CRAY RESEARCH, INC.

CRAY-1® COMPUTER SYSTEMS

CRAY-1 S SERIES
HARDWARE
REFERENCE MANUAL

HR-0808

CRAY RESEARCH, INC.

CRAY-1® COMPUTER SYSTEMS

CRAY-1 S SERIES
HARDWARE
REFERENCE MANUAL

HR-0808

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER HR-0808

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	June, 1980 - Original printing
01	December, 1980 - This change corrects and updates material for the Disk Storage Channel and the Block Multiplexer Channel of the I/O Subsystem. Parts 3 and 4 are affected.
A	June, 1981 - This revision incorporates change packet 01. No other changes have been made.
02	July, 1981 - This change corrects the Central memory sizes and phasing associated with certain configurations. Part 1, section 2 and part 2, section 2 are affected.
B	September, 1981 - Reprint with revision. This printing includes technical corrections and information on the expanded I/O Subsystem. All previous printings are obsolete.
B-01	November, 1981 - Change packet. This packet corrects block multiplexer channel information in part 3, section 7, resulting in partial reorganization of the section. Miscellaneous changes were made in part 2, section 6.

CONTENTS

FIGURES (continued)

TABLES

PART 2 - CENTRAL PROCESSING UNIT

FIGURES

FIGURES (continued)

TABLES

PART 3 - I/O SUBSYSTEM

FIGURES

FIGURES (continued)

TABLES

PART 4 - APPENDIX SECTION

FIGURES

TABLES

PART 1

SYSTEM

SYSTEM DESCRIPTION 1

INTRODUCTION

The CRAY-1 S Series of Computer Systems is based on a powerful general-purpose central processing unit CPU capable of extremely high processing rates. These rates are achieved by combining scalar and vector capabilities into the CPU, which is joined to a large, fast, bipolar integrated circuit memory. Vector processing, which is the performance of iterative operations on sets of ordered data, provides results at rates greatly exceeding the result rates of conventional scalar processing. Scalar operations complement the vector capability by providing solutions to problems not readily adaptable to vector techniques. Table 1-1 summarizes the models available in the S Series of computer systems. These models are described in greater detail in section 2 under System Configurations. The Model S/250 is no longer available.

Table 1-1. Models of the CRAY-1 S Series of Computer Systems

MODEL	S/250	S/500	S/1000	S/1200	S/1300	S/1400	S/2200	S/2300	S/2400	S/4200	S/4300	S/4400
CPU Central Memory size (64-bit words)	1/4M	1/2M	1M	1M	1M	1M	2M	2M	2M	4M	4M	4M
FRONT-END INTERFACES	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3
I/O SUBSYSTEM I/O Processors				2	3	4	2	3	4	2	3	4
Buffer memory Size				1-8M	1-8M	1-8M	1-8M	1-8M	1-8M	1-8M	1-8M	1-8M
DCU-4 Disk Controller Units				1-4	1-8	1-12	1-4	1-8	1-12	1-4	1-8	1-12
DD-29 Disk Storage Units				2-16	2-32	2-48	2-16	2-32	2-48	2-16	2-32	2-48
Block Multiplexer Controllers					1-4	1-4		1-4	1-4		1-4	1-16
Block Multiplexer Channels					1-16	1-16		1-16	1-16		1-16	1-16
MASS STORAGE SUBSYSTEM DCU-3 Disk Control Units	2-8	2-8	2-8									
DD-29 Disk Storage Units	2-32	2-32	2-32									

The CRAY-1 S Series of Computer Systems encompasses a wide variety of configurations. On the advanced models, a sophisticated I/O Subsystem matches the high processing rates with high input/output transfer rates for communication with mass storage units, other peripheral devices, and a wide variety of host computers. Several combinations of memory size and I/O capabilities are offered.

An optimum system can be configured for a particular use. This section briefly describes the system components, configurations, and operation. Table 1-2 gives the overall system characteristics. Figure 1-1 illustrates a typical system.

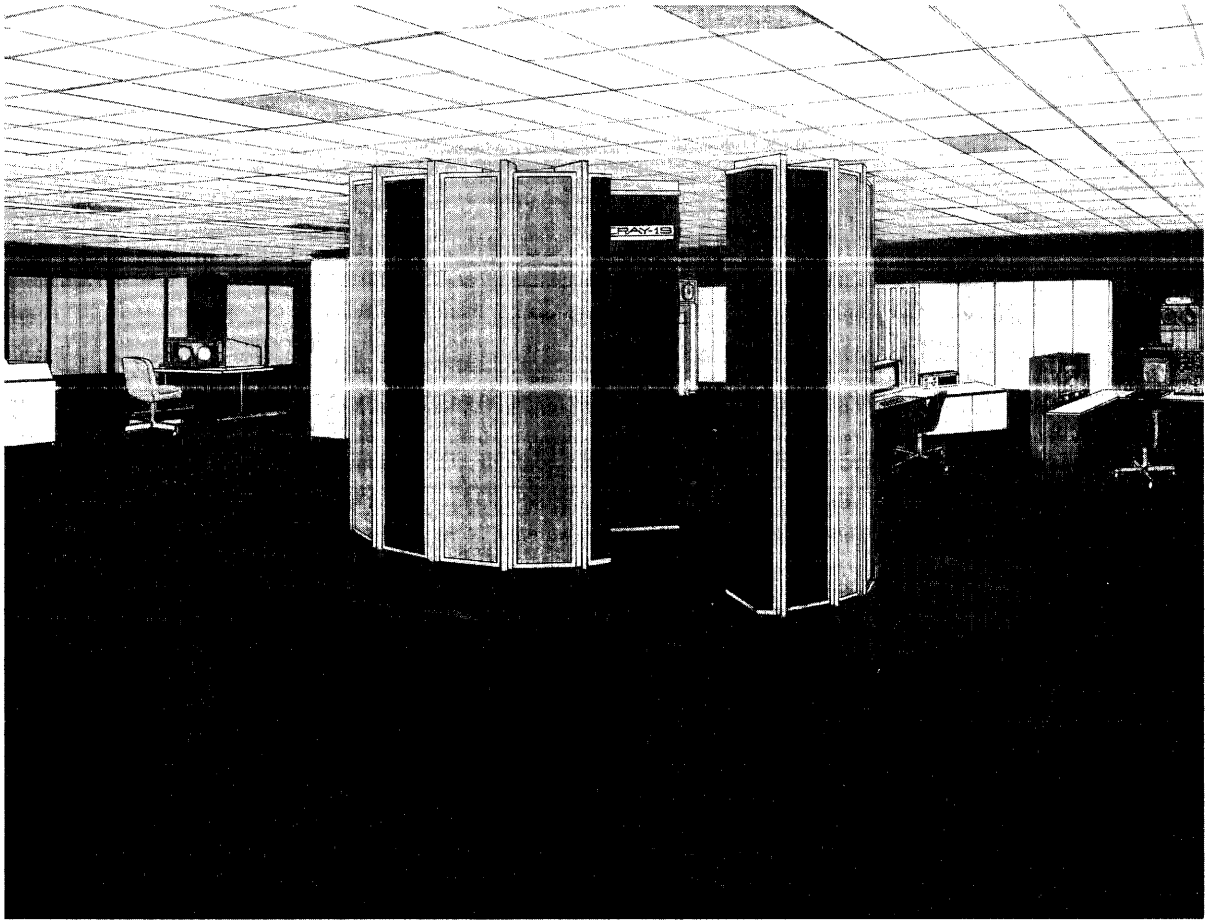


Figure 1-1. Typical CRAY-1 Computer System

Table 1-2. CRAY-1 System characteristics

Configuration	- Central Processing Unit - Maintenance Control Unit (MCU) or I/O Subsystem with 2 to 4 I/O Processors
CPU Speed	- 12.5 ns CPU clock period - 80 million floating-point additions per second rate - 80 million floating-point multiplications per second rate - Simultaneous floating-point addition and multiplication - 80 million 1/2 precision floating-point divisions per second rate - 25 million full precision floating-point divisions per second rate
Memories	- Up to 4 million 64-bit words in CPU Central Memory - 65 thousand 16-bit parcels in each I/O Processor I/O Memory - 1 to 8 million 64-bit words of I/O Subsystem Buffer Memory
Mass Storage	- 600 million byte disk drive - 48 disk drives maximum - 38.7 Mbits/s disk drive transfer rate
Input/Output	- Up to 12 CPU channel pairs (if no I/O Subsystem) - 1 standard and 1 optional Memory Channel to CPU; approx. 800 Mbits/s - Many other mainframe interfaces - 6 Direct memory access (DMA) ports to each I/O Processor - 40 channels; input or output sharing the six DMA ports per I/O Processor
Physical	- 100 sq ft floor space for CPU - 10 sq ft floor space for I/O Subsystem - 5.25 tons, CPU weight - 1.5 tons, I/O Subsystem weight - Liquid refrigeration of each chassis - 400 Hz power from motor-generators

SYSTEM COMPONENTS

The CRAY-1 Computer System is composed of a Central Processing Unit (CPU), a Maintenance Control Unit or I/O Subsystem, and mass storage devices. Supporting this equipment are condensing units for refrigeration, power distribution units for the CPU and the I/O Subsystem, and motor-generators providing system power.

CENTRAL PROCESSING UNIT

The CPU is a single integrated processing unit having a memory section, a control section, a computation section, and an input/output section.

The memory section contains from 262,144 to 4,194,304 words of 64 bits each. Memory is organized in 8 or 16 interleaved banks to allow fast access to successive addresses. The bipolar integrated circuits and the interleaving give a 4-clock-period (50 nanosecond) bank cycle time and an overall memory performance of one word every 12.5 nanoseconds. Single error correction/double error detection (SECDED) protects data integrity.

The control section contains registers and circuitry that controls instruction issue, transfers execution from program to program, and recognizes modes, error flags, and interrupt conditions.

