loads WMAP with the value pointed to by the resolved address of Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to new WMAP number.
Word 3 = Pointer to next instruction (new PC value).

All memory references (direct and indirect) are done in the Execute map and may include the A- and B-Registers. The next instruction will be fetched using the new WMAP. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XJCQ

CROSS MAP JUMP (AND LOAD C AND Q)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	0	1	0	0	1
D/I															
D/I															
D/I															

Resolves indirect references, sets the program counter to the resolved address specified by Word 3, and loads the WMAP specified by Word 2, and loads the C- and Q-Registers with new values addressed by Word 4, where:

Word 1 = instruction opcode.
Word 2 = pointer to new WMAP number.
Word 3 = pointer to next instruction (new PC value).
Word 4 = pointer to new C- and Q-Register values.

All memory references (direct and indirect) are done in the Execute map and may include the A- and B-Registers. The next instruction will be fetched using the new WMAP, under a CDS mode specified by the new C-Register value. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

SWMP   **SAVE WORKING MAP**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	0	0	1	1	0
D/I															

Stores WMAP at the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to destination in memory.

All memory references are done in the Execute map and may include the A- and B-Registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

SIMP   **SAVE INTERRUPTED MAP**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	0	0	1	1	1
D/I															

Stores IMAP at the location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to destination in memory.

All memory references are done in the Execute map and may include the A- and B-Registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

LWD1   **LOAD DATA1 MAP**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	0	0	1	0	0
D/I															

Loads the DATA1 field of the WMAP register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to new DATA1 map.

All memory references are done in the Execute map and may include the A- and B-Registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted. Map numbers outside the range of 0-31 produce undefined results.

LWD2   **LOAD DATA2 MAP**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 0 0	1 0 1
D/I					

Loads the DATA2 field of the WMAP register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to new DATA2 map.

All memory references are done in the Execute map and may include the A- and B-Registers. This instruction is privileged and is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted. Map numbers outside the range of 0-31 produce undefined results.

XLA1   **CROSS LOAD A THROUGH DATA1 MAP**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 0	1 0 0
D/I					

Loads the A-Register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

CROSS LOAD B
XLB1   THROUGH DATA1 MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	1	0	1	0	0
D/I															

Loads the B-Register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

CROSS LOAD A
XLA2   THROUGH DATA2 MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	0	1	0	0	0	1
D/I															

Loads the A-Register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

CROSS LOAD B
XLB2   THROUGH DATA2 MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	1	0	0	0	1
D/I															

Loads the B-Register from the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

CROSS STORE A
XSA1 THROUGH DATA1 MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	0	1	0	1	0	1
D/I															

Stores the A-Register contents in the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

CROSS STORE B
XSB1 THROUGH DATA1 MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	1	0	1	0	1
D/I															

Stores the B-Register contents in the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XSA2

CROSS STORE A THROUGH DATA2 MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	0	1	0	0	1	0
D/I															

Stores the A-Register contents in the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

XSB2

CROSS STORE B THROUGH DATA2 MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	1	0	0	1	0
D/I															

Stores the B-Register contents in the memory location pointed to by Word 2, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

CROSS COMPARE A THROUGH DATA1 MAP

XCA1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	0	1	0	1	1	0
D/I															

Compares the A-Register contents with a value in the memory location pointed to by Word 2 and skips if the values are not equal, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

CROSS COMPARE B THROUGH DATA1 MAP

XCB1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	1	0	1	1	0
D/I															

Compares the B-Register contents with a value in the memory location pointed to by Word 2 and skips if the values are not equal, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA1 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA1 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA1 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	0	1	0	0	1	1
D/I															

Compares the A-Register contents with a value in the memory location pointed to by Word 2 and skips if the values are not equal, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

CROSS COMPARE B
XCB2   THROUGH DATA2 MAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	1	0	1	0	0	1	1
D/I															

Compares the B-Register contents with a value in the memory location pointed to by Word 2 and skips if the values are not equal, where:

Word 1 = Instruction code.
Word 2 = Pointer to memory location in DATA2 map.

All indirect memory references are done in the Execute map and may include the A- and B-Registers and will be checked for base relativity if CDS mode is enabled. The direct memory reference is done in the DATA2 map. Because A- and B-Register addressing and base relative checking are disabled in the DATA2 map, direct addresses 0 through 1777 refer to physical memory locations. This instruction is interruptible in that it may be interrupted during indirect address resolution after three levels of indirection, and then restarted.

MW00

**CROSS MOVE WORDS,
EXECUTE TO EXECUTE**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 0	1 1 1

Moves a block of words from the Execute map to the Execute map. The A-Register specifies the source address, the B-Register specifies the destination address, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the A- and B-Registers will be checked for base relativity before execution. Upon exit these registers will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

MW01

**CROSS MOVE WORDS,
EXECUTE TO DATA1**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 1	0 0 0

Moves a block of words from the Execute map to the DATA1 map. The A-Register specifies the source address in the Execute map, the B-Register specifies the destination address in the DATA1 map, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the A-Register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE WORDS,
EXECUTE TO DATA2

MW02

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 1	0 0 1

Moves a block of words from the Execute map to the DATA2 map. The A-Register specifies the source address in the Execute map, the B-Register specifies the destination address in the DATA2 map, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the A-Register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE WORDS,
DATA1 TO EXECUTE

MW10

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 1	0 1 0

Moves a block of words from the DATA1 map to the Execute map. The A-Register specifies the source address in the DATA1 map, the B-Register specifies the destination address in the Execute map, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the B-Register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B-, or X-Registers.

CROSS MOVE WORDS,
MW11   **DATA1 TO DATA1**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 1	0 1 1

Moves a block of words from one location in the DATA1 map to another in the DATA1 map. The A-Register specifies the source address, the B-Register specifies the destination address, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

This instruction produces undefined results if the A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE WORDS,
MW12   **DATA1 TO DATA2**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 1	1 0 0

Moves a block of words from the DATA1 map to the DATA2 map. The A-Register specifies the source address in the DATA1 map, the B-Register specifies the destination address in the DATA2 map, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

This instruction produces undefined results if the A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

**CROSS MOVE WORDS,
DATA2 TO EXECUTE**

MW20

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 1	1 0 1

Moves a block of words from the DATA2 map to the Execute map. The A-Register specifies the source address in the DATA2 map, the B-Register specifies the destination address in the Execute map, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the B-Register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

**CROSS MOVE WORDS,
DATA2 TO DATA1**

MW21

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 1	1 1 0

Moves a block of words from the DATA2 map to the DATA1 map. The A-Register specifies the source address in the DATA2 map, the B-Register specifies the destination address in the DATA1 map, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

This instruction produces undefined results if A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

**CROSS MOVE WORDS,
DATA2 TO DATA2**

MW22

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 1 1	0 1 1	1 1 1

Moves a block of words from the DATA2 map to the DATA2 map. The A-Register specifies the source address, the B-Register specifies the destination address, and the X-Register specifies the number of words to be moved (which must be an integer equal to or greater than zero). Address bit 15 must be zero, as indirect source and destination references are not allowed. On return, the A-Register contains the last memory address in the source block moved plus one, the B-Register contains the last memory address in the destination block moved plus one, and the X-Register is zero.

This instruction produces undefined results if the A-, B-, or X-Register has bit 15 set or if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

**CROSS MOVE BYTES,
EXECUTE TO EXECUTE**

MB00

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 0	1 1 1

Moves a block of bytes from one location in the Execute map to another in the Execute map. The A-Register specifies the source address and the B-Register specifies the destination address. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the A- and B-Registers will be checked for base relativity before execution. Upon exit these registers will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

**CROSS MOVE BYTES,
EXECUTE TO DATA1**

MB01

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 1	0 0 0

Moves a block of bytes from a location in the Execute map to one in the DATA1 map. The A-Register specifies the source address in the Execute map, and the B-Register specifies the destination address in the DATA1 map. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the A-Register will be checked for base relativity before executing. Upon exit this register will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

**CROSS MOVE BYTES,
EXECUTE TO DATA2**

MB02

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 1	0 0 1

Moves a block of bytes from a location in the Execute map to one in the DATA2 map. The A-Register specifies the source address in the Execute map, and the B-Register specifies the destination address in the DATA2 map. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the A-Register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE BYTES, DATA1 TO EXECUTE

MB10

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 1	0 1 0

Moves a block of bytes from a location in the DATA1 map to one in the Execute map. The A-Register specifies the source address in the DATA1 map, and the B-Register specifies the destination address in the Execute map. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the B-Register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE BYTES, DATA1 TO DATA1

MB11

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 1	0 1 1

Moves a block of bytes from one location in the DATA1 map to another in the DATA1 map. The A-Register specifies the source address and the B-Register specifies the destination address. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE BYTES,
DATA1 TO DATA2

MB12

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 1	1 0 0

Moves a block of bytes from a location in the DATA1 map to one in the DATA2 map. The A-Register specifies the source address in the DATA1 map, and the B-Register specifies the destination address in the DATA2 map. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE BYTES,
DATA2 TO EXECUTE

MB20

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 1	1 0 1

Moves a block of bytes from a location in the DATA2 map to one in the Execute map. The A-Register specifies the source address in the DATA2 map, and the B-Register specifies the destination address in the Execute map. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

If CDS mode is enabled, the B-Register will be checked for base relativity before execution. Upon exit this register will contain the base relative address, incremented by the count in the X-Register.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE BYTES,
DATA2 TO DATA1

MB21

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 1	1 1 0

Moves a block of bytes from a location in the DATA2 map to one in the DATA1 map. The A-Register specifies the source address in the DATA2 map, and the B-Register specifies the destination address in the DATA1 map. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

CROSS MOVE BYTES,
DATA2 TO DATA2

MB22

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 1 1	0 1 1	1 1 1

Moves a block of bytes from one location in the DATA2 map to another in the DATA2 map. The A-Register specifies the source address and the B-Register specifies the destination address. The X-Register specifies the number of bytes to be moved (which is an unsigned 16-bit number that may equal zero). Indirect addressing is not allowed because a byte address uses all 16 bits. A byte address is two times the word address plus zero or one, which specifies the high order (bits 15 to 8) or low order (bits 7 to 0) position, respectively. On return, the A-Register contains the last memory byte address in the source block moved plus one, the B-Register contains the last byte address in the destination block moved plus one, and the X-Register is zero.

This instruction produces undefined results if the source or destination address rolls over. It is interruptible, with the context saved in the A-, B- and X-Registers.

DMS Instruction Execution Times

Table 4-1 lists the execution times for the various DMS instructions.

Assembly Language and RTE Implementation

Refer to the Assembly Language and RTE Implementation sections of the "Dynamic Mapping System" chapter for information on implementing the DMS instructions in HP Assembly Language and in an HP RTE-A operating system.

Table 4-1. Dynamic Mapping Instructions Execution Times

Instruction	Execution Time (μsec)
XLA1/XLB1, XLA2/XLB2	2.50
XSA1/XSB1, XSA2/XSB2	2.50
XCA1/XCB2, XCA2/XCB2	2.50
MW00, MW01, MW02, MW10, MW11, MW12, MW20, MW21, MW22	3.19 plus 0.91 per word moved
MB00, MB01, MB02, MB10, MB11, MB12, MB20, MB21, MB22	3.19 plus 0.91- 2.05 per byte moved
LPMR	1.37
SPMR	1.14
LDMP, STMP	18.20
XJMP	5.23
XJCQ	6.60
LWD1, LWD2	2.28
SWMP	4.55
SIMP	1.82

Note: Memory refresh during a processor memory access can make an instruction approximately 3 percent slower. Heavy DMA activity can also degrade (lengthen) execution times due to contention for memory.

5

Code and Data Separation

The basic logical address space of the HP 1000 A-Series architecture is 32768 words, in which both code and data reside. Code and Data Separation (CDS) is an enhancement to the A-Series architecture which separates code and data into separate logical address spaces. The main benefit of CDS is that it provides support of programs that may have up to 4M words of code, and this code may be either memory-resident or disc-resident. The optional HP 92078A package for RTE-A provides software support for CDS. Refer to the RTE-A Programmer's Reference Manual for a description of how to take advantage of CDS by using Macro/1000 and other HP languages.

Code and Data Addressing

CDS utilizes the Dynamic Mapping System environment of the A-Series architecture, and uses separate DMS maps to reference code and data. The term "code" refers to opcodes, DEFs to parameters, in-line constants, current-page links and constants for Memory Reference Group (MRG) instructions. The term "data" refers to variables and constants used by a program.

When CDS is disabled, both code and data are accessed through the logical address space of the computer, which is 32k words. The DMS maps this logical address space into the physical address space of up to 16M words. This is accomplished through the use of 32 memory maps of 32 pages each. A program executes in a single map, which is called the Execute map, although it may access memory through other maps using DMS instructions.

When CDS is enabled, code and data are accessed through separate maps. The Execute map number specifies which map is used to access data, and the Execute map number inclusive-ORed with '1' is used to access code. The Execute map number must be an even number between 0 and 30, inclusive. In all subsequent descriptions, DATA[n] and CODE[n] refer to memory locations in data space and code space, respectively. In addition, when CDS is enabled the base register (Q) is enabled, and all Execute map memory addresses that lie in the range 2 through 1023 have the Q-Register added by the memory accessing hardware before the memory location is accessed. Locations 0 and 1 of data space are still defined to reference the A- and B-Registers. Cross-map memory accesses, such as XLA1, are done with CDS disabled.

As an example, consider a DLD 500 instruction that is executed with CDS on, with an Execute map number of 2, and with the Q-Register equal to 5000. The DLD opcode and the DEF 500 are read from memory using map number 3, because these words are considered to be code. The memory values loaded into A and B will be read through map number 2, because these words are considered to be data. The actual address of the memory locations to be loaded is 5500, because the hardware automatically adds the Q-Register to memory addresses between 2 and 1023.

Most instructions separate code and data as was described for the previous example, but the Memory Reference Group has some exceptions. The JSB, STA current page direct, STB current page direct, and ISZ current page direct instructions may not be used when CDS is enabled because they attempt to write into code space. MRG references to base page always access memory in the data space, but MRG references to the current page always access code space for the first memory access and data space for all subsequent direct/indirect levels. That means that an LDA current page direct will load a constant from code space, that an LDA current page indirect will access a current page link in code space and then data in data space, and so on for the other MRG instructions. Note also that base page MRG references are useful for accessing variables that are Q-relative, such as the local variables or parameter pointers in a stack frame (to be described later).

The following restrictions must be met when CDS is enabled; otherwise, undefined results may occur. The Q-Register value must lie in the range of 1024 through 32767. The program counter must lie in the range 1024 to 32767, which means that jump instructions may not jump to the base page or to the A- or B-Register.

Support for linking of relocatable code is provided by the RTE-A LINK program.

General Descriptions

Procedure Call Instructions

The procedure call (PCAL) instructions are used to invoke a procedure, which may reside in code or data space. All of the PCAL instructions adjust the Q-Register to allocate and set up a new stack marker (memory locations used to link procedure invocations and exits), and branch to the new procedure.

The PCAL instructions are:

    PCALI – procedure call to current segment
    PCALX – procedure call to any segment
    PCALV – procedure call to any segment (variable)
    PCALR – procedure call to .ENTR-compatible non-CDS code in data space
    PCALN – procedure call to .ENTN-compatible non-CDS code in data space

The PCALI instruction is the fastest PCAL instruction, and it is used to call a procedure that resides in the current code address space.

Two of the PCAL instructions (PCALX, PCALV) are capable of remapping the logical code space to another area of physical memory. Each logical code space is called a segment, and these PCALs are called cross-segment PCAL instructions.

The last two PCAL instructions (PCALR, PCALN) are used to call code that is not CDS-compatible. Such code resides in the data space, and must follow the .ENTR or .ENTN procedure call sequence.

The standard PCAL call sequence is:

    PCAL opcode(PCALI, PCALX, PCALV, PCALR,
    or PCALN)
    LABEL PE
    DEC AC [,I]
    DEF A_1 [,I]
        .
        .
        .
    DEF A_AC [,I]
    (return point from procedure PE)
        .
        .
        .
PE DEC FS
    (next instruction to be executed in procedure PE)
        .
        .
        .
    EXIT opcode (EXIT, EXIT1, or EXIT2)

The PCAL opcode is the appropriate opcode to be used to access the new procedure. If the new procedure is in the same segment, then PCALI should be used. If the new procedure is in another segment, then PCALX or PCALV should be used. If the new procedure is not CDS-compatible, then PCALR or PCALN should be used. Note that the selection of the PCAL opcode is done automatically by the RTE-A LINK program, which will also automatically segment your program for you.

The LABEL to the new procedure points to the location of the new procedure. In the case of PCALI, PCALR, and PCALN, the LABEL is a DEF (a 15-bit logical address, possibly indirect) to the new procedure. In the case of PCALX, the LABEL consists of a word which contains information that determines how the logical code space must be remapped to get to the new procedure. In the case of PCALV, the DEF (which may be indirect) points to a word in data space which specifies how code space should be remapped.

AC is a word which specifies how many parameter pointers follow. Parameter pointers are 15-bit logical addresses (with the 16th bit specifying indirection) which point to variables that are being passed as parameters to the new procedure. From 0 to 255 parameter pointers may be passed in the PCAL call sequence.

Procedure Exit Instructions

The Exit instructions are:

    EXIT   – procedure exit with no skips
    EXIT1   – procedure exit with one skip
    EXIT2   – procedure exit with two skips

C, Q, Z, and IQ Instructions

Other instructions are provided to access the C-, Q-, Z- and IQ-Registers. These are:

    CCQA   (CCQB) –   copy C and Q to A (or B)
    CACQ   (CBCQ) –   copy A (or B) to C and Q
    CZA   (CZB)   –   copy Z to A (or B)
    CAZ   (CBZ)   –   copy A (or B) to Z
    CIQA   (CIQB) –   copy IQ to A (or B)
    ADQA   (ADQB) –   add Q to A (or B)
    SDSP   –   store display

Stack Frame Description

A stack frame is an area of memory in the logical data space that contains variables local to a procedure and pointers to variables of other procedures. The stack frame also contains six words of information called the stack marker, which links the procedure call chain from one procedure invocation to the next. The general layout of a stack frame is shown in Figure 5-1.

The Z-Register, also called the bounds register, increases the reliability of CDS software. The bounds register detects the growth of a stack frame past the end of the allowed data space into areas used by VMA or memory used for other purposes. On every PCAL instruction, the microcode checks that the NEXT_Q value of a created stack marker is less than the Z-Register. If this check fails then the program will interrupt to the memory protect handler (see PCALI description for more detail).

PREV_Q is the Q-register value for the calling procedure.

RETURN_P is the return address in the calling procedure.

R is the return segment indicator: R = 0 indicates the return address is in the same segment as the calling procedure (a segment reload is not required), R = 1 indicates the return address is in segment RETURN_CST (a segment reload is required.

ARG_COUNT is a number (0-255) that is the count of actual parameters passed to the called procedure. This field is maintained for all PCAL instructions.

NEXT_Q is the Q-Register value to use when building the next stack frame during a subsequent PCAL NEXT_Q may be adjusted during the execution of a procedure to alter the size of the local variable space.

STATIC_Q is a word that is used by block-structured languages such as Pascal. This word and the RESERVED word are reserved for use by Hewlett-Packard software.

Figure 5-1. Stack Frame General Layout

Detailed Descriptions

PCALI   **INTERNAL PROCEDURE CALL**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 0	0 0 0

D/I	DEF to subroutine

N = argument count (0 ≤ N ≤ 255)

D/I D/I	N DEFs to arguments
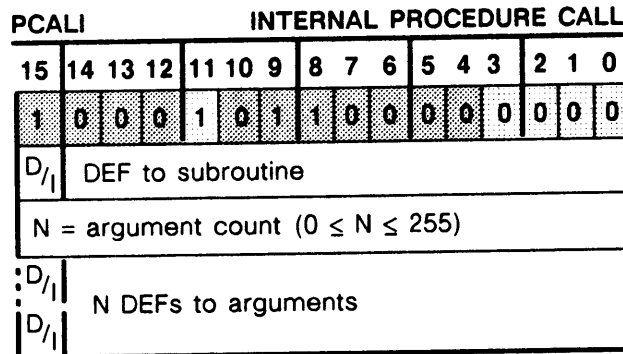
Function:	Procedure call to current code segment

Use:	*Current Code Segment*

PCALI
DEF pe [,I]
DEC ac
DEF a_1 [,I]
.
.
.

DEF a_ac [,I]

External Code Segment

pe   EQU*
      DEC fs
.
.
.

Operands:	pe: Procedure entry point ac: Actual argument count a_i :Actual argument i (multiple indirects are supported) fs : Frame size in words
Interruptible:	Yes

PCALI determines the new Q-Register value for the called stack frame, which may be found at the current NEXT_Q value. The old Q value is written into the new stack frame at PREV_Q, which provides a link from the new stack frame to the old stack frame. The argument count (AC) of parameters to be passed is read from CODE[P+2], and the parameter pointers are copied from CODE[P+3] to DATA[new Q+6] after the parameter pointers have been resolved for indirection and base relativity. The value of AC is written into the ARG_COUNT location of the stack marker. Indirects are followed in memory until a direct address is found. If the (direct) address is between 2 and 1023, the current Q-Register value is added before the parameter pointer is copied into data space. PCALI may be interrupted during parameter pointer resolution and copying, and the PCALI instruction may simply be restarted after the interrupt has been processed because the actual state of the calling procedure (specifically the P- and Q-Registers) has not been altered.

The actual parameter count (AC) is stored in the ARG_COUNT field of the new stack frame, and the upper byte of that word (RTN_CST) is undefined. The return point of the procedure (P+3+AC) is stored in the RETURN_P location of the new stack frame. The 'R' bit contains zero, which designates that a subsequent EXOT instruction should exit without loading a new segment.

The called procedure entry (PE) is found by resolving the address at CODE[P+1], and CODE[PE] contains the called frame size (FS). The NEXT_Q value of the new stack frame is set to the new Q value plus FS.

If the NEXT_Q is greater than or equal to the bounds register (Z), stack overflow has occurred and a memory protect interrupt will be executed to memory location 07 of map zero. After the interrupt, the instruction violation register is equal to the fetch address of the PCAL instruction, and the program counter value at the time of the interrupt is undefined. The Q-Register and IQ-Register point to the offending stack marker. The new stack marker and formal arguments may have been written into memory locations at addresses greater than the Z-Register value. To provide a safety zone, set the Z-Register to 264 words below the area you want to protect.

If stack overflow did not occur, PCALI branches to the called procedure by setting the program counter, P, to PE+1 and the Q-Register to the new Q value.

PCALX   EXTERNAL PROCEDURE CALL

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 0	0 0 1

D/I	Code label to subroutine

N = argument count (0 ≤ N ≤ 255)

:D/I :D/I	N DEFs to arguments :
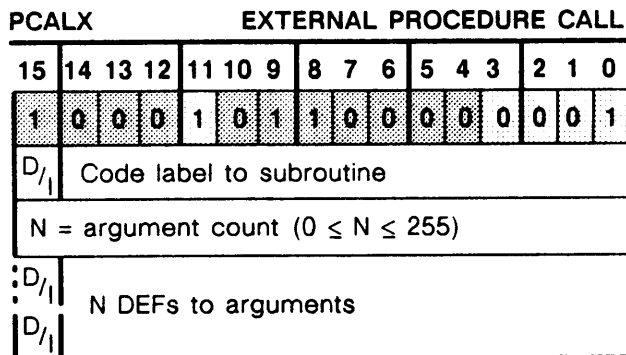
Function: Procedure call to procedure in external segment

Use:

Current Code Segment
PCALX
LABEL pe
DEC ac
DEF a_1 [,I]

     .
     .
     .

DEF a_ac [,I]

External Code Segment
pe   EQU*
     DEC fs
     .
     .
     .