The computation section handles the scalar and vector operations. It is composed of scalar, vector, and address registers; instruction buffers; and segmented functional units.

The input/output (I/O) section contains up to 12 channel pairs; each input or output channel carries 16 data bits, 3 control bits, and 4 parity bits. Normal channel speed is 50 or 160 Mbits per second, maximum. A single Memory channel communicates with the I/O Subsystem at approximately 850 Mbits per second.

Because superior performance is achieved using the Memory channel and I/O Subsystem channels, all but one of the CPU channels are generally not used when an I/O Subsystem is present.

The Central Processing Unit computation section is located in four columns of mainframe chassis. An additional four or eight columns contain Central Memory. Some versions of the CRAY-1 Computer System use the 8-column chassis with four columns of memory and four columns of computation section. Other models have an additional four columns to house more memory. These variations are discussed later with the description of each configuration. The bench at the base of each column houses the DC power supplies for that column. Figure 1-2 shows these variations of CPU chassis. Part 2 of this publication describes the CPU in detail.

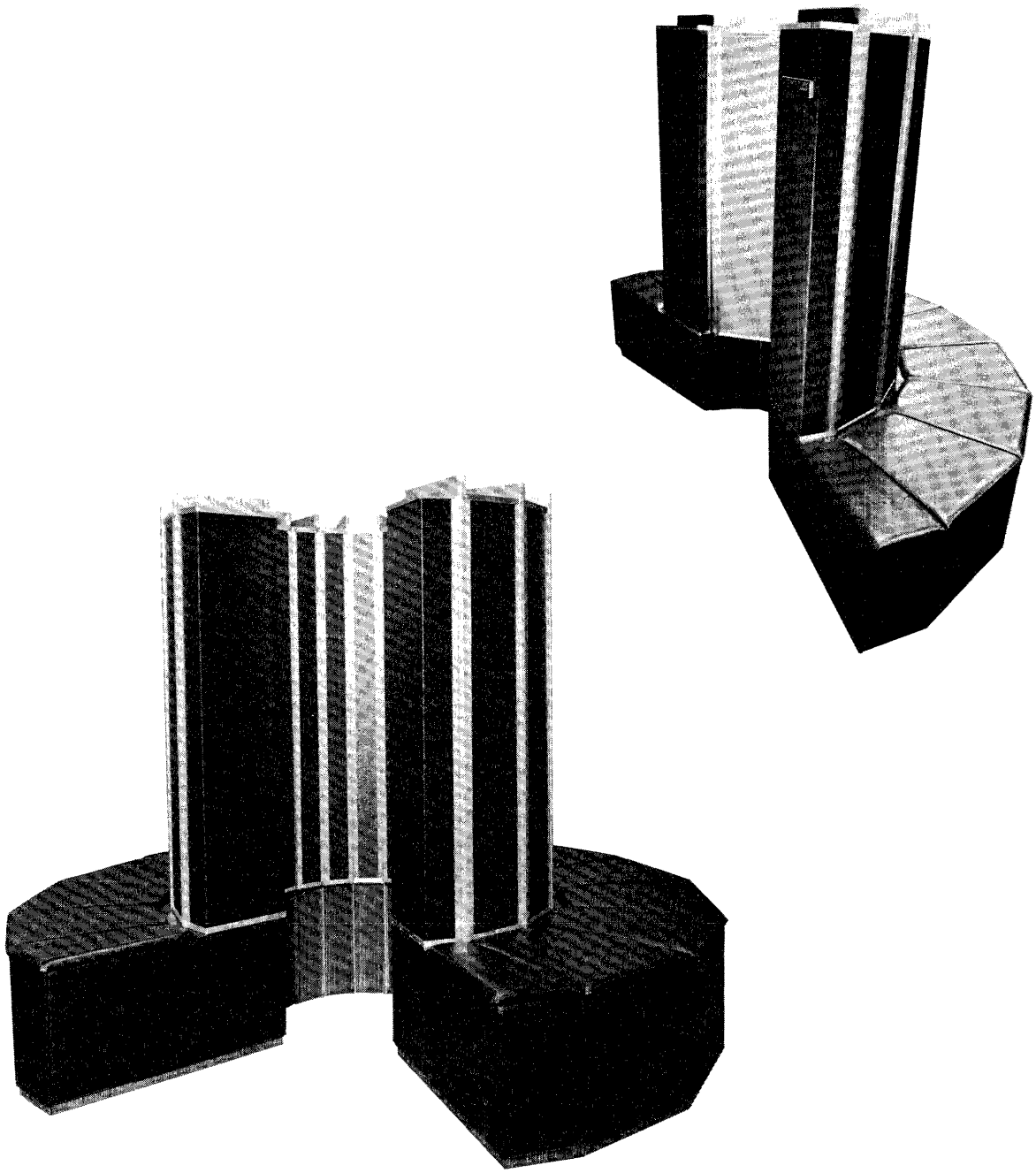


Figure 1-2. Central Processing Unit chassis variations

MAINTENANCE CONTROL UNIT

The Maintenance Control Unit (MCU) is used for supervisory operation and maintenance on CRAY-1 System Models S/250, S/500, and S/1000. It consists of a minicomputer, a tape drive, a card reader, a removable pack disk drive, a line printer, and 2 operator consoles. It may be used to enter jobs locally. Figure 1-3 shows the Maintenance Control Unit.

Figure 1-3. Maintenance Control Unit

INPUT/OUTPUT SUBSYSTEM

Models S/1200 through S/4400 of the CRAY-1 S Series are equipped with an I/O Subsystem composed of 2, 3, or 4 I/O Processors, a Buffer Memory, and required interfaces. Each I/O Processor (IOP) is independent and handles some portion of the I/O requirements for the system. I/O Processors are designed for fast data transfer between a front-end computer peripheral device and Buffer Memory or between Buffer Memory and the Central Processing Unit. Each I/O Processor has a computation section, a memory section, a control section, and an I/O section. Part 3 of this publication describes the I/O Processor in detail. One I/O Processor is assigned a supervisory role for the group of I/O Processors.

The computation section of an IOP has functional units and operand registers, and uses an accumulator in single-address operation.

The I/O Processor Memory holds 65,536 words of 16 bits each in bipolar integrated circuits; 16-bank phasing is used in this memory to maintain high-speed data transfers.

The I/O Processor section has six direct memory access (DMA) ports divided among the channels. A DMA port can transfer data at peak rates of approximately 850 Mbits per second; one port may be receiving data at the same time another port is sending data. Interfaces with internal buffering connect the I/O Processor to mass storage devices or to other high-speed equipment. Several independent channels may share one DMA port and this is done for some peripheral devices such as disk storage units. A standard set of peripherals provide for supervisory control and for maintenance: a tape drive, a line printer, and two operator consoles. Cray Research interfaces connect I/O Processors with front-end or host computers for network operation. Appendix F lists the various functions assigned to channels in a typical I/O Subsystem.

The Buffer Memory assists data transfer between peripheral devices and the Central Processing Unit. Buffer memory stores 1 million to 8 million words of 64 bits each, using single error correction/double error detection (SECDED) data protection. All I/O Processors share the Buffer Memory. The Central Processing Unit is not directly connected to the Buffer Memory. An I/O Processor relays data between the Buffer Memory and the CPU.

The I/O Subsystem is housed in a 4-column chassis similar to the CPU chassis. Figure 1-4 shows this chassis.

MASS STORAGE

The basic unit of mass storage for the CRAY-1 Series Computer Systems is the DD-29 Disk Storage Unit (DSU). This is a 600 Mbyte disk drive having a maximum data transfer rate of 35.4 Mbits per second.

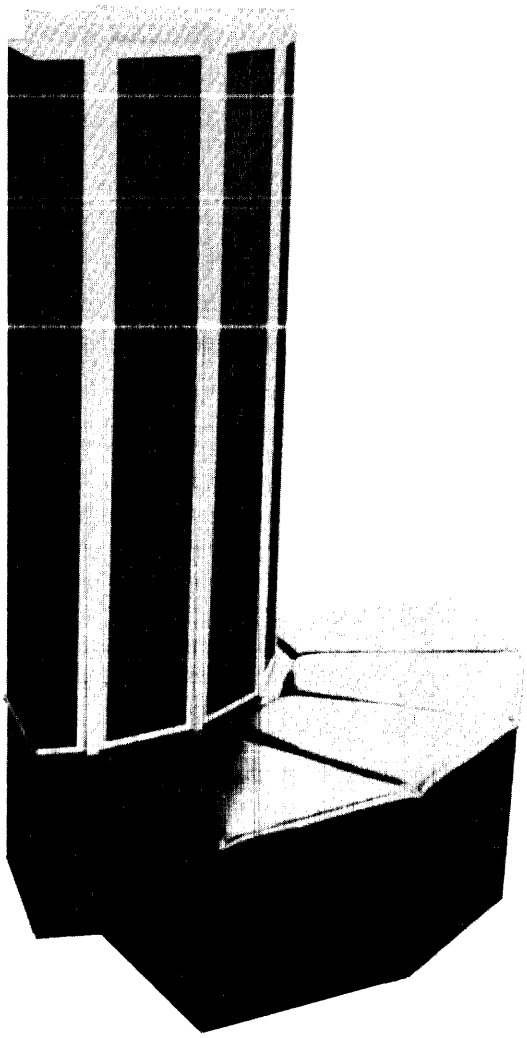


Figure 1-4. I/O Subsystem

On Models S/250, S/500, and S/1000, up to four DD-29s can be connected to one DCU-3 Disk Controller Unit (figure 1-5). A mass storage subsystem composed of 2 to 8 DCU-3 Disk Controller Units and 2 to 32 Disk Storage Units can be configured on the Models S/250, S/500, and S/1000. However, the actual number of units depends on the number of available I/O channel pairs (e.g., if 4 channel pairs are used from front-end computer systems, including 1 for the MCU, the remaining 8 can be used for mass storage.) Operating and programming information for the DCU-3 is included in CRI Publication HR-0630.

On models S/1200 through S/4400, up to four DD-29s can be connected to one DCU-4 Disk Controller Unit. The DCU-4 Controller Unit interfaces the four disk units with the I/O Processor through two DMA ports. The I/O Processor and the Disk Controller Unit are fast enough to keep all four DSUs operating at full speed without missing data or skipping revolutions. A minimum of 2 and a maximum of 48 DD-29s can be configured. Figure 1-6 shows a DD-29 Disk Storage Unit. The DCU-4 Disk Controller Unit is housed in the I/O Subsystem chassis.

Each DD-29 Disk Storage Unit has two ports for controllers. A second independent data path to each disk storage unit may exist through another Cray Research controller. Reservation logic is provided to control access to each disk storage unit.

Operational characteristics of the DD-29 Disk Storage Units are summarized in table 1-3. Further information about the DCU-4 and DD-29 is presented in part 3, section 7 of this publication.

Table 1-3. Characteristics of a DD-29 Disk Storage Unit

Bit capacity per drive	4.848×10^9	Maximum Latency	16.6 ms
Tracks per surface	823	Access time	15 - 80 ms
Sectors per track	18	Data transfer rate	38.7×10^6 bits/s
Bits per sector	32,768		
CPU words per sector	512		
Head groups per drive	10	Bits per cylinder	5.9×10^6
Recording surfaces per drive	40	CPU words per cylinder	92,160

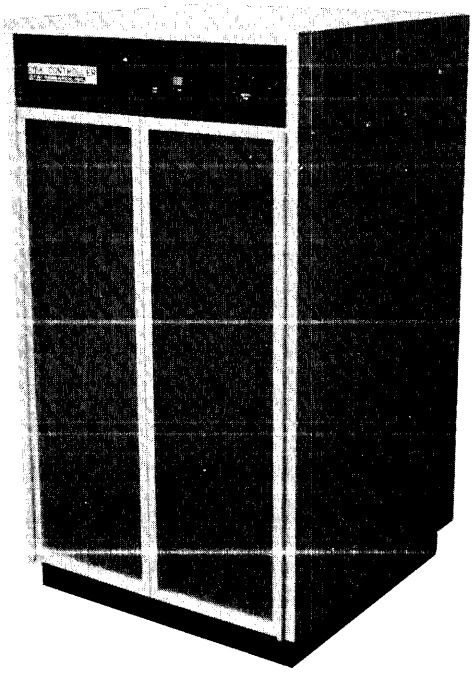


Figure 1-5. DCU-3 Disk Controller Unit

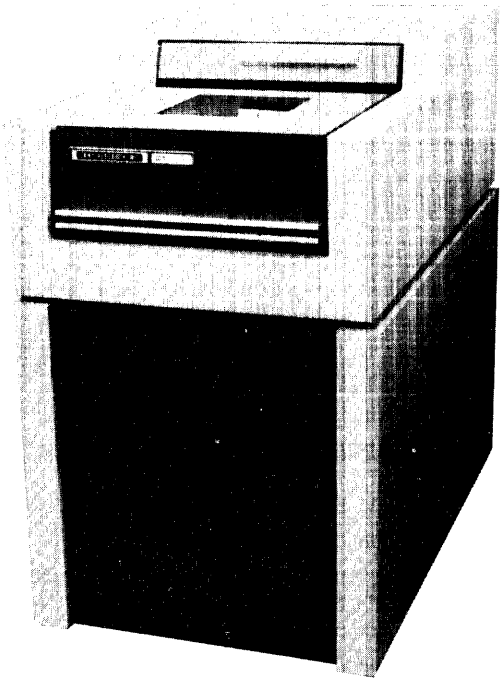


Figure 1-6. DD-29 Disk Storage Unit

CONDENSING UNITS

The condensing units contain the major components of the refrigeration system used to cool the computer chassis. Heat is removed from the condensing unit by a secondary cooling system that is not part of the CRAY-1 Computer System. Figure 1-7 shows the condensing unit.

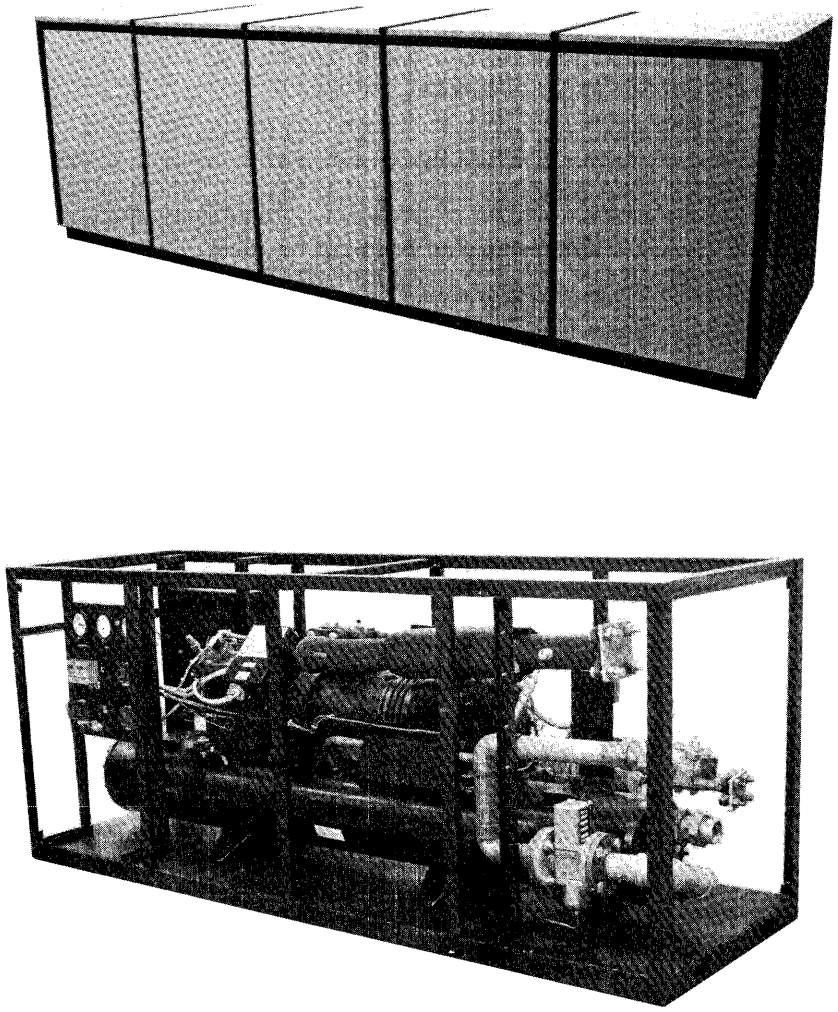


Figure 1-7. Condensing Unit

POWER DISTRIBUTION UNITS

The Central Processing Unit, I/O Subsystem, and disk controller units all operate from 400 Hz 3-phase power. The Power Distribution Unit (PDU) for the CPU contains adjustable transformers to regulate the voltage to each power supply for the CPU. The PDU also contains temperature monitoring equipment that checks temperatures at strategic locations on the CPU chassis. Automatic warning and shutdown circuitry protect the mainframe in case of overheating or excessive cooling. The control switches for the motor-generators and the Condensing Unit are mounted on the CPU Power Distribution Unit.

The Power Distribution Unit for the I/O Subsystem performs similar functions for the I/O Subsystem chassis.

The disk controller units have these functions built into the disk controller unit cabinet.

Figure 1-8 shows the Power Distribution Units for the CPU and for the I/O Subsystem.

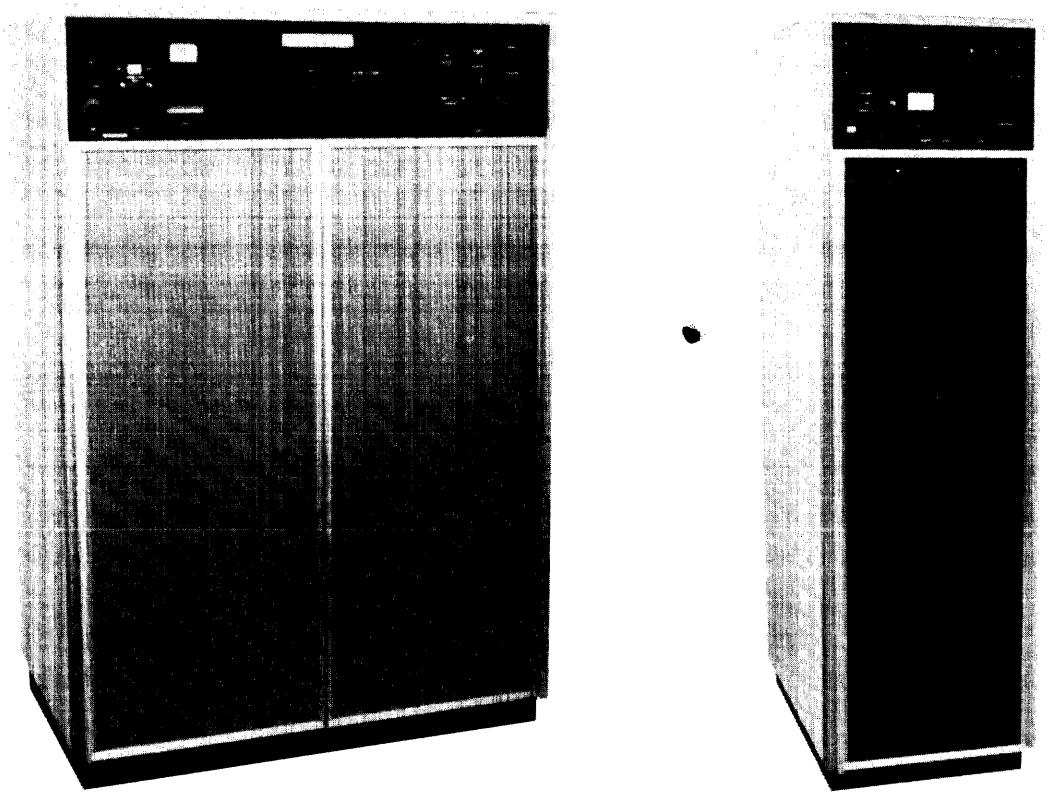


Figure 1-8. Power Distribution Units

MOTOR-GENERATORS

The Motor-generators (MGs) units convert primary power from the commercial mains to the 400 Hz power used by the computer system. They isolate the system from transients and fluctuations on the commercial power mains. The equipment consists of two or three MGs and a control cabinet. Figure 1-9 shows a Motor-generator and the control cabinet.