Operands:

pe :   Code label (Code Segment Table index and Segment Transfer
        Table index) to procedure
ac   :   Actual argument count
a_i :   Actual argument i (multiple indirects are supported)
fs   :   Frame size in words

Interruptible: Yes

PCALX determines the new Q-Register value for the called stack frame, which may be found at the current NEXT_Q value. The old Q value is written into the new stack frame at PREV_Q, which provides a link from the new stack frame to the old stack frame. The argument count (AC) of parameters to be passed is read from CODE[P+2], and the parameter pointers are copied from CODE[P+3] to DATA[new Q+6] after the parameter pointers have been resolved for indirection and base relativity. PCALX may be interrupted during the parameter pointer resolution and copying, and the PCALX instruction may simply be restarted after the interrupt has been processed because the actual state of the calling procedure (specifically the P- and Q-Registers) has not been altered.

The return point of the procedure (P+3+AC) is stored in the RETURN_P location of the new stack frame. The 'R' bit contains one, which designates that a subsequent EXIT instruction should load the new segment indicated by RETURN_CST in the stack marker. The current segment number is read from CODE[2000B], ANDed with 177400B, inclusive ORed with AC, and stored in DATA[new Q+3].

PCALX now attempts to load the external segment. The upper byte of CODE[P+1] contains the CST (Code Segment Table) index. The PCALX instruction looks up the CST entry through the base page of the code map set. (The code map set number is the Execute map number inclusive ORed with one.) The memory address of the CST entry is the CST index shifted left two times. Restriction: the CST index must be in the range 0 through 127. Note that this process of looking up a CST entry is done with the base register hardware and A/B addressability off. If bit 15 of the CST entry is '1', then the called procedure is not in memory. PCALX will interrupt to memory location 13 octal of map zero and this location must contain a JSB to the segment interrupt handler. The program counter at the time of the interrupt points to the offending PCALX instruction, and the Q value is unchanged. After the segment is loaded, the PCALX instruction may be re-executed. The CDS segment interrupt is the lowest priority interrupt, and if other interrupts are present when a fault is detected, the instruction is simply restarted after the other interrupts are serviced. The following paragraphs describe what PCALX does if the segment is present in memory.

This paragraph describes how a code segment is 'mapped in'. The lower 14 bits of the CST entry contain the starting physical page of the new code segment, which the microcode maps in by setting the PMRs (page mapping registers) of code page 1 to the physical page number, code page 2 to the physical page number plus 1, code page 3 to the physical page number plus 2, and so on. These page mapping registers are write-protected to protect the code against alteration. The base page PMR of the code map is *not* altered.

After the new code segment has been mapped in, the entry point of the called procedure is determined. The low byte of the external label (in CODE[P+1] in the old segment) contains the STT (Segment Transfer Table) index. Beginning at location 2001B in code space is a table of address pointers (with bit 15 set to zero) that point to the externally accessible procedures in this segment. Location 2001B plus the STT index contains the 15-bit address of the called subroutine, and this value is the called procedure entry (PE).

CODE[PE] contains the called frame size (FS). The NEXT_Q value of the new stack frame is set to the new Q value plus FS. If the new NEXT_Q is greater than or equal to the bounds register (Z) then stack overflow has occurred, and a memory protect interrupt will be executed at memory location 07 of map zero. After the interrupt, the instruction violation register is equal to the fetch address of the PCALX instruction, and the program counter contains an undefined value. The Q-Register and IQ-Register point to the offending stack marker. The new stack marker and formal arguments may have written into memory locations at addresses greater than the Z-Register value.

Now that the new stack marker is complete, PCAQLX branches to the called procedure by setting the program counter, P, to PE+1 and the Q-Register to the new Q value.

PCALV — **VARIABLE EXTERNAL PROCEDURE CALL**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 0	0 1 0

D/I	DEF to code label to subroutine

N = argument count (0 ≤ N ≤ 255)

D/I ... D/I	N DEFs to arguments
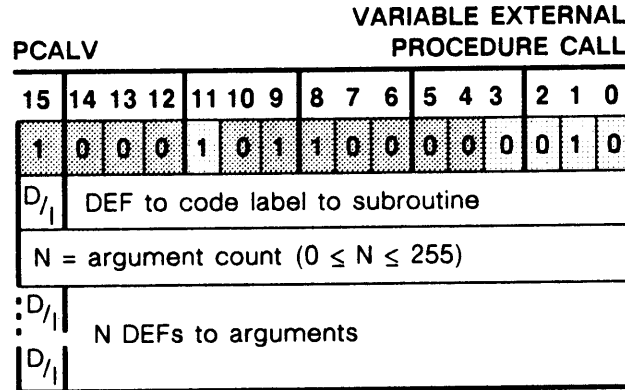
Function: Procedure call, Code to Code, External procedure

Use: *Current Code Segment*

                      PCALV
                      DEF x [,I]
                      DEC ac
                      DEF a_1 [,I]
                      .
                      .
                      .
                      DEF a_ac [,I]

                      External Code Segment

  pe   EQU*
                      DEC fs
                      .
                      .
                      .
                      Data Segment
                      xl LABEL pe
                      .
                      .
                      .

Operands: pe : Procedure entry point
                      xl : Procedure variable
                      ac : Actual argument count
                      a_i: Actual argument i
                      fs : Frame size in words

Interruptible: Yes

The difference between the PCALX and PCALV instructions is that the code label is in the call sequence in PCALX, while in PCALV it is in the data space. The pointer to the external label may be a multi-level indirect. See PCALX for a description of segment loading.

			PROCEDURE CALL,		
PCALR			.ENTR COMPATIBLE		

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 0	0 1 1

D/I	DEF to subroutine

N = argument count (0 ≤ N ≤ 255)

D/I D/I	N DEFs to arguments
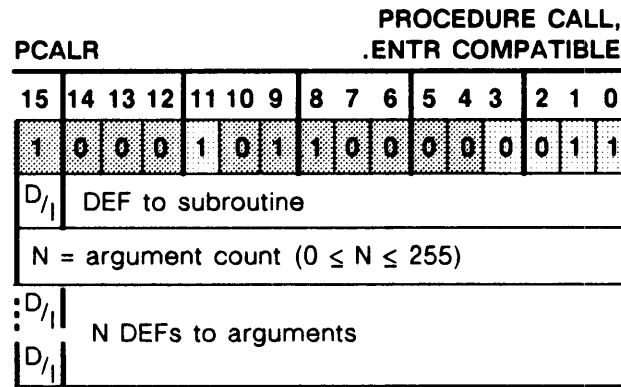
Function:	Procedure call, Code to Data, .ENTR compatible
Use:	*Current Code Segment*

```
PCALR
DEF pe [,I]
DEC ac
DEF a_1 [,I]
     .
     .
     .
DEF a_ac [,I]
```

Data Segment

```
      BSS fc

pe    NOP
      JSB .ENTR
      DEF pe-fc
       .
       .
       .
```

Operands:	pe : Procedure entry point
	ac : Actual argument count
	a_i: Actual argument i
	fc : Formal argument count

Interruptible:	Yes

PCALR is similar to PCALI except it is used for invoking procedures in the data segment that are .ENTR compatible. The mechanism for calling non-CDS code involves copying a .ENTR call sequence (minus the JSB) into the stack frame. PCAL then turns off CDS, and executes the function of a JSB to the non-CDS-code procedure by writing a return address into the new procedure entry and branching to the procedure entry plus one. The procedure entry address must be between 1024 and 32766.

A "DEF*+AC+1" is written into the reserved word location (of the stack marker) for PCALR so as to follow the .ENTR calling convention.

The stack frame created by PCALR (and PCALN) is:

```
Q ──────►   ┌─────────────────────┐
            │    Stack Marker     │
            ├─────────────────────┤
            │    DEF a_1          │
            │        .            │
            │        .            │
            │        .            │
            │    DEF a_ac         │
            │    EXIT             │
            │    EXIT1            │
            │    EXIT2            │
            └─────────────────────┘
```
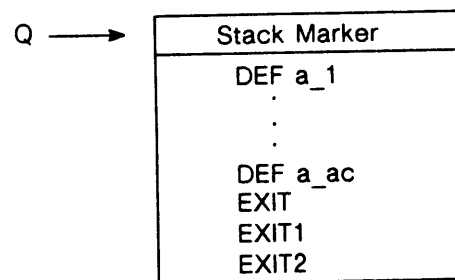
NEXT_Q in the stack marker is undefined.

PROCEDURE CALL, .ENTN COMPATIBLE

PCALN

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 0	1 0 0

D/I	DEF to subroutine

N = argument count (0 ≤ N ≤ 255)

| D/I | N DEFs to arguments |
| D/I | |
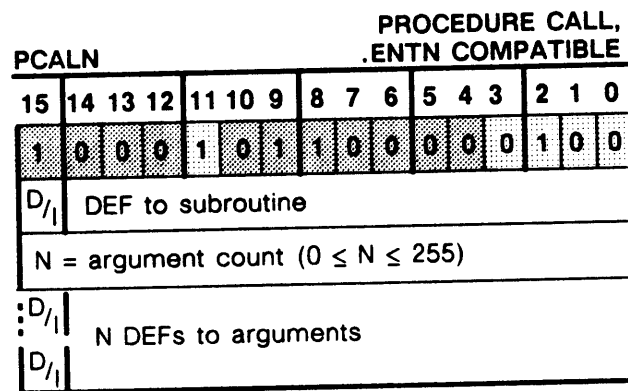
Function: Procedure call, Code to Data, Constant Internal procedure, .ENTR compatible

Use: *Code Segment*

                 PCALN
                 DEF pe [,I]
                 DEC ac
                 DEF a_1 [,I]
                     .
                     .
                     .
                 DEF a_ac [,I]

Data Segment

BSS fc

pe   NOP
        JSB .ENTN
        DEF pe-fc
        .
        .
        .

Operands:	pe : Procedure entry point
	ac : Actual argument count
	a_i: Actual argument i
	fc : Formal argument count

Interruptible:	Yes

The stack frame created by PCALN is similar to the stack frame created by PCALR. The difference between PCALR and PCALN is that the PCALR writes the return address at the non-CDS-code procedure entry, PE, with a return address of the new Q-Register value plus 5, while PCALN writes a return address of the new Q value plus 6. Thus, the return address in PCALR points to a word that points around a parameter list (as in the .ENTR convention), while the return address in PCALN points to the parameter list (as in the .ENTN convention).

SDSP   **STORE DISPLAY**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 0	1 0 1

DEC delta level offset

D/I	DEF location of dl+1 words for display
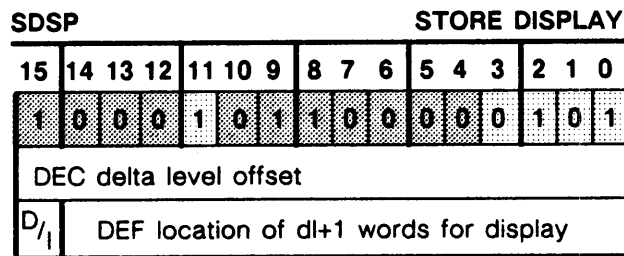
Function:	Store display in memory

Use:	SDSP
	DEC dl
	DEF dsp [,I]

Operands:	dl : delta level offset
	dsp: location of dl+1 words for display

Interruptible:	Yes

The store display instruction is used by block-structured languages such as Pascal to store a number of STATIC_Q words into memory. SDSP begins by storing the current Q-Register value into the DATA[disp]. The following is done dl times: the value just stored into memory is used as an address in memory, and this value, logically ANDed with 7777B, is stored in the word after the last word stored. The following table shows what is placed in the display by the SDSP instruction.

LOCATION	VALUE
disp	Q value for current procedure
disp+1	Q value for first lexically enclosing procedure
disp+2	Q value for second lexically enclosing procedure
.	
.	
.	
disp+dl	Q value for dl-th lexically enclosing procedure

EXIT   **PROCEDURE EXIT**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 1	1 1 1

Function:	Exit from procedure
Use:	EXIT
Interruptible:	No

The EXIT instruction is used by any called procedure (in CDS mode or non-CDS mode) to return to the calling CDS procedure. The RETURN_P word in the stack marker holds the return address, and if bit 15 of that word is 1, then a new segment must be loaded first. The return segment is specified by the RETURN_CST field of the current stack marker. (See 'mapping in' in the PCALX description.) If the returning segment is not in memory, then an interrupt to memory location 13 octal of map zero will occur, with the P- and Q-Registers unaltered by EXIT. The CDS segment interrupt is the lowest priority interrupt, and if other interrupts are present when a fault is detected then the instruction is simply restarted.

If EXIT was able to load the segment, or if the EXIT was to the current segment, then the C- and Q-Registers are loaded from the PREV_Q word, and the P-Register is set to RETURN_P.

EXIT1   PROCEDURE EXIT WITH ONE SKIP

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 1	1 0 1

Function:	Exit from procedure at normal exit + 1
Use:	EXIT1
Interruptible:	No

EXIT1 is functionally identical to EXIT except that the program counter is set to RETURN_P plus one.

EXIT2   PROCEDURE EXIT WITH TWO SKIPS

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 1	1 1 0

Function:	Exit from procedure at normal exit + 2
Use:	EXIT2
Interruptible:	No

EXIT1 is functionally identical to EXIT except that the program counter is set to RETURN_P plus two.

CACQ   COPY A TO C AND Q

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 0 0	0 0 0	1 1 1

Function:	Copy A- to C- and Q-Registers
Use:	CACQ
Operands:	A : value to load into C and Q
Interruptible:	No

The value contained in the A-Register is copied to the C- and Q-Registers. Bits 14 through 0 are copied into the Q-Register. If bit 15 of the A-Register is one, then CDS is turned off before the next instruction is fetched; otherwise, CDS is turned on.

CBCQ   **COPY B TO C AND Q**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	0	1	1	1

Function:	Copy B- to C- and Q-Registers
Use:	CBCQ
Operands:	B : value to load into C and Q
Interruptible:	No

The value contained in the B-Register is copied to the C- and Q-Registers. Bits 14 through 0 are copied into the Q-Register. If bit 15 of the B-Register is one, then CDS is turned off before the next instruction is fetched; otherwise, CDS is turned on.

CCQA   **COPY C TO Q AND A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0

Function:	Copy C- and Q-Registers to A-Register
Use:	CCQA
Operands:	A gets values in C and Q
Interruptible:	No

The C- and Q-Registers are copied into the A-Register. If CDS is enabled (C = 0), then bit 15 of the A-Register is set to zero; otherwise, it is set to logic one.

CCQB   **COPY C TO Q AND B**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0

Function:	Copy C- and Q-Registers to B-Register
Use:	CCQB
Operands:	B gets values in C and Q
Interruptible:	No

The C- and Q-Registers are copied into the B-Register. If CDS is enabled (c = 0), then bit 15 of the B-Register is set to zero; otherwise, it is set to logic one.

CAZ   **COPY A TO Z**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 0 0	0 0 1	0 0 1

Function: Copy A-Register to Z-Register

Use: CAZ

Operands: Z gets value in A

Interruptible: No

The contents of the A-Register are copied into the Z-Register. The results of setting bit 15 of the Z-Register are undefined.

CBZ   **COPY B TO Z**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 1	0 0 1

Function: Copy B-Register to Z-Register

Use: CBZ

Operands: Z gets value in B

Interruptible: No

The contents of the B-Register are copied into the Z-Register. The results of setting bit 15 of the Z-Register are undefined.

CZA   **COPY Z TO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 0 0	0 0 1	0 0 0

Function: Copy Z-Register to A-Register

Use: CZA

Operands: A gets value in Z

Interruptible: No

The contents of the Z-Register are copied into the A-Register.

CZB   **COPY Z TO B**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0

Function:	Copy Z-Register to B-Register
Use:	CZB
Operands:	B gets value in Z
Interruptible:	No

The contents of the Z-Register are copied into the B-Register.

CIQA   **COPY INTERRUPTED Q TO A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0

Function:	Copy interrupted Q-Register to A-Register
Use:	CIQA
Operands:	IQ : interrupted Q and C values
Interruptible:	No

The A-Register is loaded with the value of the IQ-Register, which is the value of the C- and Q-Registers at the time of the last interrupt or fault.

CIQB   **COPY INTERRUPTED Q TO B**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	1	0	1	0

Function:	Copy interrupted Q-Register to B-Register
Use:	CIQB
Operands:	IQ : interrupted Q and C values
Interruptible:	No

The B-Register is loaded with the value of the IQ-Register, which is the value of the C- and Q-Registers at the time of the last interrupt or fault.

ADQA   **ADD Q TO A**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	0 0 1	1 0 0	0 0 1	0 1 1

Function: Add Q-Register to A-Register

Use: ADQA

Interruptible: Yes

The Q-Register is added to the A-Register (A = A+Q). The ADQA instruction produces undefined results if executed while CDS is disabled.

ADQB   **ADD Q TO B**

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	0 0 0	1 0 1	1 0 0	0 0 1	0 1 1

Function: Add Q-Register to B-Register

Use: ADQB

Interruptible: Yes

The Q-Register is added to the B-Register (B = B+Q). The ADQB instruction produces undefined results if executed while CDS is disabled.

Assembly Language and RTE Implementation

Refer to the Assembly Language and RTE Implementation paragraphs in Chapter 3 for information on implementing the CDS instructions in HP Assembly Language and in an HP RTE-A operating system.

Execution Times

Table 5-1 shows the execution times for the CDS instructions.

Table 5-1. CDS Instruction Execution Times

INSTRUCTION	TIME (μsec)
EXIT	
no segment mapping	2.25
with segment mapping	13.50
EXIT1, EXIT2	
no segment mapping	2.48
with segment mapping	13.73
PCALI (no parameters)	4.95
per parameter passed	1.35
per indirect	0.45
PCALX (includes segment mapping)	19.35
per parameter	1.35
per indirect	0.45
PCALV (includes segment mapping)	20.70
per parameter	1.35
per indirect	0.45
PCALR	7.88
per parameter	1.35
per indirect	0.45
PCALN	7.20
per parameter	1.35
per indirect	0.45
CACQ, CBCQ	1.35
CCQA, CCQB	0.90
CAZ, CBZ	0.90
CZA, CZB	0.90
CIQA, CIQB	1.35
ADQA, ADQB	0.90
SDSP	
display size = 0	2.48
per element of display	0.90

Interrupt System

Interrupt requests can be classified into two types: system level and I/O. The interrupt system receives all interrupt requests and determines which interrupt will be serviced.

In the A400 computer, the interrupt priority of an I/O card is based on the card's proximity to the processor card. The Interrupt Service routine, however, is determined by the interrupting card's select code. This select code is independent of the card's physical location. It is determined by setting six switches on each I/O card, one per select bit code. The select code of the on-board I/O is 77 octal and cannot be changed.

Any device can be selectively enabled or disabled under program control, thus switching the device into or out of the interrupt structure. In addition, the interrupt system is divided into types of interrupts (Table 6-1). Interrupt Type 3 can be enabled or disabled under program control using a single instruction, and interrupt Types 2 and 3 combined can be enabled or disabled using a single instruction.

Power-Fail Interrupt

On the HP 2134A, 2424A, 2434A, and 2484A/B, the computer power supply is equipped with power-sensing circuits. When primary line power fails or drops below a predetermined level while the computer is running, an interrupt to memory location 00004 is automatically generated. Memory location 00004 is intended to contain a jump-to-subroutine (JSB) instruction referencing the entry point of a user-supplied power-fail subroutine. The interrupt capability of lower-priority operations is automatically inhibited while a power-fail subroutine is in process.

A minimum of five milliseconds is available between the detection of a power failure and the loss of usable power supply power to execute a power-fail subroutine; the purpose of such a routine is to transfer the current state of the computer system into memory and then halt the computer. A sample power-fail subroutine is given in Table 6-2. The battery backup (available in the HP 2434A, 2484A/B and 2134A only) supplies enough power to preserve the contents of memory for a sustained line power outage of 15 or more minutes, depending on the amount of memory installed.

Table 6-1. A400 Interrupt Assignments

Select Code (OCTAL)	Interrupt Location	Assignment	Interrupt Type
04	00004	Power Fail Interrupt	2
05	00005	Memory Parity Interrupt	1
06	00006	Time Base Generator Interrupt	3
07	00007	Memory Protect Interrupt	2
10	00010	Unimplemented Instruction Interrupt	1
11	00011	Reserved	
12	00012	Virtual Area Memory Interrupt	4
13	00013	CDS Segment Interrupt	4
14-17	00014-00017	Reserved	
20-77	00020-00077	I/O Card Interrupts	3

There is a switch-selectable option for what action the computer will take upon restoration of primary power. When the switch U1601 S8 is closed, the computer will execute either a loader or the Virtual Control Panel routine, depending on the setting of the Start-Up switches (U1601 S1 to S6).

NOTE

Switch U1601 is mounted on the edge of the A400 board and is not an operator control. The setting of this switch is normally determined by the System Manager prior to or during system installation.

When switch U1601 S8 is open, the automatic restart feature is enabled. After the self-test is executed following the return to normal power levels, an interrupt to location 00004 occurs. This time the power-down portion of the subroutine is skipped and the power-up portion begins. (Refer to Table 6-2.) Those conditions existing at the time of the power-fail interrupt are restored and the computer continues the program from the point of the interruption.

Note that an auto-restart interrupt to location 00004 occurs only if that location's contents are not zero; otherwise, the system is re-booted. This is done so that if power fails and is restored during a boot, an attempt to restart a partially loaded program can be avoided. To enable this to happen, the program being loaded should initially load location 00004 with zero and load the power-fail JSB instruction only when the load is otherwise complete.

Table 6-2. Sample Power-Fail Subroutine

LABEL	OPCODE	OPERAND	COMMENTS
PFAR	NOP		Power Fail/Auto Restart Subroutine.
	SFC	4B	Skip if interrupt was caused by power failure.
	JMP	UP	Power being restored; reset state of system.
DOWN	CLC	0B	Shut down any DMA or I/O.
	STA	SAVA	Save A-Register contents.
	CCA		Set flag indicating that computer was running when power failed.
	STA	PFFLG	
	STB	SAVB	Save B-Register contents.
	ERA,ALS		Transfer E-Register content to A-Register bit 15.
	SOC		Increment A-Register if Overflow is set.
	INA		
	STA	SAVEO	Save E- and O-Register contents.
	LDA	PFAR	Save contents of P-Register at time of power failure.
	STA	SAVP	
	SIMP		
	DEF	SAVI	Same IMAP contents.
	.		
	.		Insert user-written routine to save I/O states.
	.		
	SFS	4B	
	JMP	*-1	Wait in case power comes back up.
UP	LDA	PFFLG	Was computer running when power failed?
	SZA,RSS		
	HLT	4B	No, then halt.
	CLA		Yes, then reset computer Run flag to initial state.
	STA	PFFLG	
	.		
	.		Insert user-written routine to restore I/O devices.
	.		
	LDA	SAVEO	Restore the contents of the E-Register and O-Register.
	CLO		
	SLA,ELA		
	STF	1B	Set C-Register.
	LDA	SAVA	Restore A-Register contents.
	LDB	SAVB	Restore B-Register contents.
	STC	4B	Reset power fail logic for next power failure.
	XJMP		Cross jump to program executing at power failure.
	DEF	SAVI	
	DEF	SAVP,I	
SAVEO	OCT	0	Storage for E and O.
SAVA	OCT	0	Storage for A.
SAVB	OCT	0	Storage for B.
SAVP	OCT	0	Storage for P.
PFFLG	OCT	0	Storage for Run flag.
SAVI	OCT	0	Storage for IMAP.

Note: The memory maps used must be saved and restored, as must (if used) the states of the interrupt mask register, memory protect (conditional restore), and Global Register.

If the computer memory does not contain a subroutine to service the power-fail interrupt, location 00004 should contain a NOP instruction (00 octal).

At the end of a restart routine, consideration should be given to re-initializing the power-fail logic and to restoring the interrupt capability of the lower priority functions.

Parity Error Interrupt

Parity checking of memory is a standard feature in the A400 computer. The parity logic continuously generates correct parity for all words written into memory and monitors the parity of all words read out of memory. Parity can be programmatically set to even parity (STF 05) or cleared to odd parity (CLF 05). Correct odd parity is defined as having the total number of "1" bits in a 17-bit memory word (16 data bits plus the parity bit) equal to an odd value. If a "1" bit (or any odd number of "1" bits) is either dropped or added in the transfer process involving a standard memory array card, a Parity Error signal is generated when that word is read out of memory.

The Parity Error signal generates an interrupt to memory location 00005 if the parity system was previously enabled by an STC 05 instruction. Parity interrupts turn off the system. Location 00005 may contain either a JSB instruction referencing the entry point of the parity error subroutine or a JMP instruction pointing to a HLT instruction. (I/O instructions, including a HLT instruction, may not be in a trap cell). A parity error during a DMA transfer causes an interrupt to the memory location corresponding to the select code of the I/O card making the transfer if the proper bit has been set in the control word.

The memory address of the parity error will be loaded automatically into the parity register which is accessible to the user by a programmed LIA 05 or LIB 05 instruction for bits 0 through 15, and by an LIA 5,C or LIB 5,C for bits 16 through 23.

If a parity error occurs during a read of an instruction, that instruction is executed but memory writes are disabled. When a parity error occurs, it is recommended that the entire program or set of data containing the error location be reloaded.

Memory Protect Interrupt

The memory protect feature protects selected pages of memory against alteration or entry by programmed instructions except those involving the A- and B-Registers.

The memory protect logic, when enabled by an STC 07 instruction, also prohibits the execution of all I/O instructions except those referencing I/O select code 01 (the processor card status register and the overflow register). Execution of all HLTs is prohibited. This feature limits control of I/O operations to interrupt control only. Thus, an executive program residing in protected memory can have exclusive control of the I/O system.

The memory protect logic is disabled automatically by any interrupt (except when the interrupt location contains an I/O instruction) and must be re-enabled by an STC 07 or XJMP instruction at the end of each interrupt subroutine.

Programming rules pertaining to the use of memory protect are as follows (assuming that an STC 07 instruction has been given):

a. Locations 00000 and 00001 in the Execute map are the A- and B-Registers and are not in protected memory. Locations 00000 and 00001 in the DATA1 and DATA2 maps are real memory locations (not the A- and B-Registers) and may reside in protected memory.

b. A user-specified 1024-word page of memory is read and/or write protected by Page Mapping Register instructions described in the Dynamic Mapping chapter.

c. ·Execution will be inhibited and an interrupt to location 07 will occur if any instruction addresses a location in protected memory, or if any privileged instruction is attempted (excluding those addressing select code 01 but not HLT 01). After three successive levels of indirect addressing, the logic will allow a pending interrupt.

Following a memory protect interrupt, the address of the illegal instruction will be present in the violation register. This address is made accessible to the programmer by an LIA 07 or LIB 07 instruction, which loads the address into the A- or B-Register.

Note that DMA operation is not affected by memory protect.

Unimplemented Instruction Interrupt

An unimplemented instruction interrupt (to memory location 00010) is requested when the CPU signals that the last instruction fetched was not recognized by itself or by any other system card. This interrupt provides a straightforward entry to software routines for the execution of instruction codes not recognized by the computer hardware. The unimplemented instruction interrupt must receive immediate service in order to recover the instruction code that caused it. For this reason, and because it is desirable to permit the use of unimplemented instructions anywhere, the unimplemented instruction interrupt is never inhibited.

Time Base Generator Interrupt

A time base generator interrupt request is made when the CPU signals that its internal clock divider chain has rolled over. The clock divider is set to roll over at 10-millisecond intervals for maintaining a real-time clock. The interrupt occurs through location 00006 and can be masked (inhibited) by using bit 1 of the interrupt mask register. (The interrupt mask register allows interrupts from the TBG and the I/O cards to be selectively masked. For details on the interrupt mask register, refer to the HP 1000 A/L-Series Computer I/O Interfacing Guide, part no. 02103-90005.) The TBG can be turned on by an STC 06 instruction, and turned off by a CLC 06 or CLC 00 instruction.

Virtual Memory Area Interrupt

During the execution of a VMA instruction, the hardware may determine that the desired VMA address does not reside in physical memory and needs to be loaded from disc. This causes a VMA interrupt to memory location 000012 (octal). This interrupt can occur only when Code and Data Separation (CDS) is enabled.

CDS Segment Interrupt

During the execution of a CDS instruction, the hardware may determine that a desired CDS segment does not reside in physical memory and needs to be loaded from disc. This causes a CDS segment interrupt to memory location 000013 (octal).

Input/Output Interrupt

Interrupt locations 20 through 77 (octal) are reserved for I/O devices. In a typical I/O operation, the computer issues a programmed command such as Set Control/Clear Flag (STC,C) to one or more external devices to initiate an input (read) or an output (write) operation, via either programmed I/O or DMA. While the I/O card is in the process of transferring data, the computer may be either running a program or looping, waiting for a flag to be set. Upon completion of the read or write operation, the interface flag is set. If the corresponding control bit is set, the interface will interrupt. Its request will be passed through a priority network so that only the highest priority interrupting device receives service. The computer acknowledges the interrupt and the highest priority device receives service when the current instruction has finished executing, except under the following circumstances:

a. Interrupt system disabled or interface card interrupt disabled (or masked).

b. JMP indirect, JSB indirect, XJMP, or XJCQ instruction not sufficiently executed. These instructions inhibit all interrupts except power-fail or memory protect until the succeeding instruction is executed. After three successive levels of indirect addressing, the logic will allow a pending I/O interrupt.

c. A DMA (direct memory access) data transfer is in process.

d. Current instruction is any I/O group instruction. The interrupt in this case must wait until the succeeding instruction is executed.

After an interface card has been issued a Set Control (STC instruction) and its flag bit becomes set, all interrupt requests from lower-priority devices are inhibited until this flag bit is cleared by a Clear Flag (CLF) instruction, or until control is cleared by a Clear Control (CLC) instruction. A service subroutine in process for any device can be interrupted only by a higher-priority device; then, after the higher-priority device is serviced, the interrupted service subroutine can continue. In this way it is possible for several service subroutines to be in the interrupt state at one time; each of these service subroutines will be allowed to continue after the higher-priority device is serviced. All such service subroutines normally end with a JMP indirect or XJMP instruction to return the computer to the point of interrupt.

Note that interrupt trap cells must contain a JMP or JSB instruction because maps change on interrupt.

Interrupt Priority

The interrupt servicing priority among the system interrupts is as follows:

1. Parity error (select code 5).

2. Unimplemented instruction (select code 10).

3. Memory protect (select code 7).

4. Power-fail (select code 4).

5. Time base generator (select code 6).

6. OBIO (select code 77).

7. I/O interrupts (select codes 20 through 76).

8. Virtual Memory Area (select code 12) and CDS Segment (select code 13).

Central Interrupt Register

Each time an interrupt occurs, the address of the interrupt location is stored in the central interrupt register. The contents of this register are accessible at any time by executing an LIA 04 or LIB 04 instruction. This loads the address of the most recent interrupt into the A- or B-Register.

Processor Status Register

The processor status register is two registers: one for input and one for output. The input register shows the status of the BOOT SEL switches and is read into the upper eight bits of the A- or B-Register by an LIA/B 01 instruction. The switch, bit, and function relationships are as follows:

SWITCH	BIT	MEANING
S1	8	Boot select
S2	9	Boot select
S3	10	Boot select
S4	11	Boot select
S5	12	VCP program select
S6	13	ENQ/ACK select
S7	14	Not used
S8	15	Auto-restart enabled (1)/disabled (0)

The output register drives the CPU board LEDs. The output of the lower eight bits of the A- or B-Register are sent to the LEDs by an OTA/B 01 instruction. A logic 1 in either register lights the corresponding LED.

Interrupt Type Control

I/O address 00 is the master control address for Type 3 interrupts (TBG and I/O cards). An STF 00 instruction enables Type 3 interrupts and a CLF 00 disables Type 3 interrupts. (Type 3 interrupts are disabled when power is initially applied.) I/O address 04 is the master control address for Type 2 interrupts (power-fail and memory protect) and Type 3 interrupts combined. An STC 04 instruction enables Type 2 and 3 interrupts and a CLC 04 disables Type 2 and 3 interrupts.

The Type 2 and 3 interrupt mask from I/O address 04 is a different Type 3 mask than the Type 3 mask at I/O address 00. If either of these two masks are set, Type 3 interrupts will be disabled. In addition to these two interrupt masks, the Time Base Generator flag interrupt can also be masked by bit 1 of the Interrupt Mask Register. If any of these three masks are set, then the TBG flag interrupt will be disabled.

Instruction Summary

Table 6-3 is a summary of instructions for select codes 00 through 07. For a summary of instructions used with the I/O cards, refer to an I/O card reference manual.

Table 6-3. Instructions for Select Codes 00 through 07

Instruction	Function	Instruction	Function
STC 0	NOP	STC 4	Enable type 2 and 3 interrupts
CLC 0	System reset	CLC 4	Diable type 2 and 3 interrupts
STF 0	Enable type 3 interrupts	STF 4	NOP
CLF 0	Disable type 3 interrupts	CLF 4	NOP
SFS 0	Skip if type 3 interrupts enabled	SFS 4	Skip if power is stable
SFC 0	Skip if type 3 interrupts disabled	SFC 4	Skip if power going down
LI* 0	Load from interrupt mask register	LI* 4	Load from central interrupt register
MI* 0	NOP	MI* 4	NOP
OT* 0	Output to interrupt mask register	OT* 4	Output to central interrupt register
STC 1	NOP	STC 5	Enable parity error interrupts
CLC 1	NOP	CLC 5	Disable parity error interrupts
STF 1	Same as Set Overflow (STO)	STF 5	Set parity sense to even parity
CLF 1	Same as Clear Overflow (CLO)	CLF 5	Clear parity sense to odd parity
SFS 1	Same as Skip if Overflow set (SOS)	SFS 5	Skip if parity sense is even
LI* 1	Load from processor status register	SFC 5	Skip if parity sense is odd
MI* 1	Merge from processor status register	LI* 5	Load from parity register (bits 0-15)
OT* 1	Outputy to processor status register	LI* 5,C	Load from parity register (bits 16-23)
		MI* 5	NOP
STC 2	Enable break feature	OT* 5	NOP
CLC 2	NOP		
STF 2	Disable global register	STC 6	Turn on time base generator
CLF 2	Enable global register	CLC 6	Turn off time base generator
SFS 2	Skip if global register disabled	STF 6	Set time base generator flag
SFC 2	Skip if global register enabled	CLF 6	Clear time base generator flag
LI* 2	Load from global register	SFS 6	Skip if time base generator flag set
MI* 2	NOP	SFC 6	Skip if time base generator flag clear
OT* 2	Output to global register (Note 1)	LI* 6	NOP
		MI* 6	NOP
STC 3	NOP	OT* 6	NOP
CLC 3	NOP		
STF 3	NOP	STC 7	Enable memory protect
CLF 3	NOP	CLC 7	NOP
SFS 3	NOP	STF 7	NOP
SFC 3	NOP	CLF 7	NOP
LI* 3	Load from PSAVE	SFS 7	NOP
MI* 3	NOP	SFC 7	NOP
OT* 3	Output to PSAVE	LI* 7	Load from violation register
LI* 3,C	Load from ROM P	MI* 7	NOP
OT* 3,C	Output to ROM P	OT* 7	NOP

* = A or B
Note 1. An OTA/B 2 with A/B equal to 1 thru 7, establishes a diagnose mode; refer to Chapter 8 for details.

On-Board I/O (OBIO)

General Description

This chapter describes the On-Board Input/Output subsystem (OBIO), a set of four serial I/O ports that reside on the A400 board. OBIO is unique to the A400 computer. The A400 also supports I/O using standard interface cards common to the A/L-Series family. For more information on the cards, refer to the Input/Output (I/O) System chapter of this manual.

OBIO is comprised of an I/O processor, the I/O master circuitry, and four serial port processors. The I/O master and the single DMA channel that it provides are shared by the four port processors. The port processors are implemented in 63701V microprocessor units (MCUs).

The DMA machine outputs characters much faster than the maximum baud rate of peripherals, therefore the maximum data throughput is equal to the baud rate, minus a slight per buffer overhead time. The baud rates supported are 300, 1200, 9600 and 19.2k, and 76.8k baud. The port processors buffer data until a user definable special character is detected, a carriage return is detected, or until its buffer is full (approximately 95 characters). At that time, the port processor requests service from the CPU. The port processor also does backspace processing to minimize the interrupts that the CPU must process. The serial ports support RS-232, RS-422, RS-423, and V.24/28. Two of the four ports support modem control lines.

Normal software interfacing to the four serial OBIO ports is handled by the device drivers DDC00 and DDC01; and interface driver ID400, supplied with the RTE Operating System. For more information, refer to the RTE-A Driver Reference Manual, part no. 92077-90011.

Processor Description

A400 I/O is designed around the Hitachi HD63701V1 8-bit single-chip microcomputer unit (MCU). The HD63701V1 comes complete with 192 bytes of on-board RAM and 4096 bytes of on-board ROM. This processor is used in single chip mode (mode 7), which includes four multiplexed ports. Port 1 and port 4 are 8-bit parallel I/O ports, port 2 is the timer (unused) and Serial Communications Interface (SCI), and port 3 is an 8-bit parallel I/O port with two control lines, Input Strobe 3 (IS3-) and Output Strobe 3 (OS3-). The input frequency (Fxtal) = 4.9152 MHz, which is internally divided to give a Frequency of Operation (Fo) = 1.2288 MHz. This inverts to provide a cycle time of 0.8138 microseconds. The internal clock will therefore overflow at 53.333 milliseconds. An external clock may be connected to each port processor. Although any frequency may be used up to 153K baud, the A400 clock is a 19.2K baud input. This clock is derived by a 19.2K baud divider on the card. This rate is the maximum input data rate supported, and worse case calculations are made using this rate.

MCU Pin-Out

```
                    Vss --| 1        ⊓       40 |-- E
                   XTAL --| 2      ⎵  ⎵      39 |-- SC1   <-   (ISC3-)
                  EXTAL --| 3                38 |-- SC2   ->   (OS3-)
                    NMI- --| 4                37 |-- P30         .
                   IRQ1- --| 5                36 |-- P31         .
               Vpp/RES- --| 6                35 |-- P32         .
                   STBY- --| 7                34 |-- P33   <->  (DATA PORT)
          ("1") -> P20 --| 8                33 |-- P34         .
          ("1") -> P21 --| 9                32 |-- P35         .
    (19.2K CLOCK) -> P22 --| 10               31 |-- P36         .
    (RECEIVE DATA ) -> P23 --| 11               30 |-- P37         .
   (TRANSMIT DATA ) -> P24 --| 12               29 |-- P40   <-   (CONTROL TRANSFER)
   (MODM CONNECT-) <- P10 --| 13               28 |-- P41   <-   (MCU OUTPUT)
            (CTS-) <- P11 --| 14               27 |-- P42   ->   (DATA IN AUX BUF)
            (DSR-) <- P12 --| 15               26 |-- P43   ->   (READ CONFIGURATION)
        (DTR/RTS-) <- P13 --| 16               25 |-- P44   <-   (PORT ID 0)
             (RI-) <- P14 --| 17               24 |-- P45   <-   (PORT ID 1)
             (CD-) <- P15 --| 18               23 |-- P46   ->   (SLRQ-)
    (AUTO ANSWER-) <- P16 --| 19               22 |-- P47   ->   (PORT INTERRUPT-)
   (FAIL SELF-TEST) <- P17 --| 20               21 |-- Vcc
```

OBIO Features

The A400 supports four serial I/O ports, controlled by port processors A through D (PPA through PPD). PPB and PPC support modems, while PPA and PPD will not. PPA is the only port which may be configured as the Virtual Control Panel. The DMA machine outputs characters much faster than the baud rate, so the maximum data throughput is equal to the baud rate, minus a slight per buffer overhead time. The on-board MUX can detect framing and receiver overrun errors. The baud rates supported are 300 baud, 1200 baud, 9600 baud, and 19.2K baud (using external clock). 76.8K baud is available with RS-422. The SCI supports a standard mark/space (NRZ) format, with one start bit and one stop bit.

I/O Master

The I/O Master portion of the OBIO performs all of the program functions described in the A/L-Series I/O Interfacing Guide. The select code of the OBIO is set at select code 77 and cannot be changed.

Programming VCP

When the Virtual Control Panel (VCP) is communicating with the MCU, it only talks to port A. The VCP driver can communicate with the MCU using VCP calls (put_char and get_char).

Break Detection

Breaks are detected by the MCU if two framing errors are detected within approximately 100 milliseconds. A break to port A will cause SLRQ- to be asserted. The system will ignore this input unless the A400 card was selected as the VCP card.

MCU Default Configuration for VCP

A hard or soft reset will cause VCP to come up with the following parameters:

- Baud Rate: 9600
- ENQ/ACK disabled
- Modem Control disabled
- No echo of data without destination
- No FIFO buffering

VCP Write

Since the VCP driver uses single character transfers to send data, a similar function is implemented by the MCU. This transfers only one character to the external device, and may be used when the system is or is not in VCP mode. For a VCP single character transfer, a special bit is implemented in the write control word. If this bit is set, the control designates a single character write, and no data transfer is required. Refer to the Identity 10: EXEC Write Request section of this chapter.

VCP Read

The MCU should be configured for no buffering, have no read pending, and be configured to echo characters with no destination. Each character received from the external device then causes an MCU status available interrupt (setting the appropriate bit in status register 32). This MCU status word is flagged as data without a destination, and it is stored in bits 7-0. The VCP driver performs all editing functions.

Driver Registers

OBIO uses registers 30, 31, and 32 to communicate with the CPU. They are summarized as follows:

- Register 30 is a data register that appears to the MCU as an 8-bit register. Only the eight least significant bits are sent to the MCU when the CPU performs an OTA 30B.

- Register 31 is a control register that allows the driver to control the ports. The driver writes to this register, instructing the designated MCU that the next word will be an MCU control word, MCU status word, or actual data.

- Register 32 is a status register. Bits 7-4 reflect the pass/fail result of the port's self-test. Bits 3-0 are port interrupt bits that are cleared if the corresponding MCU is requesting interrupt service, and is independent of the value of control register 31.

OBIO Data Transfer

This section describes the typical steps the driver must perform in order to read and write a string to an external device. In a DMA quad, four words are required:

Word 1   is written to DMA control register 21.
Word 2   is written to DMA control register 31 (and MCU).

Word 3   is written to DMA control register 22.
Word 4   is written to word/byte control register 23.

The general bit definitions for DMA control word 1 are:

15	14	13	12	11	10	9	8	7	6	5	4	0
CONT	DVCMD	BYTE	RES	CINT	REM	FOUR	AUTO	IN	Various		ADD EXT BUS	

CONT   Continue: Enables a DMA reconfiguration upon completion of a self-configured DMA transfer.

DVCMD   Device Command: Issues a device command signal for each data element transferred. This is required for proper handshaking.

BYTE   Byte/word Transfer: Conducts DMA transfers in byte mode. This is required for proper handshaking.

RES   Residue: Writes word/byte count back into memory.

CINT   Completion Interrupt: Inhibits DMA completion interrupt. (Do not set flag 30B when finished.)

REM   Remote: Enables remote (non-standard) memory for DMA transfer (not used in RTE-A).

FOUR   Fetch Four Control Words: Causes DMA self-configuration to fetch four control words, that is, three DMA control words and one I/O card control word.

AUTO   Automatic: Initiates the first data transfer once DMA is configured to output, without waiting for an SRQ-. For input transfers, it enables a Device Command signal after the last data element is transferred. Note that this in only in effect during self-configuring DMA, thereby requiring all output DMA transfers to be self-configured. AUTO must be on for output, and off for input.

IN   Transfer In: Performs DMA transfers from I/O devices to memory.

Various: User definable.

ADD EXT BUS: Five bits that allow DMA access to physical memory by referencing one map set of 32 registers each.

CPU to External Device Transfers

To initiate a write to port processor A, a control word "identity 10: EXEC Write Request" followed by the actual data must be sent to the MCU:

QUAD #1:

    Word #1 – 165400B (CONT,DVCMD,BYTE,CINT,FOUR,AUTO,MAP SET 0)
    Word #2 – 000201B (send ctrl word to PPA, enable interrupts on port A)
    Word #3 – xxxxxxB (address of MCU control word)
    Word #4 – 177776B (two bytes)

QUAD #2:

    Word #1 – 065407B (DVCMD,BYTE,CINT,FOUR,AUTO,MAP SET 7)
    Word #2 – 000001B (send actual data to PPA, enable interrupts on port A)
    Word #3 – xxxxxxB (address of data in map set 7)
    Word #4 – 177633B (101 bytes)

To initiate a longer transfer than the MCU's memory will allow, read and write requests must be broken into several requests small enough for the MCU's buffer. Note that the same action may be taken by the CPU using a STC/SFC structure, but the DMA quad method is simpler and more code efficient.

MCU/Driver Communication

There are four modes of communication with the MUX driver:

- Device to MUX driver through the MCU
- MUX driver to device through the MCU
- MUX driver to MCU
- MCU to MUX driver

Each mode is selected by control register 31 bits CNTRL XFER and MCU OUTPT.

DMA Device Write

CNTRL XFER = 0, MCU OUTPT = 0. This type of word at control register 31 causes the MCU to assume that the data available at data register 30 will be actual data to be sent to the SCI. If the driver requests a write transfer to a port's device, it initiates the transfer with a DMA quad. The quad first configures the MCU for a write transfer, then follows with the data to be sent to the device. The driver does not send more than one MCU buffer full of data at a time. If the total number of characters is greater than one MCU buffer, the driver must wait for the MCU to interrupt, indicating it is finished with the previous buffer, then transfer the next buffer of data.

DMA Device Read

CNTRL XFER = 0, MCU OUTPT = 1. This type of word at control register 31 indicates a transfer initiated by the driver as a response to an MCU status word indicating the MCU has a full buffer. The driver sets up a DMA read transfer for the number of characters the MCU indicated. Chain DMA may be used in this case, since a new read configuration word may then be sent to the MCU (Identity 1 EXEC read MCU control word), followed by the actual DMA transfer. The data will always arrive in even pairs, since the last buffer is padded with a null/space if needed to end on word boundaries.

MCU Control Words

CNTRL XFER = 1, MCU OUTPT = 0. If the MCU OUTPT bit is cleared and the CNTRL XFER bit is set, the MCU treats the data available at data register 30 (port 3) as an MCU control word.

Identity 0: EXEC Read Request

This type of control word request is normally completed upon receiving an MCU status word of type "READ DATA BUFFER READY".

UPPER BYTE: READ CHARACTERISTICS

15	14	13	12	11	10	9	8
IDENT = 0	START CHAR	FIRST BUFFR	XPRNT XFER	BLOCK XFER	ECHO /PAGE	LAST BUFFR	BINRY /A.H.

LOWER BYTE = MAXIMUM NUMBER OF CHARACTERS TO READ

7	6	5	4	3	2	1	0
MSB							LSB

Bit 15: CONTROL WORD TYPE IDENTIFICATION = 0

Bit 14: START CHAR
This bit is used for editing. If you backspace into a previous buffer, the MCU sends indication of this, and saves the byte in the start of MAINBUF area. If this bit is set, it indicates the driver:

- ignored the character supplied with notification of backspaces,
- requests the character to be placed in the start of MAINBUF, and
- requests the next character to be placed in location SMAINBUF+1.

Bit 13: FIRST BUFFR

This bit is set if this is the first buffer of a (possibly) multiple buffer transfer. It instructs the MCU to ignore excessive backspaces during a normal ASCII read if set, and to send "delete" status words during a normal ASCII read if cleared. This bit also indicates when to send a DC1 in HP protocol mode.

When in block mode (setting bit 11), bit 10 takes on the meaning of page (1) or line (0) mode transfers.