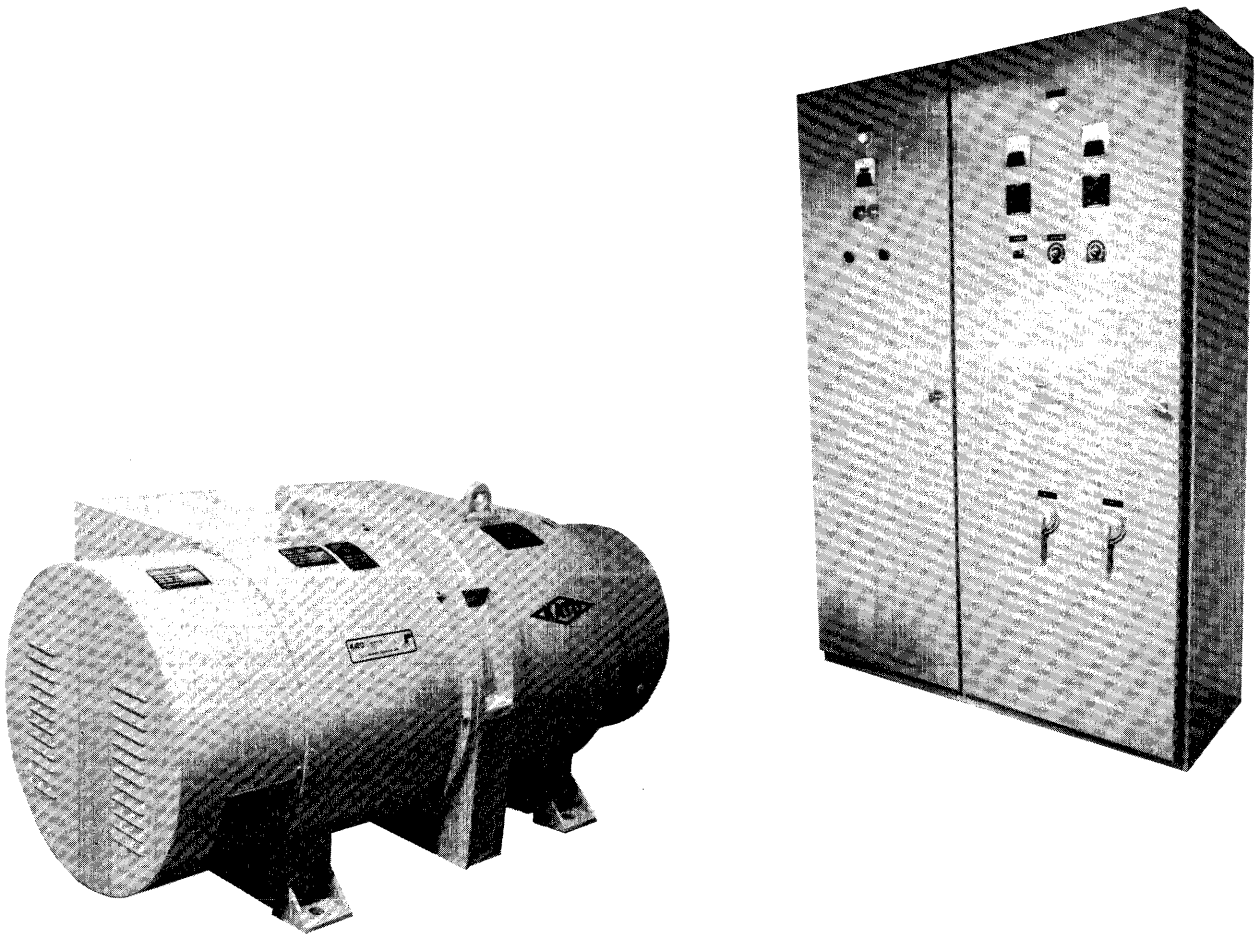


Figure 1-9. Motor-generator equipment

SYSTEM CONFIGURATION 2

INTRODUCTION

Several combinations of the basic system components are supported in the S Series of CRAY-1 Computer Systems. The Central Memory of the Central Processing Unit (CPU) is available in several different sizes. The I/O Subsystem is present on larger models in the S Series and consists of two to four processors. The following paragraphs describe the models available in the CRAY-1 S Series.

S/250, S/500, AND S/1000 MODELS

The S/500 has a 1/2-million-word Central Memory, and the S/1000 a 1-million word Central Memory. The S/250, which is a discontinued model, has a 1/4-million-word Central Memory. Figure 2-1 shows these types of configurations.

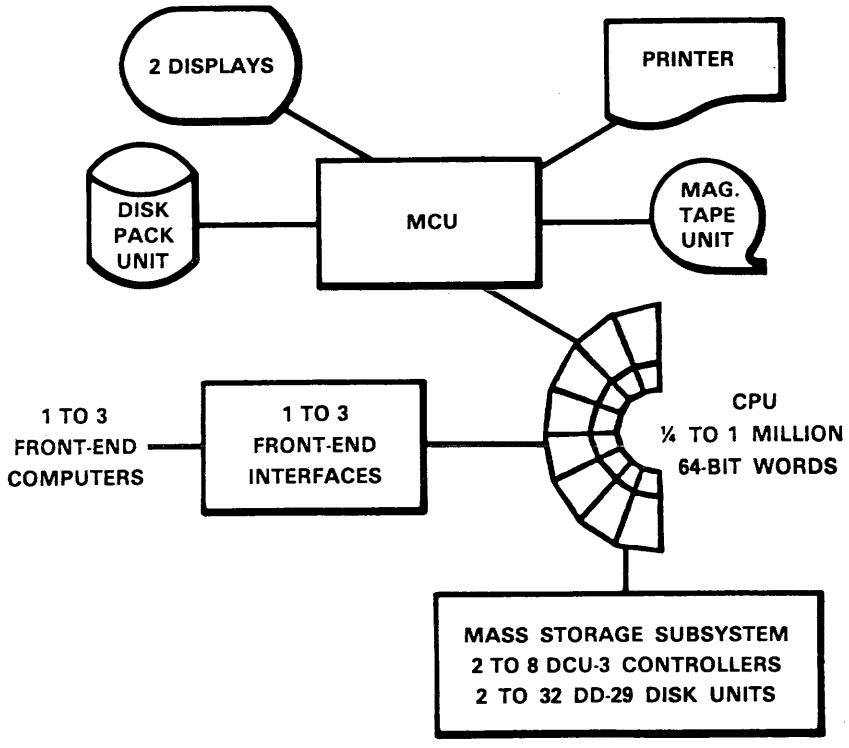


Figure 2-1. Block diagram of S/250, S/500, and S/1000 systems

All of these systems use the Maintenance Control Unit for supervision and maintenance. This consists of a minicomputer supporting the standard group of peripherals: a magnetic tape unit, a line printer, a removable pack disk drive, and two operator consoles. The mainframe of the S/250, S/500 and S/1000 models have eight chassis columns as standard. Up to three front-end computer interfaces and up to 32 disk storage units can be connected to the Central Processing Unit via the CRAY-1 I/O channels.

S/1200, S/2200, S/4200 MODELS

The S/x200 systems all have the common characteristic of a 2-processor I/O Subsystem. They differ in size of the Central Memory: S/1200 has 1 million words; S/2200 has 2 million words; and S/4200 has 4 million words. The S/1200 and S/2200 CPU chassis has 8 columns as standard. The S/4200 CPU chassis has 12 columns as standard. Figure 2-2 shows a configuration for these systems.

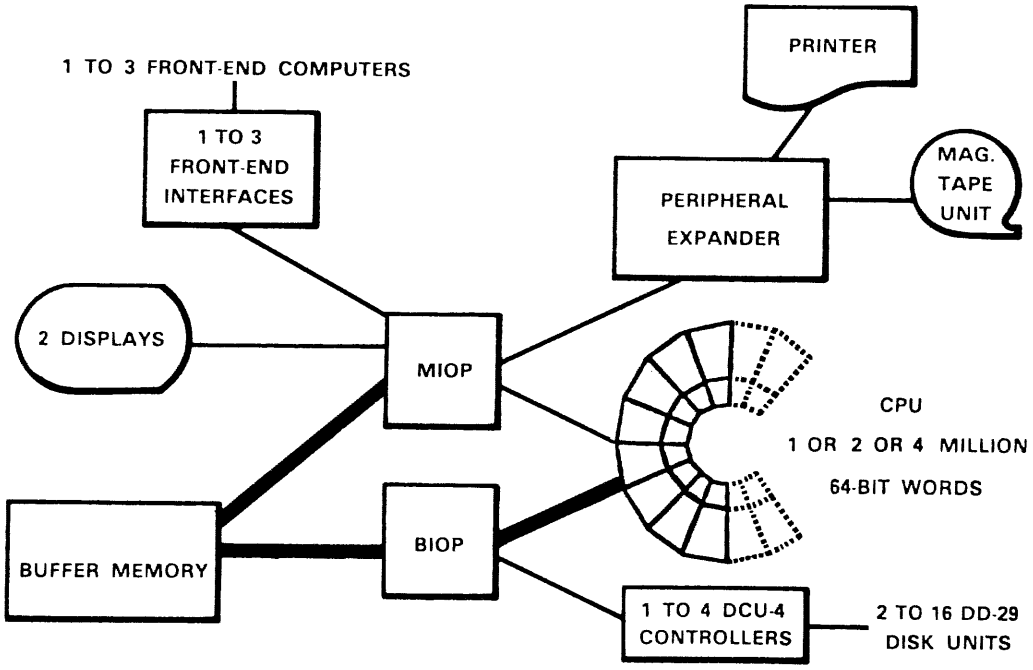


Figure 2-2. Block diagram of S/1200, S/2200, and S/4200 systems

The Master I/O Processor (MIOP) controls the front-end interfaces and the standard group of station peripherals. The Peripheral Expander interfaces the station peripherals to one DMA port of the MIOP. The Master I/O Processor also connects to the Buffer Memory and to the CPU over a CRAY-1 channel pair. Data from the front-end computer goes to Buffer Memory via the MIOP, and then to the CPU via the Buffer I/O Processor. The Master I/O Processor communicates with the CPU operating system to coordinate the activities of the I/O Subsystem.