If a control 17B request with bit 8 set has been made, transparent ASCII reads change to testing for a defined terminator, and no echo of CRLF when a terminator is detected. Bits 12 and 8 are the transparent and binary bits, respectively. They describe the type of read requested as described in the following table:

Reads do not return the terminating character. If an odd number of characters is requested, or the read is terminated on an odd boundary, the last character will be padded with a null (octal 0) for binary read transfers, or a space (octal 40) for non-binary read transfers.

The terminating condition is indicated in the appropriate MCU status word.

When FIFO buffering is enabled, no data is echoed or processed. The data is processed only when the read is actually posted to the port. Therefore, data will not be echoed until the read is posted, causing the entire buffer to be dumped at once.

Bit 10: ECHO/PAGE

If echo is set, all characters will be echoed as soon as they are received, but with these exceptions:

- A \CRLF will be echoed for DEL
- ENQ/ACK will not be echoed if in ENQ/ACK handshaking mode
- XON/XOFF will not be echoed if in XON/XOFF mode.

If the request is block mode (bit 11, BLOCK XFER set), this has the meaning of page mode (if set), or line mode (if cleared). Block mode indicates a forced no echo.

Bit 9: LAST BUFFR

This bit is set to notify the MCU that this is the last buffer of a (possible) multi-buffer transfer. It is used in ASCII reads to indicate that all data will be ignored until the terminating condition is met. If this bit is cleared, the MCU leaves itself in "double buffer" mode, until the read terminates, to save all data until the next partial read request is received.

Bit 8: BINRY / A.H.

If the BLOCK and PAGE bits are set, an auto home/lock keyboard (esc H ESC c) occurs. In other cases, it flags a binary transfer (see table).

IF THE BLOCK BIT = 0:			
READ TYPE	**TR BI**	**TERMINATING COND.**	**DESCRIPTION OF READ**
NORMAL ASCII	0 0	Detection of Cr, or EOT. No termination on character count.	Allow editing (BS & DEL); if echo bit set -> echo each character and echo CRLF for CR; if echo bit clear -> echo CRLF for CR. Note that editing occurs at last valid character, not necessarily last entered character. Echo "\CRLF" for DEL if echo is enabled.
XPARENT ASCII	1 0	Detection of user defined terminator. No termination on count.	If echo bit set -> echo each character and echo CRLF for CR. If echo bit clear -> echo CRLF for CR. (Or as per user def. term.)
NORMAL BINARY XPARNT BINARY	X 1	Satisfy character count.	If echo bit set -> echo each character.

IF THE BLOCK BIT = 1 (TR IS IGNORED):			
READ TYPE	**PAGE**	**TERMINATING COND.**	**DESCRIPTION OF READ**
BLOCK LINE	0	Detection of Cr. No termination on character count.	Inhibit echo. If first character received is a DC2(+CR) flush bufr, then send a DC1. If TR = 0, echo a CRLF when CR is detected.
BLOCK PAGE	1	Detection of RS. No termination on character count.	Inhibit echo. If first character received is a DC2, flush ten send a DC1. If TR = 0, echo a CRLF when RS is detected.

Bits 7-0: MAXIMUM NUMBER OF CHARACTERS TO READ
This is only valid if LAST BUFFR = 1. This byte tells the MCU the maximum number of characters to read. If status 0 characters read is returned (user immediately entered terminator), a driver request for that buffer is invalid, since it does not exist. If a request is made which is larger than the maximum buffer size, the buffer size request is reduced to maximum size.

Identity 10: EXEC Write Request

This type of control word request is normally completed by receiving an MCU status word of type "WRITE DATA BUFFER EMPTY". This type is used if the driver (or VCP) is instructing the MCU to perform a DMA data transfer or a non-DMA transfer of a single character. The second feature is useful in VCP single character mode to send characters to the external device as they become available, without requiring the use of DMA, or if the driver transfers data in byte mode. Identity 10 also allows the driver to configure the appending of CRLF's.

UPPER BYTE: WRITE CHARACTERISTICS

15	14	13	12	11	10	9	8
IDENTITY 1 0		WRITE -READ	HOLD CRLF	FORCE EN/AK	SINGL CHAR	LAST BUFFR	HOLD EN/AK

LOWER BYTE = SINGLE CHARACTER/DMA CHARACTER LENGTH

```
  7    6    5    4    3    2    1    0
┌────┬────┬────┬────┬────┬────┬────┬────┐
│MSB │    │    │    │    │    │    │LSB │
│    │    │    │    │    │    │    │    │
└────┴────┴────┴────┴────┴────┴────┴────┘
```
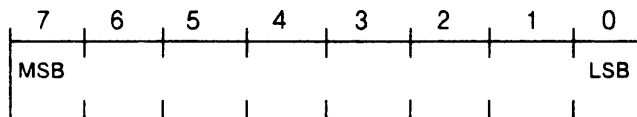
Bits 15-14: Control Word Type Identification = 01

Bit 13: WRITE-READ
This bit is used to indicate a write-read is in effect, and to buffer data until the read is available.

Bit 12: HOLD CRLF (TR BIT)
If this bit is set, no CRLF will be appended at the end of the transfer.

Bit 11: FORCE EN/AK
If this bit is set, an ENQ/ACK handshake must occur before the current write, if HOLD EN/AK is cleared.

Bit 10: SINGL CHAR
If this bit is set, the MCU does not expect any IS3– handshakes or data from data register 30. It will only output the character contained in bits 7-0 and will not process bits 13-8.

Bit 9: LAST BUFFR
This bit is used with the BINRY XFER bit. If LAST BUFFR = 1, BINRY XFER = 0, and the last valid data character is a "_", then the "_" will not be printed, and the CRLF will not be appended.

Bit 8: HOLD EN/AK (BI BIT)
If this bit is set, no ENQ/ACKs (unless forced) will occur during the current transfer.

Bits 7-0: DATA COUNT/CHARACTER
These eight bits contain the character to be transferred when bit 13 (SINGL CHAR) is set. When bit 13 is not set, it contains the number of characters the DMA transfer will contain. The MCU uses this number to determine the end of valid data in its buffer, testing for premature termination of DMA. This number may be 0 if you want to force any conditions such as ENQ/ACK control, or just write a CRLF.

Identity 11: MCU Control Requests

Identity 11 is used for all requests to the MCU for the various types of control required.

Identity 11-0000: Port3 Diagnostics

This control request is normally completed upon reading the echoed data byte. *Note that the proper sequence must be strictly observed.*

Bits 15-10: Control Word Identification = 110000

When entering this mode, the next word written to port 3 will be echoed back for verification. The proper sequence is:

1. Write to control register 31 selecting the desired port.

2. Write this MCU control word to data register 30.

3. Verify PINT bit is off at status register 32.

4. Write data byte to data register 30.

5. Wait for PINT bit on at status register 32 and OS3– strobe.

6. Read and compare data byte from data register 30.

7. Verify PINT bit is off at status register 32.

Note that this only occurs once per sequence, and the sequence must be completed in order for the MCU to continue processing.

Identity 11-0001: Load Executable Code

This control request requires the buffer to fill, and will wait indefinitely for it to fill, then execute the buffer.

Bits 15-10 Control Word Identity = 110001.

Identity 11-0010: Return MCU Dynamic State

This MCU control word returns an eight-bit MCU status byte. The requested byte is returned during the next IS3/OS3 handshake, regardless of the state of the INXFER and CNTRLXFER bits.

Bits 15-10: Control Word Identity = 110010

Bits 7-0: Memory Location

Returns dynamic status. The MCU returns 16 bits, with the eight most significant bits indicating the length of FIFO buffering data, and the least significant bits returning the contents of the location indicated by bits 7-0. If the requested memory location was P3DATA (location $0006H) the MCU will return the revision code as an offset to 4.00 (such as, 123 decimal = rev 5.23). If the requested memory location was RDR (location $0012H) the MCU will return the value of the location ROMFUDGE (used to force simple checksum of all ROM to equal 0 for self-test verification. Refer to the Firmware Architecture section for exact values of other memory locations.

The requested value is immediately returned by the MCU, so this control word must always be followed by a status word request, which is immediately returned without a status type.

Identity 11-0011: Undefined

This control word is used for future enhancements.

Identity 11-0100: De-Assert SLRQ– Line

Bits 15-10: Control Word Identity = 110100

This control word causes the port to remove SLRQ–. This is used by VCP when the "disable break" switch has been selected, to remove the condition which initiated VCP.

Identity 11-0101: Reset

Bits 15-10: Control Word Identity = 110101

Bit 0: HARD/SOFT RESET

    0: HARD RESET. This type of control word invokes a delayed response, an MCU status word, "SELF-TEST RESULT". It initiates self-test, resets all pointers, and flushes data. The self-test result is set upon each execution from restart, and is saved internally as an MCU status word. When finished, the MCU asserts the MCU status available interrupt line, and returns a type 101 self-test result word.

    1: SOFT RESET (ABORT). Any reads or writes are aborted, and any pending status words are removed. The buffers are re-initialized, and all data is lost. This is the same as a hard reset, but no self-test results are returned, and no status word becomes available. The driver then re-initializes the card.

Identity 11-0110: Enter VCP Mode

Bits 15-10: Control Word Identity = 110110

Enters VCP mode (single character transfer mode) with echo enabled, and no FIFO buffering.

Identity 11-0111: Set Protocol

Bits 15-10: Control Word Identity = 110111

This control word instructs the MCU to set the port protocol to be used in multiple character transfers.

Bits 3-0: Selected Protocol

0000:	TTY protocol
XXX1:	bi-directional XON/XOFF protocol
XX1X:	HP protocol
X1XX:	non-HP CPU protocol
1XXX:	hardwire handshake protocol

For more information, refer to the Serial I/O Drivers (Rev. 4010) section of the RTE-A Driver Reference Manual, part no. 92077-90011.

Identity 11-1000: Define User Terminator

This control word instructs the MCU to set the user-defined character.

Bits 15-10: Control Word Identification = 111000

Bit 8: Default/Set Terminator

1: changes to terminate on defined character in bits 7-0 (no echo CRLF).

0: changes to normal ASCII read (terminate on CR, echo CRLF).

Bits 7-0: Special Character

These bits programmatically set the special user defined terminator.

Identity 11-1001: Dump FIFO

This control word is used for full duplex implementations.

Bits 15-10: Control Word Identification = 111001

If this option is set, the MCU will quickly process all data in the FIFO buffer as binary data, and transfer the data from the FIFO buffer to the main buffer. Since this is used to support the Full Duplex Binary Read mode of operation, the transfer is only on word boundaries, and only up to MAINBUFLEN characters. The MCU returns a "READ BUFFER AVAILABLE" status word, with the most significant byte = 0. This control word must always be followed by a status word request, which will be returned, with no type.

Identity 11-1010: Set Baud Rate

This type of MCU control word initiates SCI configuration and causes the communication channel to reset, and therefore should only be done upon initialization.

Bits 15-10: Control Word Identification = 111010

Bits 3-0 Desired Baud Rate

B3	B2	B1	B0	Baud Rate Selected
0	1	1	0	300 baud
0	1	1	1	1200 baud
1	0	1	1	9600 baud
1	1	0	0	19.2K baud (external clock)
1	1	1	0	76.8K baud
any others				9600 baud

Identity 11-1011: Modem Control

Bits 15-10: Control Word Identification = 111011

Bits 1-0: Modem Configuration

Bit 1:   Set to allow RI interrupts

Bit 0:   Set to allow modem connection (DTR and RTS -> true). Note that the MCU can be instructed to assert DTR and RTS without allowing connection (that is, connect status word) by setting any of bits 7-2.

DTR may be forced true without making firmware attempt to connect by setting any of bits 7-2.

Identity 11-1100: FIFO Buffering (Input Buffering)

Bits 15-10: Control Word Identity = 111100

Bit 1: Enable FIFO buffering

1:   Enable FIFO buffering. Flush Auxbuf. This can be used when already in FIFO buffering to flush the auxiliary buffer.

0:   Disable FIFO buffering. In both cases if the port transmitted an XOFF, an XON will be transmitted at this point.

Bit 0: Interrupt On Each Character

1:   If bit 1 is set, then interrupt (schedule program in driver) upon each character. This feature is useful to allow the character which schedules the program to be saved.

0:   No interrupts.

Identity 11-1101: Disable Break

Bits 15-10: Control Word Identity = 111101

This control word prevents the MCU from asserting SLRQ-, and invoking VCP.

Bit 0: Break Enable Control:

    0:   Disable SLRQ-. P46 configured as an input.

    1:   Enable SLRQ-. P46 configured as an output.

Identity 11-1110: Set MCU State

This type of control request sets internal flags in the MCU to initiate one-time events.

Bits 15-11: Control Word Identity = 111110

Bit 0: Forces Read Terminate – Force a pending read to complete.

Bit 1: Performs an ENQ/ACK handshake (even if in non-HP protocol).

Bit 2: Generate Break
    This flag causes no data to be transmitted temporarily while a 250 millisecond break is transmitted. Note that this holds off processing of data for the entire time, only allowing interrupts to be serviced.

Bit 3: Speed Sensing
    Setting this bit causes the MCU to reply with a status word of type "speed data" status word to be returned. The MCU looks for a CR. It echoes a CRLF when the CR is detected at an acceptable baud rate. If in HP protocol mode, this is done with ENQ/ACK handshakes, and gives 0.5 seconds for a response at each baud rate.

Bit 4: Send an ASCII "XOFF"

Bit 5: Send an ASCII "LF"

Bit 6: Send an ASCII "CR"

Bit 7: Send an ASCII "DC1" ("XON")

Identity 11-1111: Undefined

This control word is used to allow future enhancements.

Table Of Control Word Responses

This table explains which control words are terminated immediately (that is, no status word will be returned) and which will terminate later.

IDNTY	MCU CONTROL WORD	DESCRIPTION OF MCU RESPONSE
0	EXEC Read Request	MCU status word: "Read Buffer Available"
10	EXEC Write Request	MCU status word: "Write Data Buffer Empty"
11-0000	Port 3 diagnostics	See text
11-0001	Load executable code	See text
11-0010	Return MCU status	MCU status word: 16 bit result, returned immed.
11-0100	Remove SLRQ–	Immediate termination
11-0101	Reset	MCU status word only on hard reset
11-0110	Enter VCP mode	Immediate termination
11-0111	Set protocol	Immediate termination
11-1000	Define user term.	Immediate termination
11-1001	Dump FIFO	MCU status word: 16 bit result, returned immed.
11-1010	Set baud rate	Immediate termination
11-1011	Modem control	Immediate termination (modem has asyc. ints)
11-1100	FIFO buffering	Immediate termination
11-1101	Disable break	Immediate termination
11-1110	Set MCU state	Immediate termination

MCU Status Words

CNTRL XFER = 1, MCU OUTPT = 1. These MCU status words are the first data sent to the driver after the MCU sends an MCU status available interrupt request (setting port 4 bit 0) and the driver handshakes with an OS3 strobe. The first most significant four bits indicate why the MCU sent the status available interrupt request, and the least significant 13 bits contain further information. The MCU removes the status available line (PINT) when the status word is read, upon "ENTER VCP MODE", or upon receiving a "Reset" control word. If at any time another status word becomes available while the previous has not been read, the MCU declares an internal error and initiates a soft reset.

UPPER BYTE: TYPE OF MCU STATUS AVAILABLE INTERRUPT

```
  15   14   13    12   11   10    9    8
|----+----+----|----+----+----+----+----|
| TYPE OF DATA |  TYPE SPECIFIC DATA BITS |
| MSB     LSB  | MSB                 LSB  |
|--------------+--------------------------|
```
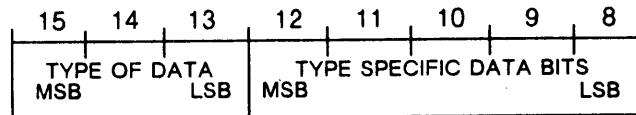
LOWER BYTE = EXTENDED DATA

```
   7    6    5    4    3    2    1    0
|----+----+----+----+----+----+----+----|
|            EXTENDED DATA                |
| MSB                              LSB    |
|_____|
```
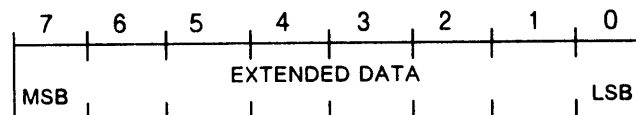
TYPE 000: Read Data Buffer Ready

Data buffer is ready. Padded with nulls for binary reads, spaces for ASCII reads.

Bit 12: Set if padding is in effect.

Bit 11: Set if a force read terminate. (Bit 9 is set also.)

Bit 10: Set if delete detected and not FIRST BUFFR.

Bit 9: Set if a terminator was detected, cleared if terminated on count.

Bit 8: Set if terminator = EOT (indicates no data to be transferred).

Bits 7-0: Number of words available (in words, due to padding).

TYPE 001: Write Data Buffer Empty

The last character in the write buffer has been sent to the external device. The next buffer may be transmitted.

TYPE 010: Destinationless. Char/Speed Sensing

Bit 8:
   1: Speed sensing enabled

   Bits 7-0 contain the word for RMCSR used by the MCU to match the user's baud rate.

0E hex:	76800 baud
0B hex:	9600 baud
07 hex:	1200 baud
06 hex:	300 baud
0C hex:	external clock (19.2K baud)
0F hex:	MCU could not detect baud rate

   0: Data With No Destination

   This response indicates a character with no destination has been detected. This occurs whenever a character is transmitted from the external device and both double buffering is not enabled or no read is pending.

   Bits 7-0: The actual character that caused the unsolicited interrupt.

TYPE 011: Reset Result

Note that if self-test fails, it may not be possible to get the result to the CPU. Reset Result sets the appropriate bit in status register 32. Two bits of the self-test result will be this port's ID. This is used by the diagnostics to verify the port ID lines.

Bit 10:
    0: Error Reset

    Bits 7-0:   contains the error reset error code:
    00 hex: Undefined
    01 hex: IS3- timeout in status word loop
    02 hex: IS3- timeout in control word loop
    03 hex: No read configuration during process character routine
    04 hex: Timeout during status word transfer of immediate transfer routine
    05 hex: No read configuration available after MCU->CPU transfer
    06 hex: No write configuration available after CPU->MCU transfer
    1X hex: Status word collision during MSENDSTATS routine (low byte contains first overrun status word type)
    2X hex: Status word collision during backspacing routine (low byte contains first overrun status word type)

    1: Hard Reset

    Bits 12-11: Port ID

    Bits 7-0: Self-Test Result:

    0: pass
    1: failed ROM checksum
    2: failed RAM address check
    3: failed RAM pattern test
    4: not in MCU operating mode 7

Note that a hard reset also indicates a powerfail on the card.

TYPE 100: Modem Information

Bits 9-8: Information type as defined below

    00: Incoming call (RI F-> T and RI masked=0) 01: Modem initiated disconnect (cnct=T DTR/RTS=T, DSR*CD*CTS T->F)
    1X: Modem connect (cnct=F DTR/RTS=T DSR*CD*CTS F -> T). It also sets cnct.

Bit 5: Value of Carrier Detect line
    0 = true (detected carrier)

Bit 4: Value of Ring Indicator line
    0 = true (ringing)

Bit 3: Value of Data Terminal Ready Request To Send lines
    0 = true (enable modem connect)

Bit 2: Value of Data Set Ready line
    0 = true (modem ready)

Bit 1: Value of Clear To Send line
   0 = true (modem receive ready)

Bit 0: Value of MCU Modem connection state
   0 = MCU acknowledges connection

TYPE 101: Backspace Information

This type of status word is sent when the user sends backspaces past the start of the current buffer, and the FIRST BUFFR bit is cleared. When the MCU detects the first excess backspace into the previous buffer, it echoes the backspaces (if appropriate) until it detects a non-backspace character. The MCU then sends that character, along with the number of backspaces detected. It also places this character in the start of MAINBUF, to either be used, or discarded by the next read request. Note that if more than 254 excess characters are entered, a backspace will cause a delete since the counter is overrun (on normal ASCII reads).

Bits 12-8: Number of Excess Backspaces
   If this number is 37 octal (all ones) it indicates an overflow of the counter, and that bits 7-0 are invalid. The number of backspaces is N-1 (that is, 0->1 backspace ... 31->32 backspaces).
Bits 7-0: Replacement Character
   This character is available in the event that the user backspaced an odd number of characters into the previous buffer. The driver then uses this character, and clears the START CHAR bit in the subsequent read request. It is the driver's responsibility to test for terminators, since they are not detected by the MCU in this case. This allows all subsequent DMA transfers to occur on even word boundaries.

TYPE 110: FIFO Buffering Data Available

Bits 7-0: character which initiated FIFO buffering interrupt.

TYPE 111: Error

Bits 10-8: Error type as defined below 0 hex: FIFO buffering buffer overflow. Character in bits 7-0.

0 hex:	FIFO buffer overflow. Character in bits 7-0.
1 hex:	SCI detected an overrun error. Data in bits 7-0.
2 hex:	SCI detected a framing error.
3 hex:	Undefined.
4 hex:	Undefined.
5 hex:	SCI detected a break.
6 hex:	Undefined.
7 hex:	No MCU status word available (PINT not set).

A framing error occurs when the 10th bit is not a stop bit (that is, not a mark). If this occurs twice in a 53.33 millisecond time frame, it is considered to be a break. A framing error usually indicates mismatched baud rates. Internally, the MCU will not report the break until it is removed, or up to four seconds. No data processing occurs during a break detection.

MUX Driver Description

This section explains how all mechanisms now supported by multiplexer driver ID400.REL are implemented by A400's MCU.

A400 MUX Driver Registers

The A400 4-channel on-board I/O (OBIO) system has three registers to make it accessible to the CPU. They are registers 30, 31, and 32.

Data Register 30

Data register 30 appears to the MCU as an 8-bit register. Therefore, whenever the CPU performs an "OTA 30B", only the 8 least significant bits are sent to the MCU. For this reason, the MCU will always wait for 2 DVCMD- states before it completely reads "16 bits" from data register 30. As an example: To send a 16-bit word using register 30B, the CPU first clears the SRQ- line (clf 30B), then sends the MCU the most significant byte (OTA 30B,C). It next asserts DVCMD- (STC 30B), and waits for the MCU to send a SRQ-, which sets flag 30B, indicating the MCU has received the first byte. After flag 30B has been set, the CPU clears flag 30B, and writes the second byte (using OTA 30B,C). Again the CPU asserts DVCMD- (STC 30B). The transfer is complete when the MCU again asserts SRQ- (which sets flag 30B).

DATA REGISTER 30

7/15	6/14	5/13	4/12	3/11	2/10	1/9	0/8
MSB							LSB
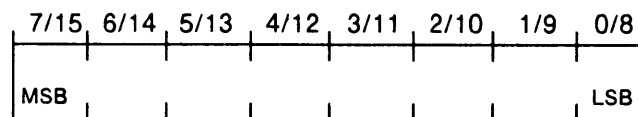
Control Register 31

Control register 31 allows the driver to control the ports. The driver writes to this register, instructing the designated MCU that the next word will be an MCU control word, MCU status word, or actual data.

CONTROL REGISTER 31

7	6	5	4	3	2	1	0
CNTRL XFER	PORT SELECT		MCU OUTPT	ENABLE PORT INTERRUPTS			
				D	C	B	A
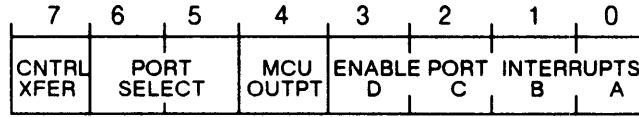
Bit 7: CNTRL XFER

Setting bit 7 indicates that the next word at data register 30 will be either an MCU control word or a MCU status word (depending upon the MCU OUTPT bit). If this bit is not set, actual data will be available at data register 30.

Bits 6-5: PORT SELECT

These two bits notify the MCU which port is the target of the transfer. The MCU compares this value with the hard-wired identification at MCU ports P42 and P43. If the values do not match, the control transfer is ignored, and data at data register 30 is ignored until a write to control register 31 matches the port's ID. The values are:

BIT 6	BIT 5	PORT SELECTED
0	0	A
0	1	B
1	0	C
1	1	D

Bit 4: MCU OUTPT–

This bit is also known by the CPU/MUX driver as INPUT XFER. Throughout this document, INPUT XFER will be referred to as MCU OUTPT. MCU OUTPT tells the MCU whether to expect input or output through data register 30. If the bit is set, the MCU considers the transfer to be an output transfer, MCU to CPU.

Bits 3-0: ENABLE PORT INTERRUPTS –

If the appropriate bit is set, a MCU status available interrupt request generated by the MCU can propagate through to the IRQ– line. This bit does not, however, affect the corresponding value at status register 32. Note that if performing programmatic I/O, if any port interrupts, it will set flag 30B, possibly adversely affecting instructions such as SFC 30B, etc.

Transfer Type Selection

Bits 7 and 4 (CNTRL XFER and MCU OUTPT) are used to describe which type of transfer will take place at data register 30. Bits 6 and 5 (Port Select) are used to indicate the destination port.

BIT 7	BIT 4	ACTION
0	0	actual data transfer, CPU -> MCU
0	1	actual data transfer, MCU -> CPU
1	0	MCU control word transfer, CPU -> MCU
1	1	MCU status word transfer, MCU -> CPU

Register 31 Decoder

The following table explains the action taken according to the lower eight bits in control register 31.

	CPU WRITE	CPU READ	MCU CTRLWD	MCU STATWD
PORT A	00X/01X	02X/03X	20X/21X	22X/23X
PORT B	04X/05X	06X/07X	24X/25X	26X/27X
PORT C	10X/11X	12X/13X	30X/31X	32X/33X
PORT D	14X/15X	16X/17X	34X/35X	36X/37X

Note that the four least significant bits tell which ports are enabled (i.e., 011 means CPU write to port A, ports A and D enabled).

Status Register 32

```
STATUS REGISTER 32
 7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| PORT SELF-TEST RESULT |  PORT INTERRUPTS  |
|  D    C    B    A   |  D    C    B    A   |
+---+---+---+---+---+---+---+---+
```
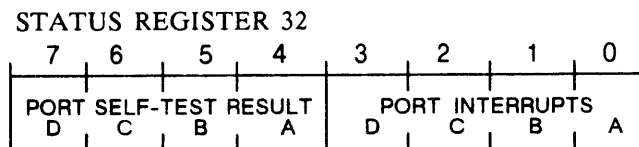
Bits 7-4: PORT SELF-TEST RESULT
These bits reflect the pass/fail result of the port's self-test. At each PON (power on) or programmatic HARD RESET, the MCU will initially set this bit, and clear it if the self-test passed. If the self-test fails, it may be possible the failure was not severe, and that the self-test result may be requested.

Bits 3-0: PORT INTERRUPTS
These status bits are cleared if the corresponding MCU is requesting interrupt service, and is independent of the value at control register 31.

Modem Control

The A400 supports six modem lines. They implement the necessary handshakes to allow modem communication. These lines are controlled by the Data Communications Equipment (DCE) and the Data Terminal Equipment (DTE). The MCU functions as the DTE.

To initiate a longer transfer than the MCU's memory will allow, read and write requests must be broken into several requests small enough for the MCU's buffer. Note that the same action may be taken by the CPU using a STC/SFC structure, but the DMA quad method is simpler and more code efficient.

Modem States

The modem is considered connected when the DSR line is "ON" (logical 0), the CTS line is "ON", and the CD line is "ON". When the CPU instructs the MCU to configure itself as a modem, it will turn "ON" the DTR line (logical 0). Thereafter, when the MCU is writing to a buffer, it checks the CTS line before transmitting. If this line is "OFF" (logical 1) the MCU will hold off transmitting until it becomes true. If the modem is instructed to perform a disconnect, it will turn "OFF" DSR (logical 1).

Modem CPU Interrupts

The MCU will only interrupt the CPU if it is configured as a modem by a CN30B command. The modem will interrupt (inform the CPU with a status word) upon connection or disconnection, if so instructed. If the last command was to connect, and it becomes disconnected, it will interrupt the CPU, or visa versa. The MCU will interrupt each time the CD line changes states, and will also interrupt each time the ring indicator line goes "ON".

RTS [Pin #4]/DTR [Pin #5]

These two lines are tied together in the hood connector. In this document, Request To Send and Data Terminal Ready are interchangeable terms. This line is generated by the DTE (MCU). This line is true when the MCU is connected, or is in the process of connection.

Clear To Send (CTS) Line [Pin #5]

This line is controlled by the DCE (modem). The DCE can temporarily restrain the MCU from sending data by lowering this line. The MCU tests this line before every character it transmits, to be sure the line is ready. It is analogous to an "XOFF" character.

Data Set Ready (DSR) Line [Pin #6]

This line is controlled by the DCE (modem). It indicates that the DCE is ready to communicate when set high to a logical 1.

Ring Indicator (RI) Line [Pin #22]

This line indicates an incoming call, and is controlled by the DCE (modem). The MCU will generate an interrupt if it is configured as a modem, and it detects this line going from "OFF" to "ON".

Carrier Detect (CD) Line [Pin #8]

This line is controlled by the DCE (modem). It indicates that a DCE is connected. This line and the DSR line must be "ON" for the MCU to transmit.

Firmware Architecture

CPU Interrupt Request Line

The MCU status available interrupt line (P40, PINT*), is configured as an output. Before the MCU sets this line, it must first save the status word to be retrieved when the driver requests the MCU status word.

Initialization

Upon a reset, PPA must be configured to a default value for VCP. In this manner, all ports will be preconfigured to the same initial value. These values are: Internal clocking of 9600 baud, and no XON/XOFF handshakes. If a carrier is detected, the port will be configured as a connected modem, otherwise it is assumed no modem is present. Unsolicited character interrupts are recognized, and all characters are echoed. The MCU performs a self-test, and upon completion, assert the MCU status available interrupt request line, with the SELF-TEST bit set.

Self-Test

The self-test will initially set the self-test failed bit, and clear it if the result passes. As much as possible, it will test the RAM, perform a CRC on the ROM, and check some ports. The result will be saved, and a MCU status available interrupt will be initiated.

Port Definitions

P10 – P17: Modem Control

The MCU modem control port 1 is used only on PPB and PPC. It monitors and controls all modem lines. These lines are connected as follows:

P17	P16	P15	P14	P13	P12	P11	P10
SFTST FAILD	WATCH RI	CD-	RI-	DTR/ RTS-	DSR-	CTS-	CNCT-

P10: MCU acknowledges modem connect Output [Port B and C only]

P11: Clear to Send Input (CTS) [Port B and C only]

P12: Data Set Ready Input (DSR) [Port B and C only]

P13: Data Terminal Ready Output (DTR) [Port B and C only]

P14: Ring Indicator Input (RI) [Port B and C only]

P15: Carrier Detect Input (CD) [Port B and C only]

P16: Watch Ring Indicator (enable RI interrupts)

P17: Last self-test executed failed Output

P20 - P24: SCI

Port lines P20-P22 configure the MCU mode upon power-up. They must all be set to "1"'s to allow the desired mode 7 configuration. P21 and P20 are pulled high by an external pull-up resistor. P22 is high upon PON. P25 – P27 give the mode the card is currently in. These lines control the Serial Communications Interface as follows:

P27	P26	P25	P24	P23	P22	P21	P20
	MCU OPERATING MODE		TDO	RDI	EXTNL CLOCK	XXXXX XXXXX	XXXXX XXXXX

P20: Logical "1" (tied high)
P21: Logical "1" (tied high)
P22: External Clock Input
P23: Receive Data Input
P24: Transmit Data Output
P25-P27: Current operating mode: all 1's for single
        chip mode

P30 - P37: MCU Data Port

P37	P36	P35	P34	P33	P32	P31	P30
BIT15 /BIT7	BIT14 /BIT6	BIT13 /BIT5	BIT12 /BIT4	BIT11 /BIT3	BIT10 /BIT2	BIT9 /BIT1	BIT8 /BIT0

This port is connected through external hardware to data register 30. P30 is the least significant bit of the data byte. Strobe lines OS3- and IS3- are used with this port for handshaking. These lines are always configured as inputs except if the port is enabled, when they can be either all input or all output.

P40 - P47: MCU Misc Port

P47	P46	P45	P44	P43	P42	P41	P40
PINT-	SLRQ-	PID1	PID0	PRT4 GREAD	PRT4 INAUX	MCU OUTPT	CNT XFR+

This port is used to watch the data from control register 31.

P40: MCU Control Transfer Input (CNTXFR+)
P41: MCU OUTPT, {CPU Input Transfer} (INXFR+)
P42: Data in AUX (FIFO) buffer (Output/Internal flag)
P43: Read buffer available (Output/Internal flag)
P45: MSB Port Identification Input
P46: Slave Request Input/Output, PPA only (SLRQ-)
P47: MCU STATUS AVAILABLE INTERRUPT REQUEST Output (PINT-)

Definition Of On-Board RAM

INTERNAL RAM MAP

LOCATION		NAME
$000BH ->	USER DEF. TERMINATR	<- (UNUSED OUTPUT
$000CH ->	EXCESS BACKSPACE	<- COMPARE REGIS-
	COUNT	<- TER, I.E. RAM)
$0040H ->	07 BYTE STACK	<- (STACKEND)
$0046H ->	(NO PROGRAM STACK)	<- (STACK)
$0047H ->	95 CHAR AUXILIARY	<- (SAUXBUF)
$00A5H ->	BUFFER (AUXBUF)	<- (EAUXBUF)
$00A6H ->	12 BYTE MISC.	<- (SMISC)
$00B1H ->	STORAGE AREA	<- (EMISC)
$00B2H ->	78 CHARACTER INPUT/	<- (SMAINBUF)
	OUTPUT BUFFER	
$00FFH ->	(MAINBUF)	<- (EMAINBUF)

AUXIN_PTR points to the location for the next incoming character from the SCI, if the buffer is not full. The buffer is full when PRT4_INAUX is set and AUXIN_PTR = AUXOUT_PTR. AUXOUT_PTR points to the location of the next character available for processing. The buffer is empty when PRT4_INAUX is cleared.

The Stack builds down, and contains enough memory for ONE interrupt service routine. Therefore, no routines may use the stack.

MISCELLANEOUS RAM DEFINITIONS

NAME	LOCATION	USE
PROTOCOLWD	$00A6H	PROTOCOL WORD
ENQCNTR	$00A7H	ENQ/ACK COUNTER & ENQ RETRY TIMER COUNTER
AUXIN_PTR	$00A8H	FIFO INPUT DATA POINTER (SEE ABOVE)
AUXOUT_PTR	$00A9H	FIFO OUTPUT DATA POINTER (SEE ABOVE)
MAINBF_PTR	$00AAH	POINTER TO 150 CHARACTER MAIN BUFFER
MISC_FLAGS	$00ABH	MCU INTERNAL FLAGS BYTE
CNTRL_WORD	$00ACH	MSB OF CONTROL WORD
CNTRLWDLSB	$00ADH	...AND LSB OF CONTROL WORD
STATWORD	$00AEH	MSB OF MCU STATUS WORD
STATWRDLSB	$00AFH	...AND LSB OF STATUS WORD
MCU_STATE	$00B0H	MCU ONE-TIME STATE INDICATOR FLAGS
MCU_SAVE	$00B1H	LOCATIONS TO BE SAVED ON SOFT RESET + SCI RETS

INTERNAL REGISTERS

NAME	LOCATION	USE
P1DDR	$0000H	PORT1 DATA DIRECTION REGISTER 1=OUTPUT
P3DDR	$0004H	PORT3 DATA DIRECTION REGISTER 1=OUTPUT
P4DDR	$0005H	PORT4 DATA DIRECTION REGISTER 1=OUTPUT
P1DATA	$0002H	PORT1 DATA REGISTER
P2DATA	$0003H	PORT2 DATA REGISTER
P3DATA	$0006H	PORT3 DATA REGISTER
P4DATA	$0007H	PORT4 DATA REGISTER
TCSR	$0008H	TIMER CONTROL AND STATUS REGISTER
C_R_MSB	$0009H	COUNTER REGISTER MSB
P3CSR	$000FH	PORT3 CONTROL AND STATUS REGISTER
RMCR	$0010H	SCI RATE AND MODE CONTROL REGISTER
TRCSR	$0011H	SCI TRANSMIT/RECEIVE CONTROL AND STATUS REGISTER
RDR	$0012H	SCI RECEIVE DATA REGISTER
TDR	$0013H	SCI TRANSMIT DATA REGISTER

(Note that location $0006H returns Revision code and location $0012 returns ROMFUDGE on dynamic state requests. See appropriate section)

DEFINITIONS OF PORT 1 DATA BITS

```
PORT1_FAIL    EQU 10000000B    BIT 7: SELF-TEST FAILED = 1
PORT1_WRI_    EQU 01000000B    BIT 6: WATCH RING INDICATOR (NEG. TRUE)
PORT1_CD_     EQU 00100000B    BIT 5: CARRIER DETECT (NEGATIVE TRUE)
PORT1_RI      EQU 00010000B    BIT 4: RING INDICATOR (NEGATIVE TRUE)
PORT1_DTR_    EQU 00001000B    BIT 3: DATA TERMINAL READY/RTS (NEG. TRUE)
PORT1_DSR_    EQU 00000100B    BIT 2: DATA SET READY (NEGATIVE TRUE)
PORT1_CTS_    EQU 00000010B    ;BIT 1: CLEAR-TO-SEND (NEGATIVE TRUE)
PORT1CNCT_    EQU 00000001B    ;BIT 0: MCU IN MODEM  CONNECT STATE (NEG. TRUE)
```

EXPLANATION OF MODEM STATE (INVALID IF IN HARDWIRE HANDSHAKE MODE!)

```
CNCT_ = 0 DTR_ = 0 :    NOT MODEM, NOT ACTIVE
CNCT_ = 0 DTR_ = 1 :    WAITING TO BE CONNECTED (CPU INITIATED)
CNCT_ = 1 DTR_ = 0 :    WAITING TO BE DISCONNECTED (CPU INITIATED)
CNCT_ = 1 DTR_ = 1 :    ACTIVE MODEM DEFINITIONS OF PROTOCOL WORD
```

DEFINITIONS OF PROTOCOL WORD

```
PROTO_VCP     EQU 10000000B BIT 7: IN VCP MODE
PROTO_SXOF    EQU 01000000B BIT 6: SENT AN XOFF NO ROOM TO RECEIVE)
PROTO_WACK    EQU 00100000B BIT 5: WAITING FOR ACK STATE
PROTO_RXO     EQU 00010000B BIT 4: RECEIVED AN XOFF (CAN'T TRANSMIT) BITS 3-0: PROTOCOL TYPE
PROTO_WIRE    EQU 00001000B BIT 3: HARDWIRE HAN SHAKE
PROTO_4IN     EQU 00000100B BIT 2: FOREIGN CPU
PROTO_HP      EQU 00000010B BIT 1: HP PROTOCOL (ENQ/ACK & DC1)
PROTO_X       EQU 00000001B BIT 0: XON/XOFF PROTOCOL (IF BIT 1 SET: RXOFF ONLY ) DEFINITIONS OF
                                   MCU_STATE WORD
```

DEFINITIONS OF MCU_STATE WORD

```
STATE_DC1     EQU 10000000B    BIT 7: SEND AN ASCII "DC1 (XON)"
STATE_CR      EQU 01000000B    BIT 6: SEND AN ASCII "CR"
STATE_LF      EQU 00100000B    BIT 5: SEND AN ASCII "LF"
STATE_XOFF    EQU 00010000B    BIT 4: SEND AN ASCII "XOFF"
STATE_SENS    EQU 00001000B    BIT 3: DO SPEED SENSING
STATE_BRAK    EQU 00000100B    BIT 2: GENERATE A BREAK
STATE_ENAK    EQU 00000010B    BIT 1: SEND AN ASCII "ENQ", WAIT FOR ACK
STATE_FRRD    EQU 00000001B    BIT 0: FORCE READ COMPLETE
```

DEFINITIONS OF MCU_SAVE WORD (SOME BITS SAVED ON A SOFT RESET)

```
SAVE_TYPEA    EQU 10000000B    BIT 7: IN TYPE-AHEAD MODE (RETURN TO DBL BUFR)
SAVE_ICHAR    EQU 01000000B    BIT 6: INTERRUPT ON  PER-CHAR IN TYPE AHEAD
SAVE_RPNDG    EQU 00100000B    BIT 5: ALLOW DOUBLE BUFFERING, READ IS PENDING
SAVE_USRDF    EQU 00010000B    BIT 4: VALID USER DEFINED TERMINATOR
SAVE_FREE1    EQU 00001000B    BIT 3: FREE
SAVE_FREE2    EQU 00000100B    BIT 2: FREE
SAVE_SRETW    EQU 00000010B    BIT 1: SCI RETURN TO DMA WRITE LOOP
SAVE_SRETR    EQU 00000001B    BIT 0: SCI RETURN TO DMA READ LOOP
```

DEFINITIONS OF MISC_FLAGS WORD

```
MISC_GREAD   EQU 10000000B   BIT 7: FLAG HAVE A FULL READ BUFFER
MISC_SNDBF   EQU 01000000B   BIT 6: SEND DATA IN  MAIN BUFFER TO EXT. DEVICE
MISC_WDONE   EQU 00100000B   BIT 5: SEND WRITE  BUFFER EMPTY STATUS  WHEN ALL DONE
MISC_DELBF   EQU 00010000B   BIT 4: TELL CPU TO DELETE THE PREVIOUS  DATA
MISC_PBRAK   EQU 00001000B   BIT 3: POSSIBLE BREAK ...TEST
MISC_XS_BS   EQU 00000100B   BIT 2: WE HAVE EXCESS BACKSPACES
MISC_OLDRI   EQU 00000010B   BIT 1: PREVIOUS VALUE OF RING INDICATOR (0:TRUE)
MISC_INAUX   EQU 00000001B   BIT 0: DATA IS AVAILABLE IN AUXILIARY BUFFER
```

WHAT TO DO WHEN DATA IS AVAILABLE (MISC_INAUX = 1):

STATE	SAVE_TYPEA	SAVE_RPNDG	MISC_GREAD
DEST. DATA	0	0	0
SAVE/T.A.	1	0	X
SAVE/BETWEEN READS	0	1	1
PROCESSING/RET T.A.	1	1	0
PROCESS./NO RET T.A.	0	1	0

8

Input/Output (I/O) System

The purpose of the input/output system is to transfer data between the computer and external devices. As shown in Figure 8-1, data can be transferred either by a direct memory access (DMA) feature or through the A- or B-Register in the CPU (non-DMA). Each A/L-Series I/O interface has DMA logic and DMA is normally used for most I/O data transfers. Once the DMA logic has been initialized, no programming is involved and the transfer occurs in two distinct steps as follows:

1. Between the external device and its I/O interface card in the computer;

2. Between the I/O interface and memory via the backplane data bus.

This two-step process also applies to a DMA output transfer except in reverse order.

As mentioned above, data may be transferred under program control without using the DMA feature. This type of transfer allows the computer to manipulate the data during the transfer process. A non-DMA input transfer is a three-step process as follows:

1. Between the external device and its I/O interface;

2. Between the I/O interface and the A- or B-Register via the data bus and the processor; and

3. Between the A- or B-Register and memory via the processor and the data bus.

Note that in the DMA transfer the processor is bypassed. Since a DMA transfer eliminates programmed loading and storing via the accumulators, the time involved is very short. The DMA feature is discussed in more detail later in this chapter.

Input/Output Addressing

As shown in Figure 8-2, an external device is connected by cable directly to the break-out panel of the A400 four-channel MUX or to an interface card located in the computer card cage. The interface card, in turn, plugs into one of the input/output backplane slots, each of which is assigned a fixed interrupt priority. Note, however, that the select code of the A/L-Series interface cards is independent of the priority. The computer communicates with a specific device on the basis of its select code, which is set by switches on the interface card. The A400 on-board I/O is hard-wired to select code 77 octal and cannot be changed.

The on-board I/O always has the highest priority. The interface card inserted next to the A400 board has the second highest priority as shown in Figure 8-2. The priority of a device can be changed by exchanging its interface card and cable with those occupying some other I/O slot. This will change the priority but not the I/O address (select code). Due to priority chaining, there can be no vacant slots from the on-board I/O to the lowest priority slot used. Only select codes 20 through 76 (octal) are available for input/output cards; the lower select codes (00 through 17) are reserved for other features.
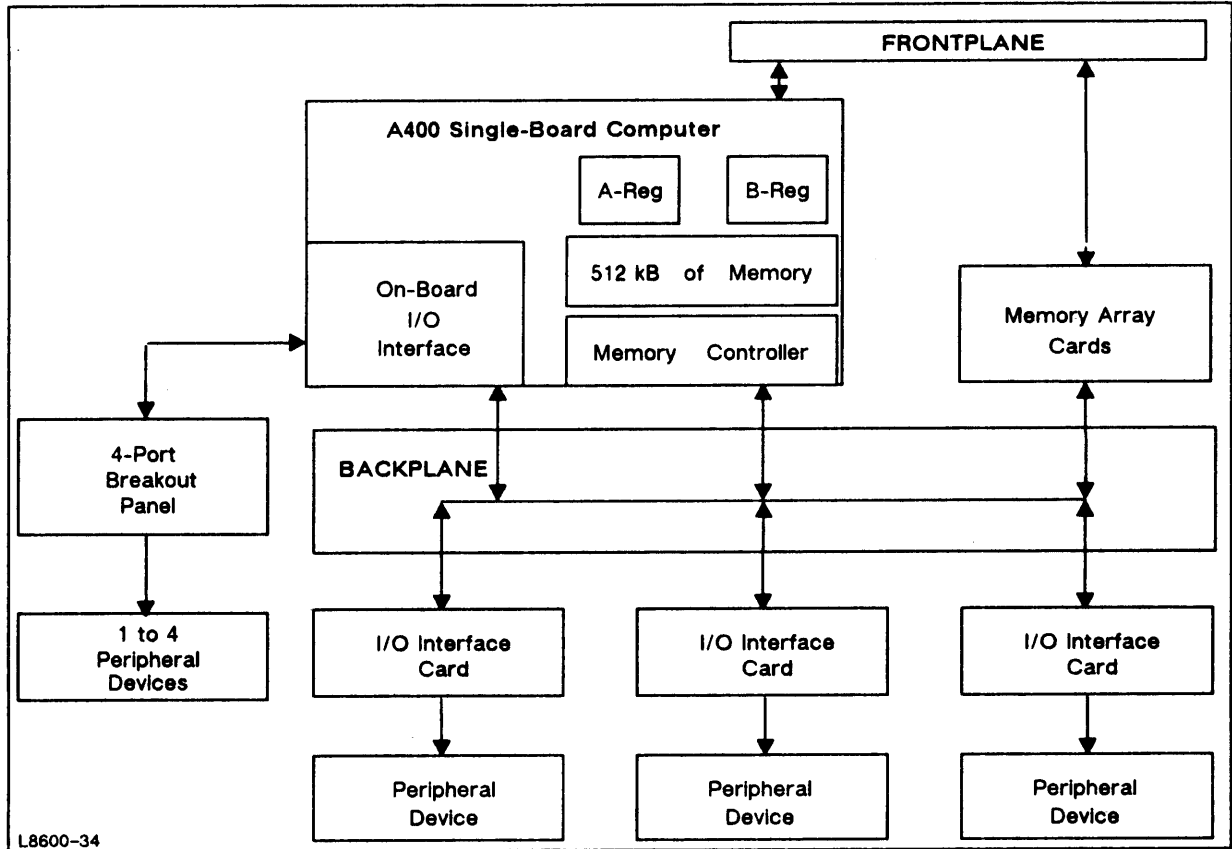
Figure 8-1. Input/Output System

Without Memory Array

Slot 1

Slot 6

With Memory Array

Memory
Frontplane
Connector

Slot 1

Slot 6

A400 Computer

To 4-Channel
Breakout Panel
(highest priority I/O)