The Buffer I/O Processor (BIOP) is the main link between the Central Processing Unit and the mass storage devices. Data from mass storage is often transferred through the BIOP I/O Memory to the Buffer Memory. Then the data is returned to the BIOP I/O Memory and sent to the CPU. The BIOP is the only I/O Processor having a standard Memory Channel to the CPU. The Memory Channel operates at speeds of approximately 800 Mbits per second. A second Memory Channel to another IOP is optional. A Model S/x200 supports up to 16 disk storage units.

S/1300, S/2300, S/4300 MODELS

The S/x300 systems all have the common characteristic of a 3-processor I/O Subsystem. They differ in size of the Central Memory: S/1300 has 1 million words; S/2300 has 2 million words; and the S/4300 has 4 million words. The S/1300 and S/2300 CPU chassis has 8 columns as standard. The S/4300 CPU chassis has 12 columns as standard. These configurations are the same as those described previously, except for the addition of a third I/O Processor. The third I/O Processor can be used either to control block multiplexer channels or to handle additional disk storage units. These two configurations are shown in figures 2-3 and 2-4, respectively.

The Auxiliary I/O Processor (XIOP) used for block multiplexer channels interfaces to a maximum of four BMC-4 Block Multiplexer Controllers, each of which can handle up to four block multiplexer channels. The XIOP uses one DMA port for each controller and another DMA port to connect with the Buffer Memory. Thus, data flow from the block multiplexer channel goes through the XIOP to the Buffer Memory, then from the Buffer Memory through the Buffer I/O Processor to the CPU.

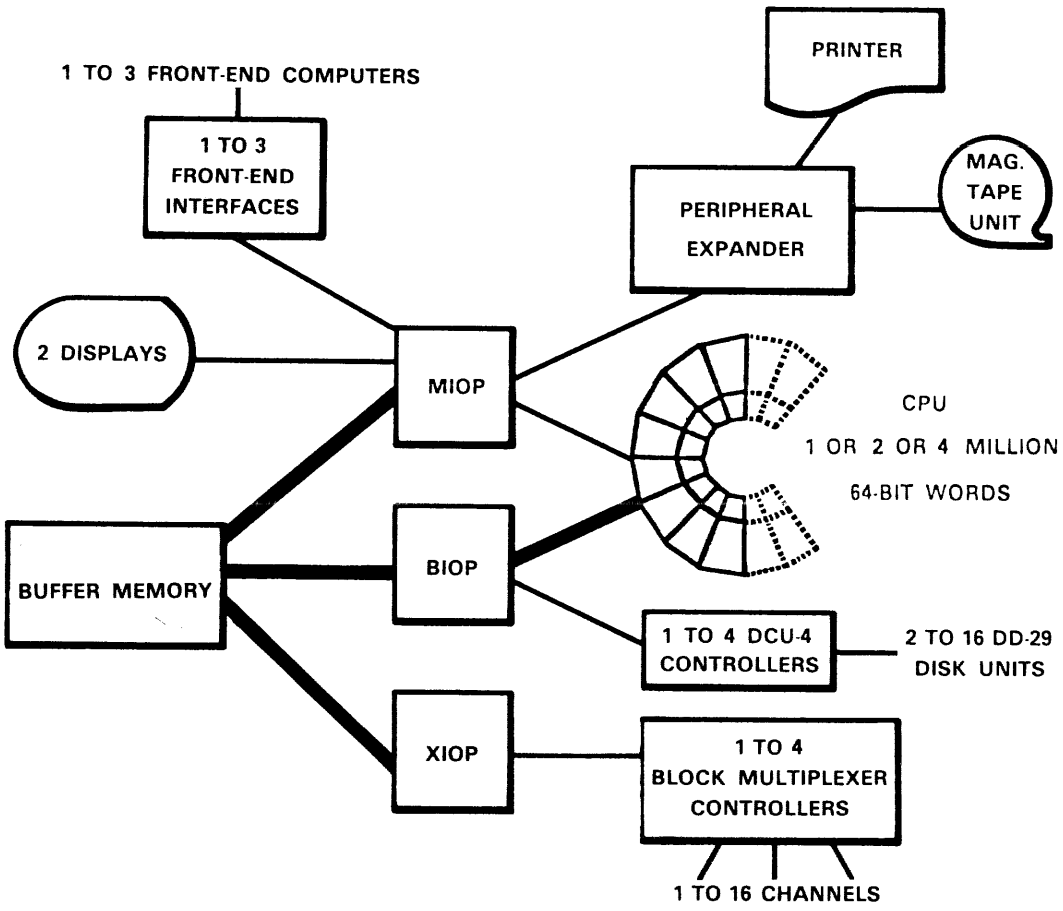


Figure 2-3. Block diagram of S/1300, S/2300, and S/4300 systems with block multiplexer channels

When the third I/O Processor is used for additional disk storage units, it is called a Disk I/O Processor, or DIOP. This processor can handle up to four disk controller units with up to 16 disk storage units. This addition effectively doubles the mass storage capacity over that of the S/X200 models--up to 32 disk storage units are possible. The disk data is transferred through the DIOP into Buffer Memory and then to the Central Processor through the Buffer I/O Processor.

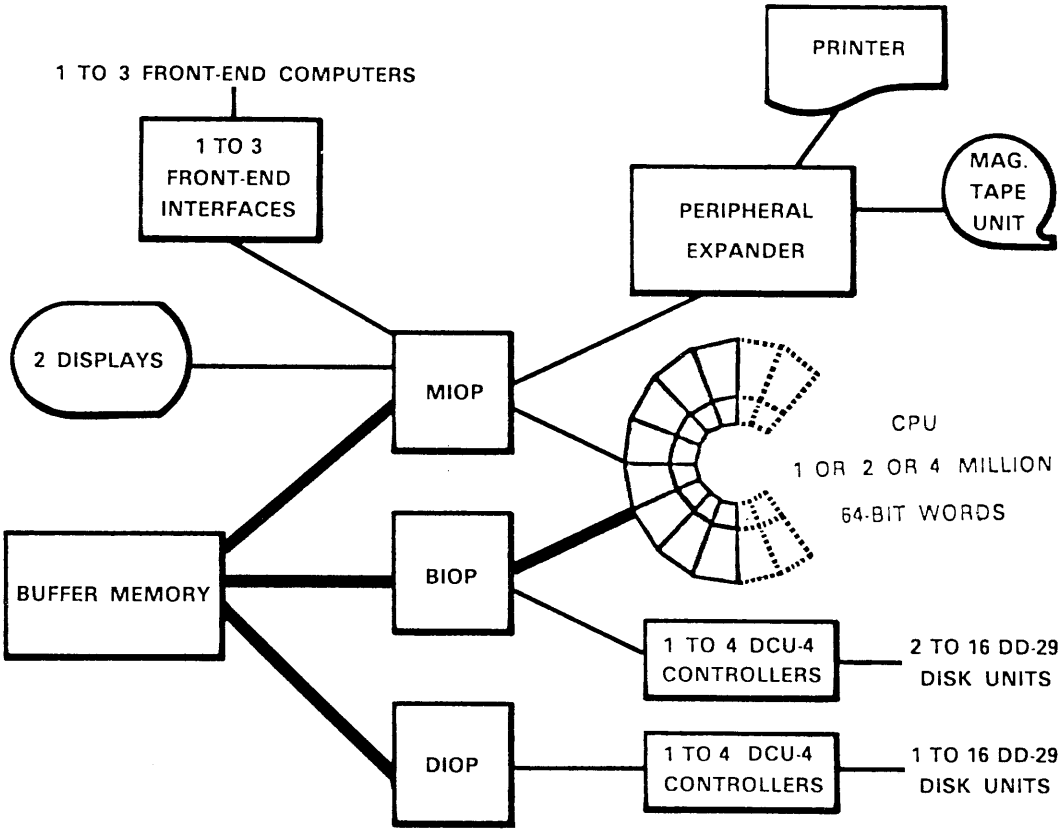


Figure 2-4. Block diagram of S/1300, S/2300, and S/4300 systems with increased disk capacity

S/1400, S/2400, S/4400 MODELS

The S/x400 systems all have the common characteristic of a 4-processor I/O Subsystem. They differ in the size of Central Memory: the S/1400 has 1 million words; the S/2400 has 2 million words; and the S/4400 has 4 million words. The S/1400 and S/2400 CPU chassis has 8 columns as standard. The S/4400 CPU chassis has 12 columns as standard. Figures 2-5 and 2-6 show these configurations. In these configurations, the third I/O Processor is assigned to Block Multiplexer Controllers or disk storage units and the fourth processor handles additional disk storage units. A MIOP and a BIOP are used as in the configurations described previously. These two configurations are shown in figure 2-5 for the block multiplexer channels, and figure 2-6 for the increased disk capacity. Each configuration is a permanent, factory-wired configuration.