To I/O Device with
Second Highest Priority
(System Disc Drive)

Memory Array
Card

A400 Computer

To 4-Channel
Breakout Panel
(highest priority I/O)

To I/O Device with
Second Highest Priority
(System Disc Drive)
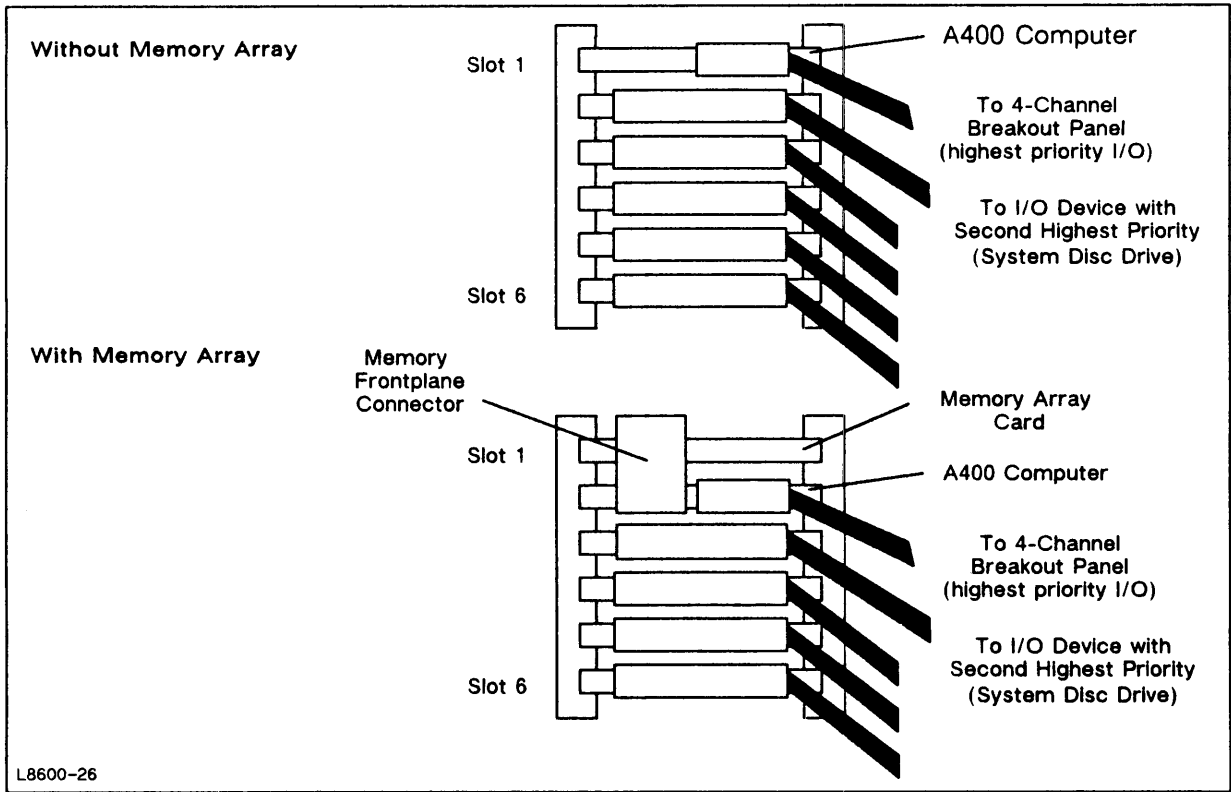
L8600-26

Figure 8-2. I/O Priority Assignments
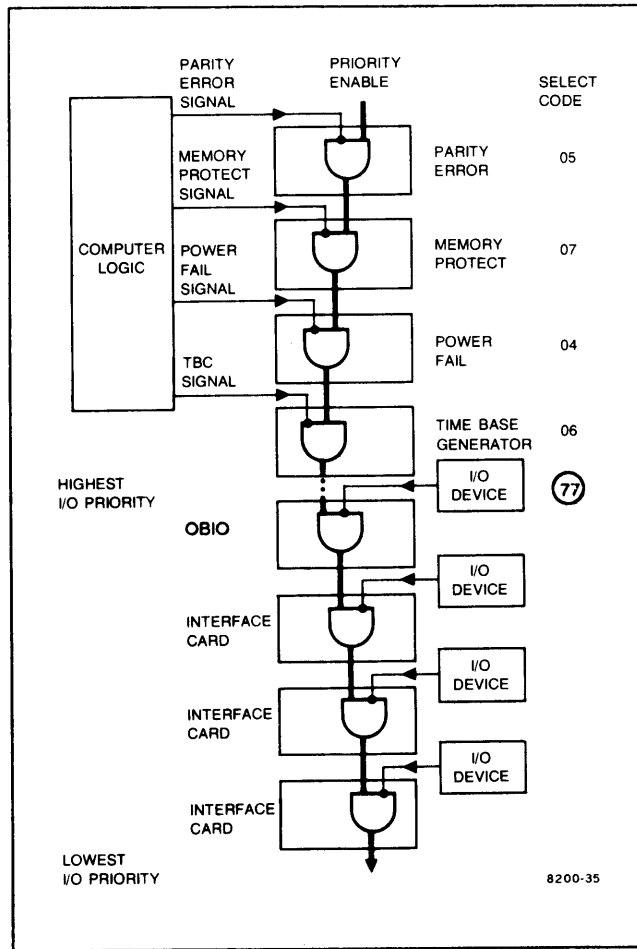
Figure 8-3. Priority Linkage (Simplified)
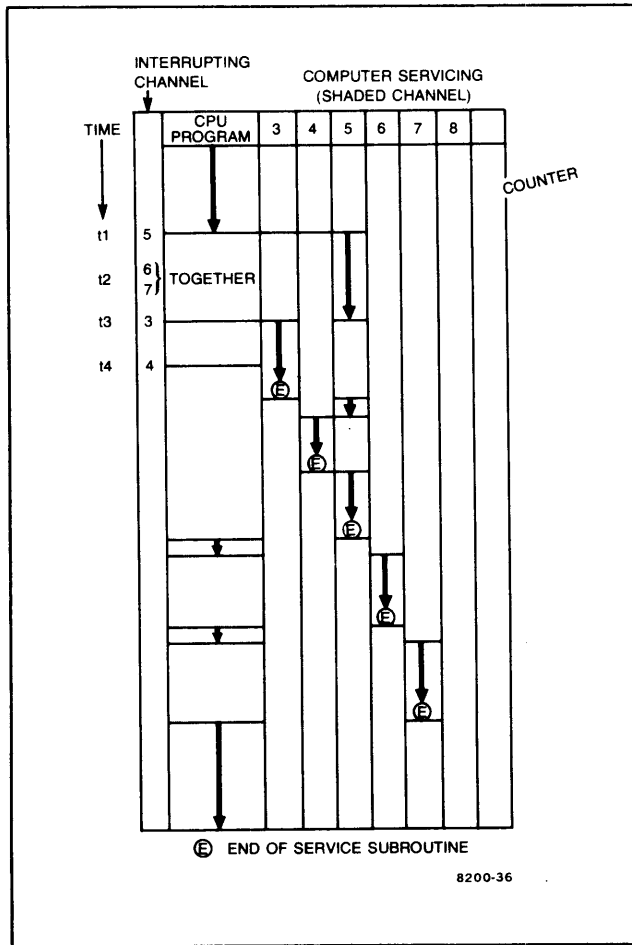
Figure 8-4. Interrupt Sequence

Input/Output Priority

The plug-in card slots of the A400 computers are numbered 1 through 6 for the Micro 14, 1 through 14 for the Micro/1000, and 1 through 20 for the 2134A. If additional memory is desired, slots above the A400 board are used. Slots below the A400 board are available for I/O cards, with the on-board I/O having the highest I/O interrupt priority and the slot below the A400 board having the second highest priority. An I/O channel consists of an I/O device (or devices) and its I/O interface and is assigned the number of the card slot.

When an input/output device is ready to be serviced, it causes its interface card to request an interrupt so that the computer will interrupt the current program and service the device. Since many device interfaces will be requesting service at random times, it is necessary to establish an orderly sequence for granting interrupts. Also, it is desirable that high-speed devices should not have to wait for low-speed device transfers. Both of these requirements are met by a series-linked priority structure illustrated by Figure 8-3. The bold line, representing a priority enabling signal, is routed in series through each interface capable of causing an interrupt. The interface cannot interrupt unless this enabling signal is present at its input.

Each device (or other interrupt function) can break the enabling line when it requests an interrupt. If two devices simultaneously request an interrupt, the device with the highest priority will be the first one that can interrupt because it has broken the enable line for the lower-priority device. The other device cannot begin its service routine until the first device is finished. However, a still higher-priority device (one interfaced through a lower-numbered slot) may interrupt the service routine of the first device. Figure 8-4 illustrates a hypothetical case in which several devices request service by interrupting a CPU program. Both simultaneous and time-separated interrupt requests are considered.

Assume that the computer is running a CPU program when an interrupt from I/O channel 5 occurs (at reference time t1), and that the card in slot 5 (channel 5) is assigned select code 22. With the I/O interface supplying the select code as the memory address, a JSB instruction in the interrupt location for select code 22 causes a program jump to the service routine for the channel-5 device (select code 22). The JSB instruction automatically saves the return address (in a location which you must reserve in your routine) for a later return to the CPU program.

The routine for channel 5 (select code 22) is still in progress when several other devices request service (set flag). First, channels 6 and 7 request simultaneously at time t2; however, since neither one has priority over channel 5, their flags are ignored and channel 5 continues its transfer. But at t3, a higher priority device on channel 3 requests service. This request interrupts the channel 5 transfer and causes the channel 3 transfer to begin. The JSB instruction saves the return address for return to the channel 5 routine.

During the channel 3 transfer, the channel 4 flag is set (t4). Since it has lower priority than channel 3, channel 4 must wait until the end of the channel 3 routine. And since the channel 3 routine, when it ends, contains a return address to the channel 5 routine, program control temporarily returns to channel 5 (even though the waiting channel 4 has higher priority). The JMP,I instruction used for the return inhibits all interrupts until fully executed. At the end of this short interval, the channel 4 interrupt request is granted.

When channel 4 has finished its routine, control is returned to channel 5, which at last has sufficient priority to complete its routine. Since channel 5 has been saving a return address in the main CPU program, it returns control to this point.

The two waiting interrupt requests from channels 6 and 7 are now enabled. Channel 6 has the higher priority and goes first. At the end of the channel 6 routine, control is temporarily returned to the CPU program. Then the lowest priority channel (channel 7) interrupts and completes its transfer. Finally, control is returned to the CPU program, which resumes processing.

Interface Elements

The I/O interface provides the communication link between the computer and one or more external devices. The interface includes several basic elements which either the computer or the device can control in order to effect the necessary communication. These basic elements are the Global Register, control bits, flag bits, data buffer register, and control register. Other registers, associated only with DMA, are discussed in the "Direct Memory Access" section of this chapter. The control and flag bits and the data buffer and control registers of an interface can be addressed directly when its select code is in the Global Register (GR) and the GR is enabled. Refer to the interface card reference manuals or the "On-Board I/O" chapter of this manual, for specific information on the data and control registers.

Global Register

In the A-Series computers, the select code that is in the Global Register specifies which I/O interface is enabled to execute I/O instructions. The Global Register (GR) is a register on each I/O interface that can be loaded with the select code of any one of the I/O interfaces. (At any given time, the GR on all I/O interfaces is loaded with the same select code.) When the GR is enabled, an I/O instruction is executed only by the I/O interface whose select code matches the select code in its GR. Also, the GR allows other registers on the selected I/O interface to be accessed programmatically by I/O instructions. The Global Register on all I/O interfaces may be simultaneously loaded with an OTA/B 02 instruction, enabled with a CLF 02 instruction, and disabled with an STF 02 instruction.

Control Bits

The control bits on an interface are used to turn on a specific I/O function. In addition, a control bit must be set to allow the corresponding flag bit to interrupt. There are three control bits associated with each I/O select code: control 20, 21, and 30. Control 30 is the only control bit that can be accessed with or without the Global Register being enabled. When control 30 is set it generates an action command, allowing one word or character to be read or written. Control 20 and 21 can only be accessed when the Global Register is enabled. When control 20 is set it turns on DMA self-configuration. The setting of control 21 enables DMA transfers.

Flag Bits

The flag bits (when set) are used primarily to interrupt or to signal completion of a task. Flag 30, the only flag bit accessible without using the Global Register, signals either that one data element has been transferred or that an interrupting condition has been detected. There are three other flags, all of which must be accessed with the Global Register enabled. Flag 20 signals DMA self-configuring transfer complete; flag 21 signals DMA transfer complete; and flag 22 signals parity error during DMA. The device cannot clear the flag bit. If the corresponding control bit is set, priority is high, and the interrupt system is enabled, then setting the flag bit will cause an interrupt to the location corresponding to the I/O interface's select code.

Data Buffer Register

The data buffer register (designated Register 30) is used for the intermediate storage of data during an I/O transfer. Typically, the data capacity is 16 bits.

Control Register

The control register (designated Register 31) enables a general purpose interface card to be configured for compatibility with a specific I/O device or to be programmed for particular modes of operation. The control register must be programmatically set up for each particular application. Refer to the interface card manuals for specific information on the control register.

Direct Memory Access

The direct memory access (DMA) capability of each interface provides a direct data path between memory and a peripheral device, making it practical to use DMA for most data transfers. The use of DMA to perform I/O data transfers reduces the number of interrupts from one per byte or word to one per complete DMA block transfer. (Maximum DMA block size is 65,536 bytes.)

The maximum DMA transfer rate is 4.4 million bytes per second; this also is the combined limit for DMA transfers by two or more I/O interfaces. Except when the DMA feature is operating at full bandwidth, the central processor can interleave memory cycles with the DMA operation. The DMA feature is provided by the following elements:

1. The common backplane that links the processor, memory, and I/O interfaces;

2. The capability of the I/O interfaces to execute I/O instructions; and

3. The Global Register which:

a. Enables only the I/O interface whose select code is in the Global Register to execute I/O instructions, freeing the address bits of the I/O instruction; and

b. Enables the I/O-instruction address bits to be used to access registers on the I/O interface specified by the Global Register.

Each I/O interface has four registers associated with DMA. Three of them must be loaded with control words that specify the DMA operation. The fourth register is used for a special type of DMA operation called self-configured DMA, which is discussed later. All of these registers can be accessed only when the select code of the desired I/O interface is in the Global Register. The DMA registers and their functions are as follows:

* Register 20, DMA Self-Configuration Address Register;

* Register 21 (for Control Word 1), DMA Control Register;

* Register 22 (for Control Word 2), DMA Address Register; and

* Register 23 (for Control Word 3), Word/Byte Count Register.

Control Word 1

Control Word 1 (CW1) must be loaded into Register 21 of the desired I/O interface as part of the DMA initialization process. The general definitions of the bits in Control Word 1 are given in Figure 8-5. Note that the requirements of individual I/O interfaces may vary slightly from the general definitions and that it is necessary to refer to the I/O interface reference manuals or the "On-Board I/O" chapter of this manual for specific programming information.

Control Word 2

Control Word 2 (CW2) loads into Register 22 the address of the first memory location to be read from or stored into when the DMA operation is initiated. The most significant bit (bit 15) is not used by the DMA control logic; when CW2 is read for status, bit 15 is the complement of bit 7 in CW1 (Figure 8-5).

Control Word 3

Control Word 3 (CW3) loads into Register 23 the two's-complement number of data elements to be transferred by DMA. Data elements may be either words or bytes as specified by bit 13 of CW1 (Figure 8-5). The end of a DMA data transfer is indicated by the transition from −1 to 0 of the value in Register 23 (the Word/Byte Count Register); this causes the I/O interface to generate a completion interrupt. (A DMA transfer can also be terminated in other ways as described in the interface card manuals.)

15	14	13	12	11	10	9	8	7	6 5	4 0
CONT	DVCMD	BYTE	RES	CINT	REM	FOUR	AUTO	IN	Various	ADDR EXT BUS

CONT (Continue), bit 15.

   Bit 15 = 1: Enable a DMA re-configuration upon completion of a self-configured DMA transfer.

   Bit 15 = 0: Stop DMA after current transfer.

DVCMD (Device Command), bit 14.

   Bit 14 = 1: Issue a Device Command signal for each data element transferred.

   Bit 14 = 0: No Device Command signal issued.

BYTE (Byte/word transfer), bit 13.

   Bit 13 = 1: Conduct DMA transfer in byte mode.

   Bit 13 = 0: Conduct DMA transfer in word mode.

RES (Residue), bit 12.

   Bit 12 = 1: Write word/byte count back into memory.

   Bit 12 = 0: Word/byte count is not written.

CINT (Completion Interrupt), bit 11.

   Bit 11 = 1: Inhibit DMA completion interrupt.

   Bit 11 = 0: Request completion interrupt when word/byte count goes from -1 to 0 and bit 15 equals 0.

REM (Remote), bit 10.

   Bit 10 = 1: Enable remote (non-standard) memory for DMA transfer.

   Bit 10 = 0: Remote memory not enabled.

FOUR (Fetch four control words), bit 9.

   Bit 9 = 1: Causes DMA self-configuration to fetch four control words; i.e., three DMA control words and one I/O card control word.

   Bit 9 = 0: Fetch three control words for DMA self-configuration.

AUTO (Automatic), bit 8. This bit is read only during self-configured DMA.

   Bit 8 = 1: Initiate first data transfer once DMA is configured to output, without waiting for an SRQ. For input transfers, enable a Device Command signal after the last data element is transferred.

   Bit 8 = 0: For output transfers, wait for a Service Request (SRQ) signal before performing the first transfer. For input transfers, the last data element is not followed by a Device Command.

IN (Transfer In), bit 7.

   Bit 7 = 1: Perform DMA transfer from I/O device to memory.

   Bit 7 = 0: Perform DMA transfer from memory to I/O device.

Various, bits 5 and 6, User definable.

ADDR EXT BUS, bits 4-0

   These five bits allow DMA accesses to physical memory by referencing one map set of 32 registers each.

8200-53

Figure 8-5. General Bit Definitions for Control Word 1

DMA Transfer Initialization

A DMA data transfer is started by:

1. Loading the Global Register with the select code of the desired I/O interface;

2. Loading the three DMA registers: DMA control into Register 21, DMA address into Register 22, and word/byte count into Register 23;

3. Loading the control register (Register 31) of the I/O interface (described in the individual interface card reference manuals or the "On-Board I/O" chapter); and

4. Issuing an STC instruction to Register 21 (DMA Control Register).

A typical programming sequence to configure the DMA logic for a DMA transfer is as follows:

```
LDA  SC           Load select code
OTA  2,C          Set up Global Register
CLC  21B
LDA  CW1
OTA  21B          Output DMA control word
LDA  CW2
OTA  22B          Output DMA starting address
LDA  CW3
OTA  23B          Output DMA word/byte count
LDA  CNTL
OTA  31B          Output I/O interface control word
STC  21B,C        Start DMA and device
<continue any other processing>
```

Self-Configured DMA

Each I/O interface also has logic that can automatically load the DMA registers discussed previously with the DMA control words from sequential locations in memory. This process is performed by using the I/O interface's Register 20, the Self-Configuration Register. The DMA self-configuration feature is initialized by setting the value of Register 20 to the memory address of a list of DMA "triplets" or "quadruplets".

A triplet is of the form: DMA control word, DMA transfer address, and word/byte count. The triplet words are the words to be loaded into Registers 21, 22, and 23, respectively. A quadruplet is of the form: DMA control word, I/O interface control word, transfer address, and word/byte count. Bit 8 of the DMA control word (Control Word 1) determines whether a triplet or quadruplet is loaded. (A quadruplet is used only when the I/O interface control word must be changed; refer to the interface card manuals or the "On-Board I/O" chapter for detailed information.) As each register is loaded, the contents of Register 20 are incremented, leaving it pointing to the memory location to be loaded into the next register.

DMA self-configuration can be chained to enable consecutive DMA transfers via the same I/O interface with a minimum of interrupts. If bit 15 of Control Word 1 in a triplet (or quadruplet) is a logic 1, the DMA registers will be loaded with the next triplet or quadruplet in memory (as pointed to by Register 20) upon completion of the current DMA block transfer. When bit 15 (and bit 11) is a logic 0, the current DMA block transfer is followed by a completion interrupt.

DMA Data Transfer

Figure 8-6 illustrates, in general, the sequence of operations for a DMA input data transfer (the minor differences for an output transfer are explained in text). Note that the Global Register has been enabled and loaded with the I/O interface's select code.

The initialization routine sets up the DMA control registers on the I/O interface (1) and issues the start command (STC 21,C) to the DMA Control Bit (Control 21). (If the operation is an output, the I/O interface buffer is also loaded at this time.) The DMA logic is now turned on and the computer program continues with other instructions.

Setting the DMA Control bit (2) causes the I/O interface to send a Start signal (with a data word if it is an output transfer) to the external device (3). The device goes through a read or write cycle and returns a Done signal (with a data word if it is an input transfer). The Done signal (4) requests the DMA logic (5) to transfer a word into (or out of) memory (6). The process now loops back to step 3 to transfer the next word.

After the specified number of words has been transferred, the DMA logic generates a completion interrupt (7). The program control is now forced to a completion routine (8), the content of which is the programmer's responsibility.

For more detailed information on DMA, refer to the *A/L-Series I/O Interfacing Guide*, part no. 02103-90005.
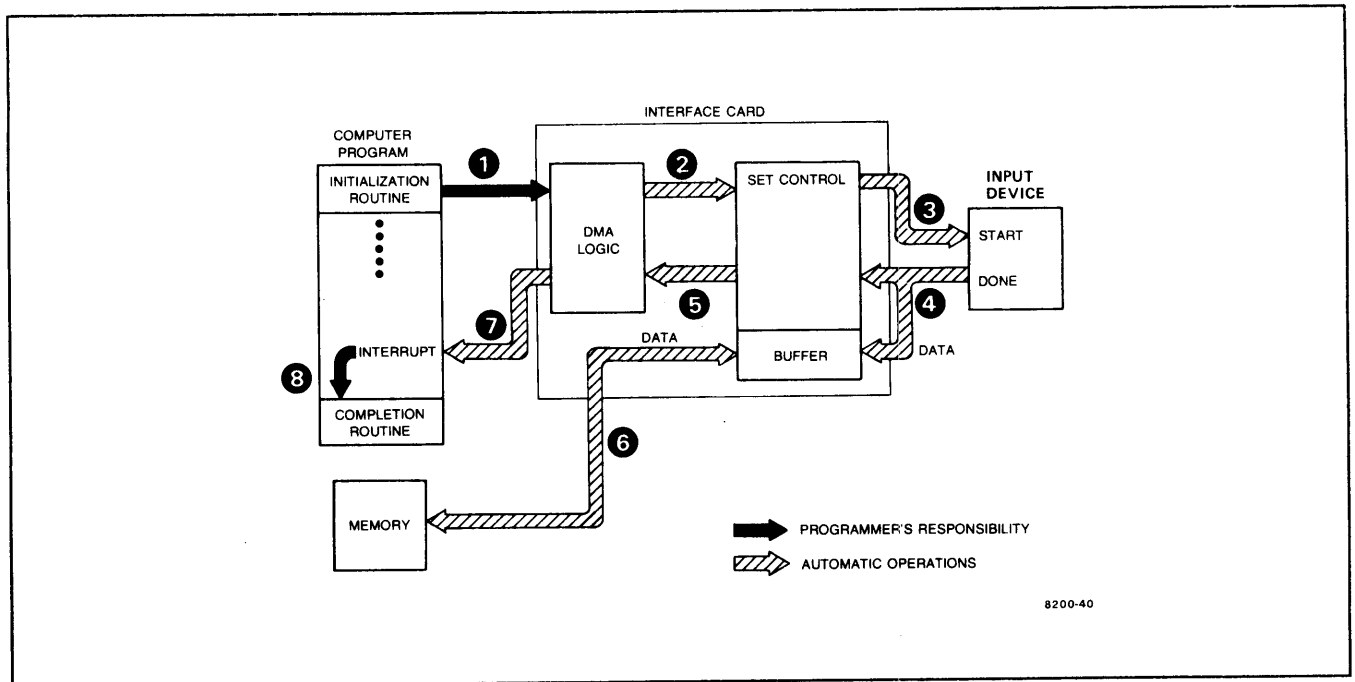
Figure 8-6. DMA Input Data Transfers

Non-DMA Data Transfer

The following paragraphs describe how data is transferred between memory and input/output devices without using DMA. The sequences presented are simplified in order to present an overall view without the involvement of software operating systems or device drivers.