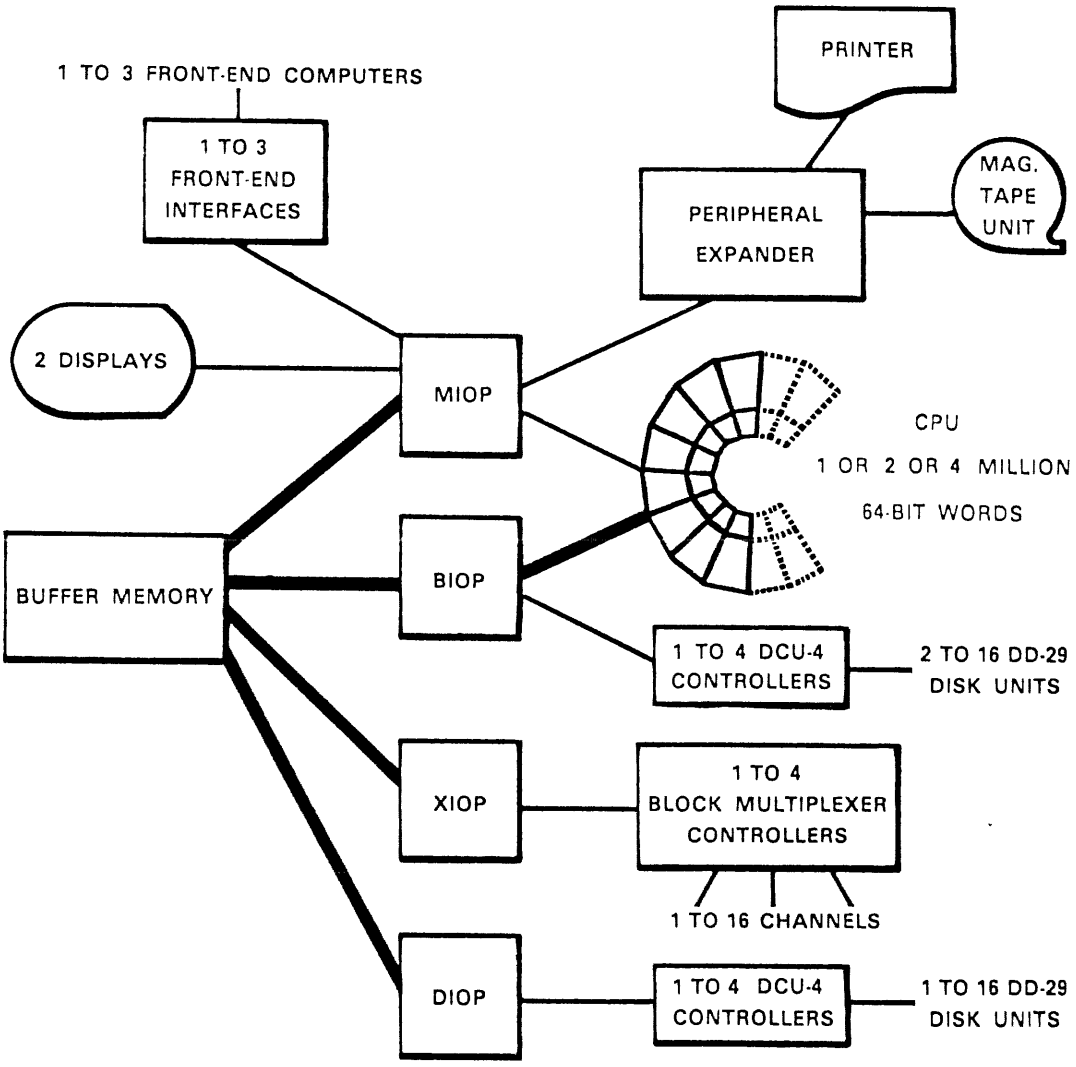


Figure 2-5. Block diagram of S/1400, S/2400, and S/4400 systems with block multiplexer channels

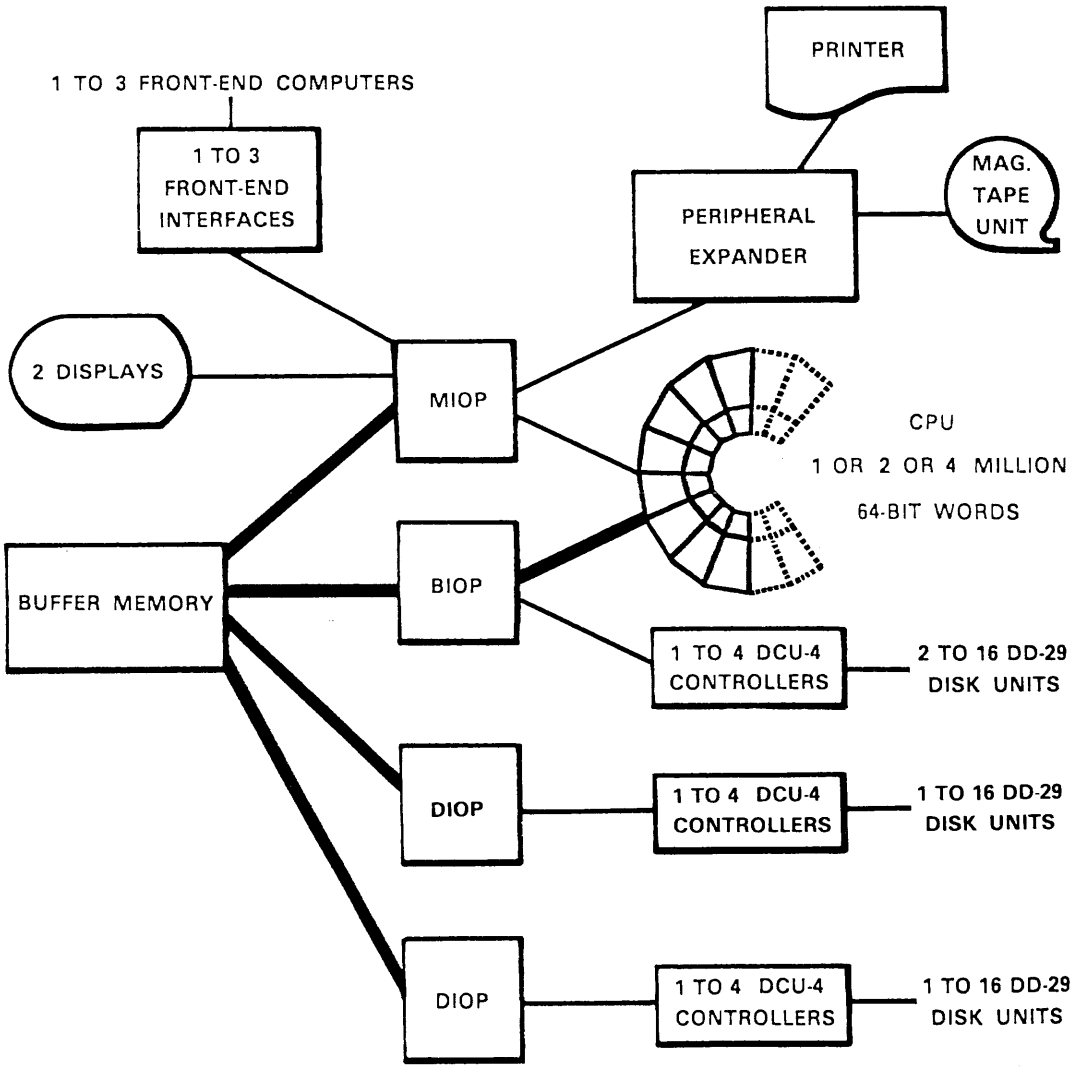


Figure 2-6. Block diagram of S/1400, S/2400, and S/4400 systems with increased disk capacity

When the third I/O Processor is used as an auxiliary I/O Processor (XIOP) for block multiplexer channels, it can handle up to 16 channels via a maximum of four Block Multiplexer Controllers. Channel data flows to the CPU via the Buffer Memory and the Buffer I/O Processor.

When the third I/O Processor is used for expanded disk capacity, it is called a Disk I/O Processor, DIOP. This configuration makes available the maximum mass storage resource. Up to 48 disk storage units can be controlled by a BIOP and two DIOPs.

MAINTENANCE CONTROL UNIT

The S/250, S/500, and S/1000 CRAY-1 Computer Systems are each equipped with a 16-bit minicomputer system that serves as a maintenance tool and provides control for the system initialization. After the CRAY-1 operating system has been initialized and is operational, communication with the Maintenance Control Unit (MCU) is via a software protocol. The MCU is connected to a CRAY-1 channel pair with additional control signals for execution of the master clear operation, I/O master clear operation, dead dump operation, and sample parity error operation. The Maintenance Control Unit includes:

1. A 16-bit minicomputer with 32K words of 16-bit memory
2. A 132-column line printer
3. An 800 bpi 9-track tape unit
4. Two display terminals
5. A removable pack disk drive

Included with the MCU system is a software package that enables it to serve as a local batch station during production hours. As a local station, it may be used to submit diagnostic routines for execution or to submit other batch jobs. These diagnostics are typically stored on the local disk and are submitted to the CRAY-1 by operator command.

The system initialization procedure is referred to in this manual as the deadstart sequence. This sequence is described in detail with the CPU control section.

Detailed information about the MCU is presented in separate publications.

INTERFACES TO FRONT-END COMPUTER

A front-end computer system is a self-contained system that executes under the control of its own operating system. The CRAY-1 computer systems is interfaced to one or more front-end computer systems that provide input data to the CRAY-1 Computer System and receive output from the CRAY-1 to be distributed to a variety of peripheral equipment. The interfaces compensate for differences in channel widths, machine word size, electrical logic levels, and control protocols. The MIOP communicates through a CRAY-1 I/O channel pair to a channel adapter module in the CPU chassis. The channel adapter module drives long cables of up to 300 feet. The interface cabinet is usually placed near the front-end computer; the CRAY-1 and the front-end computer cables connect to the interface cabinet. The standard interfaces are described in part 2, section 7.