Input Data Transfer (Interrupt Method)

Figure 8-7 illustrates the sequence of events required to input data using the interrupt method. Note that some operations are under control of the computer program (programmer's responsibility) and some of the operations are automatic. Note also that the Global Register has been loaded and enabled and the I/O interface's control register has been loaded.

The operations begin (1) with the programmed instruction STC 30,C which sets the Control bit (Control 30) and clears the Flag bit (Flag 30) on the I/O interface. Because the next few operations are under control of the hardware, the computer program may continue the execution of other instructions. Setting the Control bit causes the card to output a Start signal (2) to the device, which reads out a data character and asserts the Done signal (3).

The device Done signal sets the Flag bit, which in turn generates an interrupt (4) provided that the interrupt conditions are met; i.e., the interrupt system must be on (STF 00 previously given), no higher priority interrupt is pending, and the Control bit is set (done in step 1). The interrupt causes the current computer program to be suspended and control is transferred to a service subroutine (5). It is the programmer's responsibility to provide the linkage between the interrupt location (which agrees with the select code) and the service subroutine. It is also the programmer's responsibility to include in his service subroutine the instructions for processing the data (loading into an accumulator, manipulating if necessary, and storing into memory).

The subroutine may then issue further STC 30,C instructions to transfer additional data characters. One of the final instructions in the service subroutine must be CLC 30,C. This step (6) restores the interrupt capability to lower priority devices and returns the I/O interface to its static "reset" condition (Control clear and Flag clear). This condition is initially established by the computer at power turn-on and it is the programmer's responsibility to return the I/O interface to the same condition on the completion of each data transfer operation. At the end of the subroutine, control is returned to the interrupted program via previously established linkages.
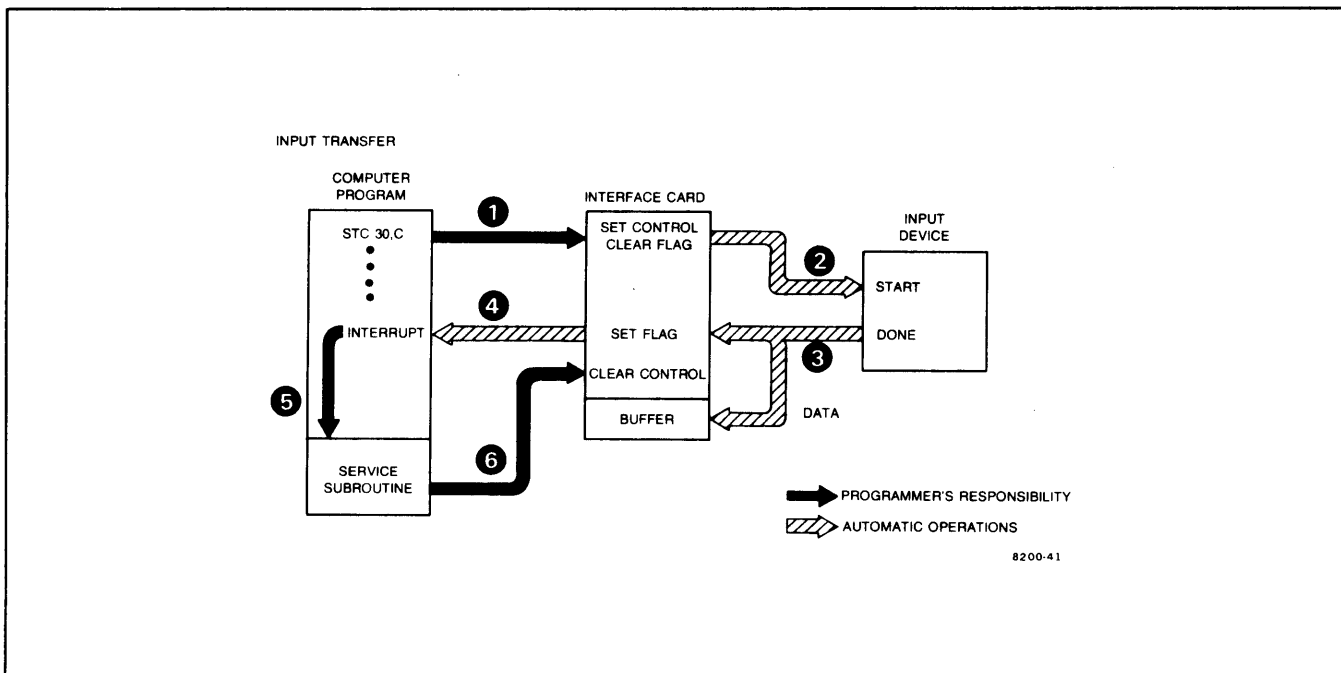
Figure 8-7. Input Data Transfer (Interrupt Method)

Output Data Transfer (Interrupt Method)

Figure 8-8 illustrates the sequence of events required to output data using the interrupt method. Again note the distinction between programmed and automatic operations. Note also that the Global Register has been loaded and enabled and that the I/O interface's control register has been loaded. It is assumed that the data to be transferred has been loaded into the A-Register and is in a form suitable for output.

The output operation begins with a programmed instruction (OTA 30) to transfer the contents of the A-Register to the I/O interface buffer (1). This is followed (2) by the instruction STC 30,C which sets the Control bit (Control 30) and clears the Flag bit (Flag 30) on the I/O interface. Because the next few operations are under control of the hardware, the computer program may continue the execution of other instructions. Setting the Control bit causes the interface to output the buffered data and a Start signal (3) to the device, which writes (for example, records and stores) the data character and asserts the Done signal (4).

The device Done signal sets the card's Flag bit, which in turn generates an interrupt (5) provided that the interrupt system is on, priority is high, and the Control bit is set (done in step 2). The interrupt causes the current computer program to be suspended and control is transferred to a service subroutine (6). It is the programmer's responsibility to provide the linkage between the interrupt location (which agrees with the select code) and the service subroutine. The detailed contents of the subroutine are also the programmer's responsibility and the contents will vary with the type of device.

The subroutine may then output further data to the I/O interface and reissue the STC 30,C command for additional data character transfers. One of the final instructions in the service subroutine must be a clear control (CLC 30,C). This step (7) allows lower priority devices to interrupt and restores the I/O interface to its static "reset" condition (Control clear and Flag clear). At the end of the subroutine, control is returned to the interrupted program via the previously established linkages.

OUTPUT TRANSFER

COMPUTER
PROGRAM

INTERFACE CARD

OTA 30
STC 30,C

BUFFER

DATA

OUTPUT
DEVICE

SET CONTROL
CLEAR FLAG

START

INTERRUPT

SET FLAG

DONE

CLEAR CONTROL

SERVICE
SUBROUTINE

PROGRAMMER'S RESPONSIBILITY

AUTOMATIC OPERATIONS

8200-42
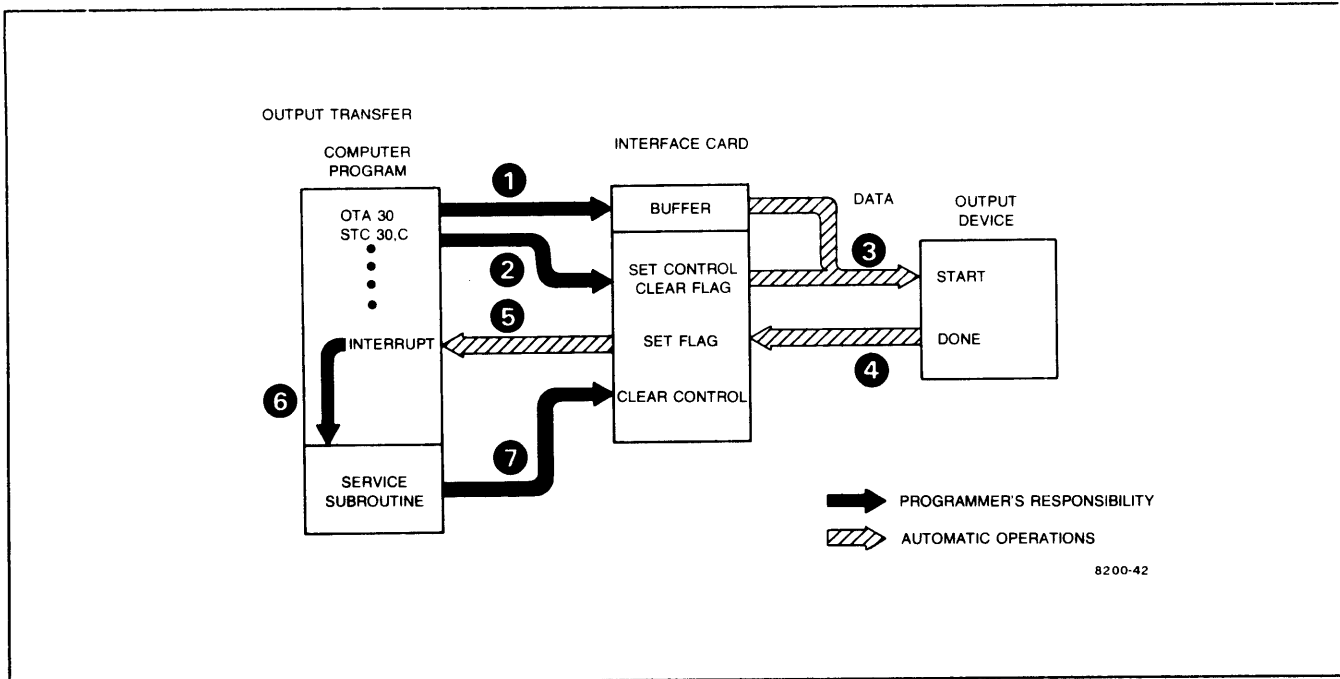
Figure 8-8. Output Data Transfer (Interrupt Method)

Non-Interrupt Data Transfer

It is also possible to transfer data without using the interrupt system. This involves a "wait-for-flag" method in which the computer commands the device to operate and then waits for the completion response. In using this method to transfer data, computer time is relatively unimportant. It is assumed that the interrupt system is turned off (STF 00 not previously given). It is also assumed that the Global Register has been loaded and enabled and that the I/O interface's control register has been loaded.

As shown in Table 8-1, the programming is very simple; each of the routines will transfer one word or character of data.

INPUT. As described in the "Input Data Transfer (Interrupt Method)" paragraph, an STC 30,C instruction begins the operation by commanding the device to read one word or character. The computer then goes into a waiting loop, repeatedly checking the status of the Flag bit (Flag 30). If the Flag bit is not set, the JMP *-1 instruction causes a jump back to the SFS instruction. (The *-1 operand is assembler notation for "this location minus one.") When the Flag bit is set, the skip condition for SFS is met and the JMP instruction is skipped. The computer thus exits from the waiting loop and the LIA 30 instruction loads the device input data into the A-Register.

OUTPUT. The first step, which transfers the data to the I/O interface buffer, is the OTA 30 instruction. Then STC 30,C commands the device to operate and accept the data. The computer then goes into a waiting loop as described in the preceding paragraph. When the Flag bit becomes set, indicating that the device has accepted the output data, the computer exits from the loop. (The final NOP is for illustration purposes only.)

Table 8-1. Non-Interrupt Transfer Rates

Instructions	Comments
Input Routine	
STC 30,C	Start device
SFS 30	Is input ready?
JMP *-1	No, repeat previous instruction
LIA 30	Yes, load input into A-Register
Output Routine	
QTB 30	Output data to I/O card's data register
STC 30,C	Start device
SFS 30	Has device accepted the data?
JMP *-1	No, repeat previous instruction
NOP	Yes, proceed

Diagnose Modes

A diagnose mode allows the I/O interfaces to be accessed for diagnostic or test purposes. A diagnose mode is established when an OTA/B 2 instruction (output to the Global Register) is executed with the A- or B-Register value equal to one through seven. (The diagnose mode is terminated when an OTA/B 2 instruction is executed with the A- or B-Register equal to zero.) When establishing a diagnose mode the current contents of the Global Register (GR) are not altered.

The diagnose mode can be on an individual I/O interface or on all I/O interfaces. If the GR is disabled then all I/O interfaces accept the diagnose mode. If the GR is enabled, only the I/O interface whose select code is in the GR will accept the diagnose mode. Diagnose Mode 7 is used to disable any service request (SRQ) signal coming into the I/O chip which may cause DMA to cycle during a test. (Mode 7 can be disabled only by a CRS signal (CLC 0).) Diagnose Modes 4 through 6 are reserved for future definition. Diagnose Modes 1 through 3 are described in the following paragraphs.

Diagnose Mode 1

When an OTA/B 2 instruction is executed with the A- or B-Register equal to 1, each I/O interface responds by turning off priority to the next I/O interface. When the instruction is complete the only I/O interface receiving priority will be the highest priority I/O interface. When a subsequent LIA/B 2 instruction is executed, the I/O interface receiving priority sets the A- or B-Register equal to its select code and identification data (ID) and passes priority to the next I/O interface. Having responded once it will not respond again unless Mode 1 is established again.

The next LIA/B 2 executed sets the A- or B-Register equal to the second I/O interface's select code and ID. The second I/O interface at completion of the instruction passes priority to the next I/O interface. This process continues until the last I/O interface responds. After the last I/O interface responds the next LIA/B 2 will not affect the A- or B-Register and therefore can be detected as a no response. (An OTA/B 2 with the A- or B-Register equal to 0 terminates this sequence.)

Mode 1 can also be used to retrieve the select code and ID of a desired I/O interface without going through the priority process. This is accomplished by establishing Mode 1 and then executing an LIA/B xx, where xx is the I/O interface select code. This procedure will not modify a priority sequence already in process. The Mode 1 select code and ID format are shown in Table 8-2.

Table 8-2. Diagnose Mode 1

A/B Bits	Meaning
15	Intelligent interface
14 13 12 11 10 9	Interface card type identification number
8 7 6	Interface card revision code
5 0	Interface card select code

Diagnose Mode 2

Diagnose Mode 2 causes an I/O interface to respond to an LIA/B 2 instruction in the same manner as in Mode 1 except that the data set into the A- or B-Register is as shown in Table 8-3.

Table 8-3. Diagnose Mode 2

A/B Bits	Meaning
15 14 13	Always zero
12	1 = Break feature is enabled
11	1 = Receiving interrupt priority
10	Always one
9	Control bit
8	Flag bit
7	1 = Global Register equals select code of interface card
6	Global Register enabled/disabled
5 4 3 2 1 0	Current Global Register value

Diagnose Mode 3

Diagnose Mode 3 allows an I/O chip to do a DMA transfer without affecting the I/O interface. When Mode 3 is entered the I/O chip does a DMA input transfer of the data in the configuration address register to the location in memory pointed to by the DMA address register. The configuration address register is incremented after each transfer so that the data can be verified. The transfer continues until the DMA count is incremented to zero. Mode 3 also prevents any STC instructions from generating a device command to the I/O interface.

Appendix A
Reference Tables and Conversions

This appendix contains the following reference information and tables:

- ASCII character codes

- Octal/decimal conversions

- Exponential and log equivalents

- Math constants

- Instruction codes in octal

- Base set instruction codes in binary

- Extend and overflow examples

- Interrupt and control summary

CHARACTER CODES

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
A	040400	000101
B	041000	000102
C	041400	000103
D	042000	000104
E	042400	000105
F	043000	000106
G	043400	000107
H	044000	000110
I	044400	000111
J	045000	000112
K	045400	000113
L	046000	000114
M	046400	000115
N	047000	000116
O	047400	000117
P	050000	000120
Q	050400	000121
R	051000	000122
S	051400	000123
T	052000	000124
U	052400	000125
V	053000	000126
W	053400	000127
X	054000	000130
Y	054400	000131
Z	055000	000132
a	060400	000141
b	061000	000142
c	061400	000143
d	062000	000144
e	062400	000145
f	063000	000146
g	063400	000147
h	064000	000150
i	064400	000151
j	065000	000152
k	065400	000153
l	066000	000154
m	066400	000155
n	067000	000156
o	067400	000157
p	070000	000160
q	070400	000161
r	071000	000162
s	071400	000163
t	072000	000164
u	072400	000165
v	073000	000166
w	073400	000167
x	074000	000170
y	074400	000171
z	075000	000172
0	030000	000060
1	030400	000061
2	031000	000062
3	031400	000063
4	032000	000064
5	032400	000065
6	033000	000066
7	033400	000067
8	034000	000070
9	034400	000071
NUL	000000	000000
SOH	000400	000001
STX	001000	000002
ETX	001400	000003
EOT	002000	000004
ENQ	002400	000005

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
ACK	003000	000006
BEL	003400	000007
BS	004000	000010
HT	004400	000011
LF	005000	000012
VT	005400	000013
FF	006000	000014
CR	006400	000015
SO	007000	000016
SI	007400	000017
DLE	010000	000020
DC1	010400	000021
DC2	011000	000022
DC3	011400	000023
DC4	012000	000024
NAK	012400	000025
SYN	013000	000026
ETB	013400	000027
CAN	014000	000030
EM	014400	000031
SUB	015000	000032
ESC	015400	000033
FS	016000	000034
GS	016400	000035
RS	017000	000036
US	017400	000037
SPACE	020000	000040
!	020400	000041
''	021000	000042
#	021400	000043
$	022000	000044
%	022400	000045
&	023000	000046
'	023400	000047
(	024000	000050
)	024400	000051
*	025000	000052
+	025400	000053
,	026000	000054
-	026400	000055
.	027000	000056
/	027400	000057
:	035000	000072
;	035400	000073
<	036000	000074
=	036400	000075
>	037000	000076
?	037400	000077
@	040000	000100
[	055400	000133
\	056000	000134
]	056400	000135
^	057000	000136
—	057400	000137
`	060000	000140
{	075400	000173
		076000	000174
}	076400	000175
~	077000	000176
DEL	077400	000177

First Character — Second Character

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

A-2   Reference Tables and Conversions
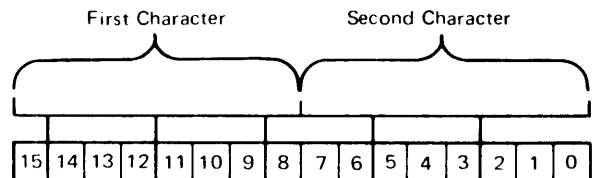
ADDITION

TABLE

0	01	02	03	04	05	06	07
1	02	03	04	05	06	07	10
2	03	04	05	06	07	10	11
3	04	05	06	07	10	11	12
4	05	06	07	10	11	12	13
5	06	07	10	11	12	13	14
6	07	10	11	12	13	14	15
7	10	11	12	13	14	15	16

EXAMPLE

```
Add:    3677    OCTAL
      + 1331    OCTAL

       (111-)   CARRIES

        5230    OCTAL
```

MULTIPLICATION

TABLE

1	02	03	04	05	06	07
2	04	06	10	12	14	16
3	06	11	14	17	22	25
4	10	14	20	24	30	34
5	12	17	24	31	36	43
6	14	22	30	36	44	52
7	16	25	34	43	52	61

EXAMPLE

```
Multiply:   657    OCTAL
          ×  54    OCTAL

          3274
          4153

         45024    OCTAL
```

(Reminder: add in octal)

COMPLEMENT

To find the two's complement form of an octal number. (Same procedure whether converting from positive to negative or negative to positive.)

RULE

1. Subtract from the maximum representable octal value.

2. Add one.

EXAMPLE

Two's complement of 556_8

```
  177777
− 000556

  177221
+      1

 177222₈
```

OCTAL/DECIMAL CONVERSIONS

OCTAL TO DECIMAL

TABLE

OCTAL	DECIMAL
0- 7	0- 7
10-17	8-15
20-27	16-23
30-37	24-31
40-47	32-39
50-57	40-47
60-67	48-55
70-77	56-63
100	64
200	128
400	256
1000	512
2000	1024
4000	2048
10000	4096
20000	8192
40000	16384
77777	32767

EXAMPLE

Convert 463_8 to a decimal integer.

$$400_8 = 256_{10}$$
$$60_8 = 48_{10}$$
$$3_8 = \underline{\quad 3_{10}}$$
$$307 \quad \text{decimal}$$

DECIMAL TO OCTAL

TABLE

DECIMAL	OCTAL
1	1
10	12
20	24
40	50
100	144
200	310
500	764
1000	1750
2000	3720
5000	11610
10000	23420
20000	47040
32767	77777

EXAMPLE

Convert 5229_{10} to an octal integer.

$$5000_{10} = 11610_8$$
$$200_{10} = 310_8$$
$$20_{10} = 24_8$$
$$9_{10} = \underline{\quad 11_8}$$
$$12155_8$$

(Remainder: add in octal)

NEGATIVE DECIMAL TO TWO'S COMPLEMENT OCTAL

TABLE

DECIMAL	2's COMP
-1	177777
-10	177766
-20	177754
-40	177730
-100	177634
-200	177470
-500	177014
-1000	176030
-2000	174060
-5000	166170
-10000	154360
-20000	130740
-32768	100000

EXAMPLE

Convert -629_{10} to two's complement octal.

$$-500_{10} = 177014_8$$
$$-100_{10} = 177634_8$$
$$-20_{10} = 177754_8 \quad \text{(Add in octal)}$$
$$-9_{10} = \underline{\quad 177767_8}$$
$$176613_8$$

For reverse conversion (two's complement octal to negative decimal):
1. Complement, using procedure on facing page.
2. Convert to decimal, using OCTAL TO DECIMAL table.