A primary goal of the interface is to maximize the utility of the front-end channel connected to the CRAY-1. Such a channel is generally slower than CRAY-1 channels.

Peripheral equipment attached to the front-end computer varies depending on the use of the System.

A front-end computer may service the CRAY-1 in the following ways:

- As a local operator station
- As a local batch entry station
- As a data concentrator for multiplexing several other stations into a single CRAY-1 channel
- As a remote batch entry station

Detailed information about the front-end system is not presented in this publication.

SYSTEM OPERATION

The overall system consists of the components as described previously, the communication paths among them, and the software that moves the data within the devices. The following paragraphs briefly describe the system communication and job flow for systems that include an I/O Subsystem. Following that is a description of the deadstart process used to bring the system to an operational state.

I/O SUBSYSTEM COMMUNICATION

The CRAY-1 S Series system provides communication paths between the Central Processing Unit and two I/O Processors, between each I/O Processor and the Buffer Memory, and among all the I/O Processors. The arrangement is shown in figure 2-7.

Communication between the Central Processing Unit and the I/O Processors is over one approximately 50 Mbits per second CRAY-1 I/O channel pair to the Master I/O Processor, and over one approximately 850 Mbits per second Memory Channel to the Buffer I/O Processor. The CRAY-1 I/O channel pair is used for exchanging system control information with the Master IOP, while the Memory Channel transfers data through the Buffer IOP.

One DMA port of each I/O Processor is connected with the Buffer Memory through an approximately 800 Mbit/second channel. The Buffer Memory is used to organize data from one I/O Processor and store it until the Buffer I/O Processor can remove that data and pass it to the Central Processing Unit. In this way, each I/O Processor communicates with every other I/O Processor in high-speed data block transfers.

Additionally, each I/O Processor is connected with the other I/O Processors by slower channels called "accumulator channels". These channels pass one 16-bit parcel at a time from the accumulator of one I/O Processor to the accumulator of another I/O Processor. These are used primarily for control and status reporting.

Any errors occurring in the system memories or Memory Channel are reported to the Master I/O Processor via special error channels that are separate from the data channels. Thus all error handling for the system is initiated by the Master I/O Processor.

The resulting communications network among the processors speeds the flow of data from the front-end computers, peripheral devices and mass storage units; it stores the data as necessary; and it passes the data to the CPU. The network also facilitates transfer of results from the CPU to the final destination.

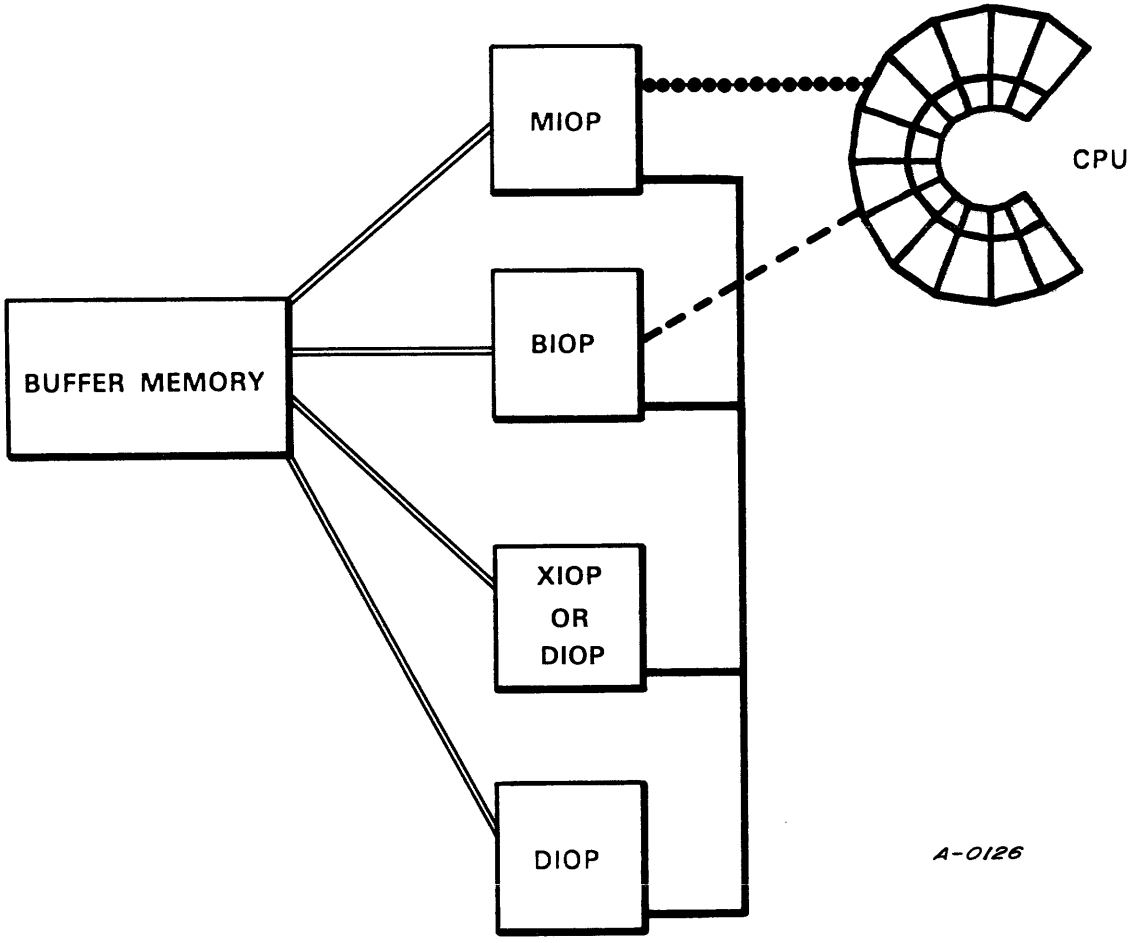


Figure 2-7. I/O Subsystem communication

JOB FLOW

Figure 2-8 shows a simplified view of the job flow within the system; numbers index the sequence of operations. Jobs originate in the front-end computer network, with the front-end computer passing (1) job control statements, programs, and data files to the Master I/O Processor. The Master I/O Processor transfers the job (2) to the Buffer Memory. The MIOP informs the CRAY-1 Operating System (3) of the existence of data in the Buffer Memory. The CRAY-1 Operating System stores data by a request (4) to the MIOP to have the DIOP store the data. The MIOP commands the Disk I/O Processor (5) to transfer data from the Buffer Memory (6) to mass storage (7).

The operating system analyzes the resources required for the job, and when these are available, calls the Master I/O Processor for the job (8). The MIOP tells the DIOP (9) to bring the job in from mass storage (10) and store it in Buffer Memory (11) for transfer to the Buffer IOP. As soon as possible, the Buffer IOP begins transferring the job from Buffer Memory (12) to its own I/O Memory and then to the Central Processing Unit (13) over the Memory Channel. When the transfer is complete, the CPU begins executing the job control statements.

Several transfers to and from mass storage may be necessary during job execution. Output from the job is sent to mass storage (14 - 20), and when the job is finished, the MIOP is notified. The Master I/O Processor tells the Disk I/O Processor (21) to bring in the results from mass storage (22) and place them in Buffer Memory (23). The operating system tells the MIOP (24) to take data from the Buffer Memory (25) and pass it to the front-end computer (26).

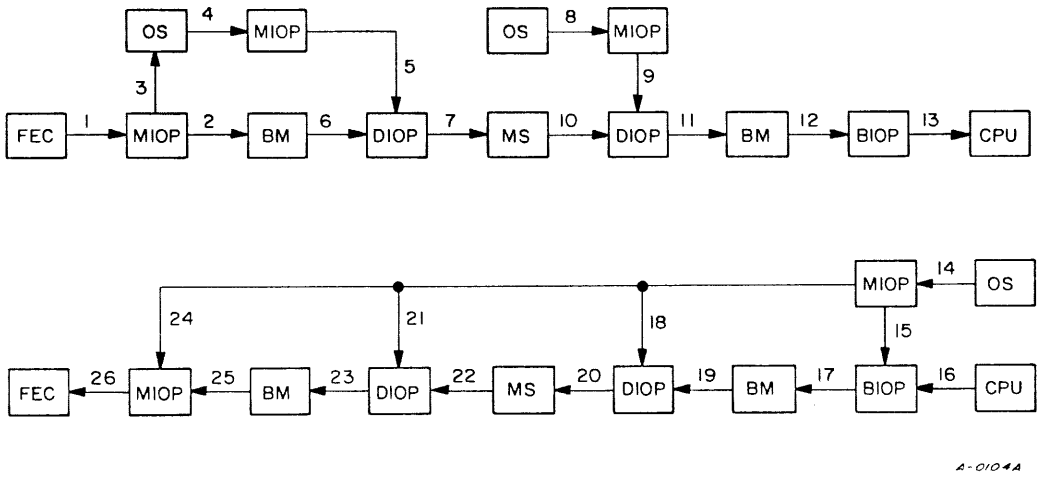


Figure 2-8. Job flow diagram

DEADSTART

A model S/1200 through S/4400 system is initially started from a magnetic tape bootstrap which loads into the memory of the Master I/O Processor. This program is enough to load further software from the tape and store it in Buffer Memory. When the Buffer Memory has the deadstart program, the Master I/O Processor commands the Buffer I/O Processor to deadstart from Buffer Memory. The Buffer I/O Processor loads the program which enables it to load further program data from a disk storage unit. The disk data is passed through the Buffer Memory to the Master I/O Processor which can completely initiate the I/O Subsystem. Then, under control of the Master I/O Processor, the main portion of the operating system is loaded into the Central Memory and the system becomes operational.

Initial installation of the deadstart program and operating system on the disk storage unit is done by maintenance personnel from magnetic tape. In the case of a failure in the Master I/O Processor, a maintenance deadstart panel can be used to load a deadstart or diagnostic program into the Master or Buffer I/O Processor.