8200-44

A-4   Reference Tables and Conversions

$2 \pm \eta$ IN DECIMAL

2^η	η	$2^{-\eta}$
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.01562 5
128	7	0.00781 25
256	8	0.00390 625
512	9	0.00195 3125
1 024	10	0.00097 65625
2 048	11	0.00048 82812 5
4 096	12	0.00024 41406 25
8 192	13	0.00012 20703 125
16 384	14	0.00006 10351 5625
32 768	15	0.00003 05175 78125
65 536	16	0.00001 52587 89062 5
131 072	17	0.00000 76293 94531 25
262 144	18	0.00000 38146 97265 625
524 288	19	0.00000 19073 48632 8125
1 048 576	20	0.00000 09536 74316 40625
2 097 152	21	0.00000 04768 37158 20312 5
4 194 304	22	0.00000 02384 18579 10156 25
8 388 608	23	0.00000 01192 09289 55078 125
16 777 216	24	0.00000 00596 04644 77539 0625
33 554 432	25	0.00000 00298 02322 38769 53125
67 108 864	26	0.00000 00149 01161 19384 76562 5
134 217 728	27	0.00000 00074 50580 59692 38281 25
268 435 456	28	0.00000 00037 25290 29846 19140 625
536 870 912	29	0.00000 00018 62645 14923 09570 3125
1 073 741 824	30	0.00000 00009 31322 57461 54785 15625
2 147 483 648	31	0.00000 00004 65661 28730 77392 57812 5
4 294 967 296	32	0.00000 00002 32830 64365 38696 28906 25

$10 \pm \eta$ IN OCTAL

10^η	η	$10^{-\eta}$	10^η	η	$10^{-\eta}$
1	0	1.000 000 000 000 000 000 00	112 402 762 000	10	0.000 000 000 006 676 337 66
12	1	0.063 146 314 631 463 146 31	1 351 035 564 000	11	0.000 000 000 000 537 657 77
144	2	0.005 075 341 217 270 243 66	16 432 451 210 000	12	0.000 000 000 000 043 136 32
1 750	3	0.000 406 111 564 570 651 77	221 411 634 520 000	13	0.000 000 000 000 003 411 35
23 420	4	0.000 032 155 613 530 704 15	2 657 142 036 440 000	14	0.000 000 000 000 000 264 11
303 240	5	0.000 002 476 132 610 706 64	34 327 724 461 500 000	15	0.000 000 000 000 000 022 01
3 641 100	6	0.000 000 206 157 364 055 37	434 157 115 760 200 000	16	0.000 000 000 000 000 001 63
46 113 200	7	0.000 000 015 327 745 152 75	5 432 127 413 542 400 000	17	0.000 000 000 000 000 000 14
575 360 400	8	0.000 000 001 257 143 561 06	67 405 553 164 731 000 000	18	0.000 000 000 000 000 000 01
7 346 545 000	9	0.000 000 000 104 560 276 41			

2^x IN DECIMAL

x	2^x	x	2^x	x	2^x
0.001	1.00069 33874 62581	0.01	1.00695 55500 56719	0.1	1.07177 34625 36293
0.002	1.00138 72557 11335	0.02	1.01395 94797 90029	0.2	1.14869 83549 97035
0.003	1.00208 16050 79633	0.03	1.02101 21257 07193	0.3	1.23114 44133 44916
0.004	1.00277 64359 01078	0.04	1.02811 38266 56067	0.4	1.31950 79107 72894
0.005	1.00347 17485 09503	0.05	1.03526 49238 41377	0.5	1.41421 35623 73095
0.006	1.00416 75432 38973	0.06	1.04246 57608 41121	0.6	1.51571 65665 10398
0.007	1.00486 38204 23785	0.07	1.04971 66836 23067	0.7	1.62450 47927 12471
0.008	1.00556 05803 98468	0.08	1.05701 80405 61380	0.8	1.74110 11265 92248
0.009	1.00625 78234 97782	0.09	1.06437 01824 53360	0.9	1.86606 59830 73615

$\eta \log_{10} 2$, $\eta \log_2 10$ IN OCTAL

η	$\eta \log_{10} 2$	$\eta \log_2 10$	η	$\eta \log_{10} 2$	$\eta \log_2 10$
1	0.30102 99957	3.32192 80949	6	1.80617 99740	19.93156 85693
2	0.60205 99913	6.64385 61898	7	2.10720 99696	23.25349 66642
3	0.90308 99870	9.96578 42847	8	2.40823 99653	26.57542 47591
4	1.20411 99827	13.28771 23795	9	2.70926 99610	29.89735 28540
5	1.50514 99783	16.60964 04744	10	3.01029 99566	33.21928 09489

MATHEMATICAL CONSTANTS IN OCTAL SCALE

$\pi = (3.11037\ 552421)_{(8)}$

$\pi^{-1} = (0.24276\ 301556)_{(8)}$

$\sqrt{\pi} = (1.61337\ 611067)_{(8)}$

$\ln \pi = (1.11206\ 404435)_{(8)}$

$\log_2 \pi = (1.51544\ 163223)_{(8)}$

$\sqrt{10} = (3.12305\ 407267)_{(8)}$

$e = (2.55760\ 521305)_{(8)}$

$e^{-1} = (0.27426\ 530661)_{(8)}$

$\sqrt{e} = (1.51411\ 230704)_{(8)}$

$\log_{10} e = (0.33626\ 754251)_{(8)}$

$\log_2 e = (1.34252\ 166245)_{(8)}$

$\log_2 10 = (3.24464\ 741136)_{(8)}$

$\gamma = (0.44742\ 147707)_{(8)}$

$\ln \gamma = -(0.43127\ 233602)_{(8)}$

$\log_2 \gamma = -(0.62573\ 030645)_{(8)}$

$\sqrt{2} = (1.32404\ 746320)_{(8)}$

$\ln 2 = (0.54271\ 027760)_{(8)}$

$\ln 10 = (2.23273\ 067355)_{(8)}$

8200-46

MEMORY REFERENCE INSTRUCTIONS

INDIRECT ADDRESSING

Refer to octal instruction codes given on the following page.
To combine code for indirect addressing, merge "100000" with octal instruction code.

REGISTER REFERENCE INSTRUCTIONS

SHIFT-ROTATE GROUP (SRG)

1. select to operate A or B.
2. select 1 to 4 instructions, not more than one from each column.
3. combine octal codes (leading zeros omitted) by inclusive or.
4. order of execution is from column 1 to column 4.

A OPERATIONS

1	2	3	4
ALS (1000)	CLE (40)	SLA (10)	ALS (20)
ARS (1100)			ARS (21)
RAL (1200)			RAL (22)
RAR (1300)			RAR (23)
ALR (1400)			ALR (24)
ERA (1500)			ERA (25)
ELA (1600)			ELA (26)
ALF (1700)			ALF (27)

B OPERATIONS

1	2	3	4
BLS (5000)	CLE (4040)	SLB (4010)	BLS (4020)
BRS (5100)			BRS (4021)
RBL (5200)			RBL (4022)
RBR (5300)			RBR (4023)
BLR (5400)			BLR (4024)
ERB (5500)			ERB (4025)
ELB (5600)			ELB (4026)
BLF (5700)			BLF (4027)

ALTER-SKIP GROUP (ASG)

1. select to operate on A or B.
2. select 1 to 8 instructions, not more than one from each column.
3. combine octal codes (leading zeros omitted) by inclusive or.
4. order of execution is from column 1 to column 8.

A OPERATIONS

1	2	3	4
CLA (2400)	SEZ (2040)	CLE (2100)	SSA (2020)
CMA (3000)		CME (2200)	
CCA (3400)		CCE (2300)	

5	6	7	8
SLA (2010)	INA (2004)	SZA (2002)	RSS (2001)

B OPERATIONS

1	2	3	4
CLB (6400)	SEZ (6040)	CLE (6100)	SSB (6020)
CMB (7000)		CME (6200)	
CCB (7400)		CCE (6300)	

5	6	7	8
SLB (6010)	INB (6004)	SZB (6002)	RSS (6001)

INPUT/OUTPUT INSTRUCTIONS

CLEAR FLAG

Refer to octal instruction codes given on the following page.
To clear flag after execution (instead of holding flag), merge "001000" with octal instruction code.

8200-47

INSTRUCTION CODES IN OCTAL

Memory Reference									
		RSS	002001	CBY	105751	..FCM	105232	MW21	105736
		SEZ	002040	CMW	105776	.NGL	105214	MW22	105737
ADA	04(0XX)—	SLA	002010	CXA	101744	.SETP	105227	SIMP	105707
ADB	04(1XX)—	SLB	006010	CXB	105744	..TCM	105233	STMP	105703
AND	01(0XX)—	SSA	002020	CYA	101754	.XFER	105220	SPMR	105701
CPA	05(0XX)—	SSB	006020	CYB	105754	.ZFER	105237	SWMP	105706
CPB	05(1XX)—	SZA	002002	DSX	105761				
IOR	03(0XX)—	SZB	006002	DSY	105771	**Double Integer**			
ISZ	03(1XX)—			ISX	105760				
JMP	02(1XX)—	**Input/Output**		ISY	105770	.DAD	105014		
JSB	01(1XX)—			JLY	105762	.DCO	105204		
LDA	06(0XX)—	CLC	1067–	JPY	105772	.DDE	105211		
LDB	06(1XX)—	CLF	1031–	LAX	101742	.DDI	105074		
STA	07(0XX)—	CLO	103101	LAY	101752	.DDIR	105134		
STB	07(1XX)—	HLT	1020–	LBT	105763	.DDS	105213		
XOR	02(0XX)—	LIA	1025–	LBX	105742	.DIN	105210		
		LIB	1065–	LBY	105752	.DIS	105212		
Binary		MIA	1024–	LDX	105745	.DNG	105203		
		MIB	1064–	LDY	105755	.DMP	105054		
Shift-Rotate		OTA	1026–	MBT	105765	.DSB	105034		
		OTB	1066–	MVW	105777	.DSBR	105114		
ALF	001700	SFC	1022–	SAX	101740				
ALR	001400	SFS	1023–	SAY	101750	**VMA/EMA**			
ALS	001000	SOC	102201	SBS	105773				
ARS	001100	SOS	102301	SBT	105764	.IMAP	105250		
BLF	005700	STC	1027–	SBX	105740	.IRES	105244		
BLR	005400	STF	1021–	SBY	105750	.JMAP	105252		
BLS	005000	STO	102101	SFB	105767	.JRES	105245		
BRS	005100			STX	105743	.LBP	105257		
CLE	000040	**Extended Arithmetic**		STY	105753	.LBPR	105256		
ELA	001600			TBS	105775	.LPX	105255		
ELB	005600	ASL	1000(01X)–	XAX	101747	.LPXR	105254		
ERA	001500	ASR	1010(01X)–	XAY	101757	.PMAP	105240		
ERB	005500	DIV	100400	XBX	105747				
NOP	000000	JLA	100600	XBY	105757	**Oper. Syst. Set**			
RAL	001200	DLD	104200						
RAR	001300	DST	104400	**Floating Point**		.CPUID	105300		
RBL	005200	JLB	104600			.FWID	105301		
RBR	005300	LSL	1000(10X)–	FAD	105000	.SIP	105303		
SLA	000010	LSR	1010(10X)–	FDV	105060	.WFI	105302		
SLB	004010	MPY	100200	FIX	105100				
		RRL	1001(00X)–	FLT	105120				
Alter-Skip		RRR	1011(00X)–	FMP	105040	**Dynamic Map Syst.**			
				FSB	105020				
CCA	003400	**Binary**		.FIXD	105104	LDMP	105702		
CCB	007400			.FLTD	105124	LPMR	105700		
CCE	002300	**Ext. Inst. Group**		.TADD	105002	LWD1	105704		
CLA	002400			.TDIV	105062	LWD2	105705		
CLB	006400	ADX	105746	.TFTD	105122	MB00	101727		
CLE	002100	ADY	105756	.TFXD	105106	MB01	101730		
CMA	003000	CAX	101741	.TFXS	105102	MB02	101731		
CMB	007000	CAY	101751	.TMPY	105042	MB10	101732		
CME	002200	CBS	105774	.TSUB	105022	MB11	101733		
INA	002004	CBT	105766			MB12	101734		
INB	006004	CBX	105741	**Language Inst. Set**		MB20	101735		
						MB21	101736		
				.BLE	105207	MB22	101737		
				.CFER	105231	MW00	105727		
				.DFER	105205	MW01	105730		
				.CPM	105236	MW02	105731		
				.ENTC	105235	MW10	105732		
				.ENTN	105234	MW11	105733		
				.ENTP	105224	MW12	105734		
				.ENTR	105223	MW20	105735		

Assuming: no indirect addressing.
no combined instructions.
shifts taken in first position only.
hold flag after I/O execution.

* Not directly user callable.
Used by HP software.

Refer to preceding page for octal combining tables.

XCA1	101726
XCA2	101723
XCB1	105726
XCB2	105723
XJMP	105710
XJCQ	105711
XLA1	101724
XLA2	101721
XLB1	105724
XLB2	105721
XSA1	101725
XSA2	101722
XSB1	105725
XSB2	105722

Code and Data Sep.

ADQA	101413
ADQB	105413
CACQ	101407
CAZ	101411
CBCQ	105407
CBZ	105411
CCQA	101406
CCQB	105406
CIQA	101412
CIQB	105412
CZA	101410
CZB	105410
EXIT	105417
EXIT1	105415
EXIT2	105416
PCALI	105400
PCALN	105404
PCALR	105403
PCALV	105402
PCALX	105401
SDSP	105405

BASE SET INSTRUCTION CODES IN BINARY

15	14 13 12	11	10	9	8	7	6	5	4	3	2	1	0

MEMORY REFERENCE INSTRUCTIONS

15	14 13 12	11	10 9	8 7 6 5 4 3 2 1 0
D/I	AND 001	0	Z/C	← MEMORY ADDRESS →
D/I	XOR 010	0	Z/C	
D/I	IOR 011	0	Z/C	
D/I	JSB 001	1	Z/C	
D/I	JMP 010	1	Z/C	
D/I	ISZ 011	1	Z/C	
D/I	AD* 100	A/B	Z/C	
D/I	CP* 101	A/B	Z/C	
D/I	LD* 110	A/B	Z/C	
D/I	ST* 111	A/B	Z/C	

SHIFT/ROTATE GROUP

15	14 13 12	11	10	9	8 7 6		5	4	3	2 1 0	
0	000	A/B	0	D/E	*LS	000	†CLE	D/E	‡SL*	*LS	000
		A/B	0	D/E	*RS	001		D/E		*RS	001
		A/B	0	D/E	R*L	010		D/E		R*L	010
		A/B	0	D/E	R*R	011		D/E		R*R	011
		A/B	0	D/E	*LR	100		D/E		*LR	100
		A/B	0	D/E	ER*	101		D/E		ER*	101
		A/B	0	D/E	EL*	110		D/E		EL*	110
		A/B	0	D/E	*LF	111		D/E		*LF	111
		NOP 000				000		000			000

ALTER/SKIP GROUP

15	14 13 12	11	10	9 8		7 6		5	4	3	2	1	0
0	000	A/B	1	CL*	01	CLE	01	SEZ	SS*	SL*	IN*	SZ*	RSS
		A/B		CM*	10	CME	10						
		A/B		CC*	11	CCE	11						

INPUT/OUTPUT GROUP

15	14 13 12	11	10	9	8 7		6	5 4 3 2 1 0
1	000	1	H/C	HLT	000		← SELECT CODE →
		1	0	STF	001		
		1	1	CLF	001		
		1	0	SFC	010		
		1	0	SFS	011		
		A/B	1	H/C	MI*	100	
		A/B	1	H/C	LI*	101	
		A/B	1	H/C	OT*	110	
		0	1	H/C	STC	111	
		1	1	H/C	CLC	111	
		1	0	STO	001	000	001
		1	1	CLO	001	000	001
		1	H/C	SOC	010	000	001
		1	H/C	SOS	011	000	001

EXTENDED ARITHMETIC GROUP

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	000	MPY** 000	010	000	000
		DIV** 000	100	000	000
		JLA 000	110	000	000
		DLD** 100	010	000	000
		DST** 100	100	000	000
		JLB 100	110	000	000
		ASR 001	000	0 1	NUMBER OF BITS →
		ASL 000	000	0 1	
		LSR 001	000	1 0	
		LSL 000	000	1 0	
		RRR 001	001	0 0	
		RRL 000	001	0 0	

FLOATING POINT INSTRUCTIONS

15	14 13 12	11 10 9	8 7	6 5 4	3	2 1 0
1	000	101	00	FAD 000	0	000
				FSB 001		
				FMP 010		
				FDV 011		
				FIX 100		
				FLT 101		

Notes: * = A or B, according to bit 11. †CLE: Only this bit is required.
D/I, A/B, Z/C, D/E, H/C coded 0/1. ‡SL*: Only this bit and bit 11 (A/B as applicable) are required.
**Second word is Memory Address.

8200-56

A-10 Reference Tables and Conversions

Bit positions: **15 | 14 13 12 | 11 10 9 | 8 7 | 6 | 5 4 3 | 2 1 0**

FLOATING POINT INSTRUCTION (Continued)

15	14 13 12	11 10 9	8 7	6 5 4	3	2 1 0
1	000	101	00	.TADD 000	0	010
				.TSUB 001		
				.TMPY 010		
				.TDIV 011		
				.TFXS 100		
				.TFTS 101		
				.FIXD 100		100
				.FLTD 101		
				.TFXD 100		110
				.TFTD 101		

DOUBLE INTEGER INSTRUCTIONS

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	000	101	000	001	.DAD 100
				011	.DSB 100
				101	.DMP 100
				111	.DDI 100
			001	001	.DSBR 100
			001	011	.DDIR 100
			010	000	.DNG 011
					.DCO 100
				001	.DIN 000
					.DDE 001
					.DIS 010
					.DDS 011

LANGUAGE INSTRUCTION SET

15	14 13 12	11 10 9	8 7	6	5 4	2 1 0
1	000	101	010	0	00	.DFER 101
						.BLE 111
					01	.NGL 100
					10	.XFER 000
						.ENTR 011
						.ENTP 100
						.SETP 111
					11	.CFER 001
						..FCM 010
						..TCM 011
						.ENTN 100
						.ENTC 101
						.CPM 110
						.ZFER 111

VIRTUAL MEMORY INSTRUCTIONS

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	000	101	010	100	.PMAP 000
					.IRES 100
					.JRES 101
				101	.IMAP 000
					.JMAP 010
					.LPXR 100
					.LPX 101
					.LBPR 110
					.LBP 111

OPERATING SYSTEM INSTRUCTION SET

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
1	000	101	011	000	.CPUID 000
					.FWID 001
					.WFI 010
					.SIP 011

BASE SET INSTRUCTION CODES IN BINARY (Continued)

15	14 13 12	11	10 9	8 7 6	5 4 3	2 1 0
DMS INSTRUCTIONS						
1	000	1	01	111	000	LPMR 000
		1				SPMR 001
		1				LDMP 010
		1				STMP 011
		1				LWD1 100
		1				LWD2 101
		1				SWMP 110
		1				SIMP 111
		1			001	XJMP 000
		A/B			010	XL*1 100
		A/B				XS*1 101
		A/B				XC*1 110
		A/B				XL*2 001
		A/B				XS*2 010
		A/B				XC*2 011
		B/W				M°00 111
		B/W			011	M°01 000
		B/W				M°02 001
		B/W				M°10 010
		B/W				M°11 011
		B/W				M°12 100
		B/W				M°20 101
		B/W				M°21 110
		B/W				M°22 111
SCIENTIFIC INSTRUCTION SET						
1	000		101	011	010	TAN 000
						SQRT 001
						ALOG 010
						ATAN 011
						COS 100
						SIN 101
						EXP 110
						ALOGT 111
					011	TANH 000
						DPOLY 001
						/CMRT 010
						/ATLG 011
						.FPWR 100
						.TPWR 101
VECTOR INSTRUCTION SET						
1	000		101	000	000	VADD 001
						VSUB 011
						VMPY 100
						VDIV 101
						VSAD 110
						VSSB 111
					001	VSMY 000
						VSDV 001
					010	DVADD 001
						DVSUB 011
						DVMPY 100
						DVDIV 101
						DVSAD 110
						DVSSB 111
					011	DVSMY 000
						DVSDV 001
				001	000	VPIV 001
						VABS 011
						VSUM 101

Notes: * = A (0) or B (1), according to bit 11.
      ° = B (0) or W (1), according to bit 11.

15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
VECTOR INSTRUCTION SET (Continued)					
				001	VNRM 111
					VDOT 000
					VMAX 001
					VMAB 010
					VMIN 011
					VMIB 101
					VMOV 110
					VSWP 111
				010	DPIV 001
					DVABS 011
					DVSUM 101
					DVNRM 111
1	000	101	001	011	DVDOT 000
					DVMAX 001
					DVMAB 010
					DVMIN 011
					DVMIB 101
					DVMOV 110
					DVSWP 111
CODE AND DATA SEPARATION					
1	000	001	100	000	CCQA 110
					CACQ 111
				001	CZA 000
					CAZ 001
					CIQA 010
					ADQA 011
		101	100	000	PCALI 000
					PCALX 001
					PCALV 010
					PCALR 011
					PCALN 100
					SDSP 101
					CCQB 110
					CBCQ 111
				001	CZB 000
					CBZ 001
					CIQB 010
					ADQB 011
					EXIT1 101
					EXIT2 110
					EXIT 111

EXTENDED INSTRUCTION GROUP

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAX/SAY/SBX/SBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	0	0
CAX/CAY/CBX/CBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	0	1
LAX/LAY/LBX/LBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	1	0
STX/STY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	0	1	1
CXA/CYA/CXB/CYB	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	1	0	0
LDX/LDY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	1	0	1
ADX/ADY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	1	1	0
XAX/XAY/XBX/XBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	1	1	1
ISX/ISY/DSX/DSY	1	0	0	0	1	0	1	1	1	1	1	1	X/Y	0	0	I/D
JUMP INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	////	0	1	0

JLY = 0
JPY = 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BYTE INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	0	////	////	////

LBT = 0 1 1
SBT = 1 0 0
MBT = 1 0 1
CBT = 1 1 0
SFB = 1 1 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIT INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	1	////	////	////

SBS = 0 1 1
CBS = 1 0 0
TBS = 1 0 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WORD INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	1	1	////	////

CMW = 0
MVW = 1

8200-52

EXTEND AND OVERFLOW EXAMPLES

SAME SIGN (POSITIVE)

SIGN

A/B REGISTER
AUGEND → `0` `0`

ADDEND → `0` `0`

RESULT A/B → `0` OV=UNCHANGED
E=UNCHANGED

SIGN

A/B REGISTER
AUGEND → `0` `1`

ADDEND → `0` `1`

RESULT A/B → `1` OV=1
E=UNCHANGED

SAME SIGN (NEGATIVE)

SIGN

A/B REGISTER
AUGEND → `1` `0`

ADDEND → `1` `0`

RESULT A/B → `0` OV=1
E=1

SIGN

A/B REGISTER
AUGEND → `1` `1`

ADDEND → `1` `1`

RESULT A/B → `1` OV=UNCHANGED
E=1

DIFFERENT SIGNS

SIGN

A/B REGISTER
AUGEND → `0` `0`

ADDEND → `1` `0`

RESULT A/B → `1` OV=UNCHANGED
E=UNCHANGED

SIGN

A/B REGISTER
AUGEND → `0` `1`

ADDEND → `1` `1`

RESULT A/B → `0` OV=UNCHANGED
E=1

SIGN

A/B REGISTER
AUGEND → `1` `1`

ADDEND → `0` `1`

RESULT A/B → `0` OV=UNCHANGED
E=1

SIGN

A/B REGISTER
AUGEND → `1` `0`

ADDEND → `0` `1`

RESULT A/B → `1` OV=UNCHANGED
E=UNCHANGED

8200-48

INTERRUPT AND CONTROL SUMMARY

INST	S.C. 00	S.C. 01	S.C. 02	S.C. 03	S.C. 04	S.C. 05	S.C. 06	S.C. 07
STC	NOP	NOP	Enable break mode.	NOP	Enable Type 2 and 3 interrupts.	Enable parity error interrupts.	Turn on Time Base Generator.	Turn on memory protect.
CLC	System reset.	NOP	NOP	NOP	Disable Type 2 and 3 interrupts.	Disable parity error interrupts.	Turn off Time Base Generator.	NOP
STF	Enable Type 3 interrupts.	STO	Disable Global Register.	NOP	NOP	Set parity sense to even parity.	Set Time Base Generator flag.	NOP
CLF	Disable Type 3 interrupts.	CLO	Enable Global Register.	NOP	NOP	Set parity sense to odd parity.	Clear Time Base Generator flag.	NOP
SFS	Skip if Type 3 interrupts are enabled.	SOS	Skip if Global Register is disabled.	NOP	Skip if power not going down	Skip if parity sense is even.	Skip if Time Base Generator flag is set.	NOP
SFC	Skip if Type 3 interrupts are disabled.	SOC	Skip if Global Register is enabled.	NOP	Skip if power is going down.	Skip if parity sense is odd.	Skip if Time Base Generator flag is clear.	NOP
LI*	Load from interrupt mask register.	Load from processor status register.	Load from Global Register.	Load from PSAVE or (with ,C) ROMP.	Load from central interrupt register.	Load bits 0-15 from parity error register, or (with ,C) bits 16-23.	NOP	Load from violation register.
MI*	NOP	Merge from processor status register.	NOP	NOP	NOP	NOP	NOP	NOP
OT*	Output to interrupt mask register.	Output to processor status register.	Output to Global Register. (Note 1)	Output to PSAVE or (with ,C ROMP.	Output to central interrupt register.	NOP	NOP	NOP

Note 1: An OTA/B 2 with A/B equal to one through seven establishes a diagnose mode; refer to "Diagnose Modes" paragraph in Chapter 8 for details.

HEWLETT
PACKARD