For a Model S/250, S/500, or S/1000 system using a Maintenance Control Unit, deadstarting occurs from the removable-pack disk storage unit through the minicomputer. The main operating system is then loaded from the same disk storage unit into Central Memory. This operation is described with the control section of the Central Processing Unit.

PART 2

CENTRAL PROCESSING UNIT

GENERAL INFORMATION 1

INTRODUCTION

The Central Processing Unit (CPU) is the computer that executes programs, runs user jobs, and oversees the job flow within the CRAY-1 S Series Computer System. Scalar and vector processing capabilities are combined with large, fast Central Memory and high-volume I/O channels. Figure 1-1 represents the basic organization of the Central Processing Unit. The memory is expandable from one-quarter million to four million words of 64-bits each. The Memory Channels are used for high-speed data transfers to and from I/O Processors in the I/O Subsystem. Twelve I/O channel pairs provide access to front-end computers, mass storage controllers, and to the I/O Subsystem.

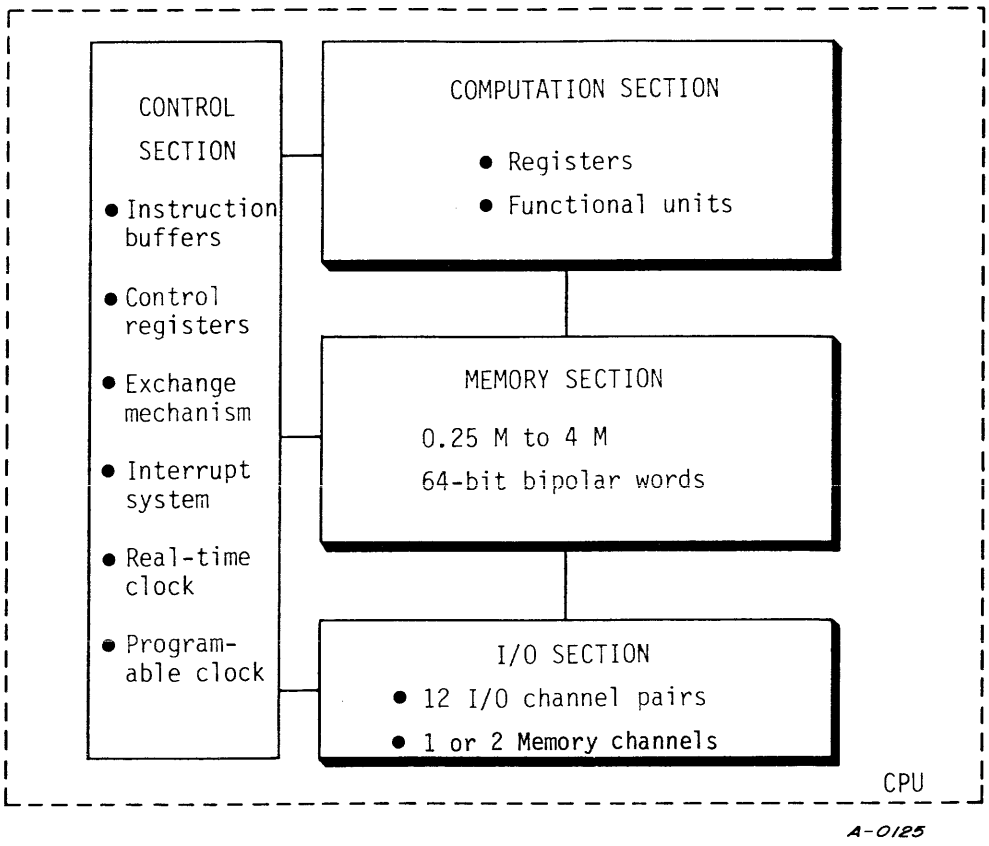


Figure 1-1. Basic organization of the CPU

The following paragraphs provide additional general information about the four sections of the CPU; later sections describe the features in greater detail.

Figure 1-2 illustrates the components of the CPU and presents a generalized view of the flow of data in the system.

REGISTER CONVENTIONS

Frequent use is made in this manual of parenthesized register names. This is shorthand notation for the expression "the contents of register ---." For example, "Branch to (P)" means "Branch to the address indicated by the contents of the program parcel counter, P."

Extensive use is also made of subscripted designations for the A, B, S, T, and V registers. For example, "Transmit (Tjk) to Si" means "Transmit the contents of the T register specified by the jk designators to the S register specified by the i designator."

In this manual, register bit positions are numbered from right to left as powers of 2, starting with bit 2^0. Bit 2^{63} of an S, V, or T register value represents the most significant bit in the operand. Bit 2^{23} of an A or B register value represents the most significant bit in the operand.

NUMBER CONVENTIONS

Unless otherwise indicated, numbers in this manual are decimal numbers. Octal numbers are indicated with an 8 subscript. Exceptions are register numbers, channel numbers, and instruction forms, which are given in octal without the subscript.

CLOCK PERIOD

The basic unit of CPU computation time is 12.5 nanoseconds and is referred to as a clock period (CP). Instruction issue, memory references, and other timing considerations are often measured in clock periods.

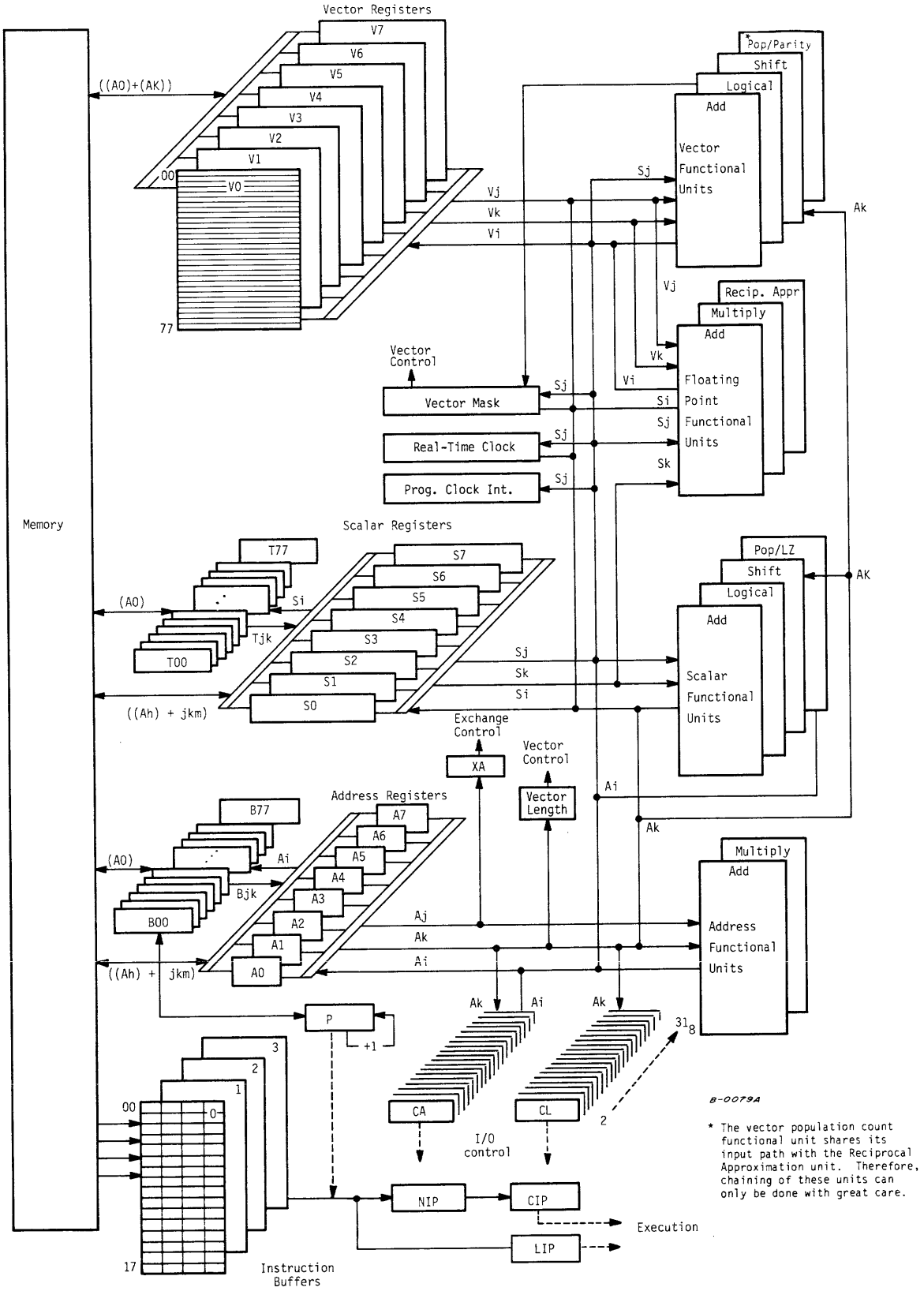


Figure 1-2. Control and data paths in the CPU

MEMORY SECTION

The memory for the CRAY-1 consists of 8 or 16 banks of bipolar LSI memory. Four memory size options are available: 524,288 or 1,048,576 or 2,097,152 or 4,194,304 words. A 226,144-word memory is no longer available. Each word is 72 bits--64 data bits and 8 check bits. The banks are independent of each other.

Sequentially addressed words reside in sequential banks. The memory cycle time is 4 CPs (50 ns). The access time, that is, the time required to fetch an operand from memory to an operational register, is 11 CPs (137.5 ns). There is no inherent memory speed degradation for 16-bank memories of less than 4 million words.

The maximum transfer rate for B, T, and V registers is one word per CP. For A and S registers, it is one word per 2 CPs. Transfer of instructions to the instruction buffers occurs at a rate of 16 parcels (four words) per CP.

Thus, the CPU memory sizes support the requirements of scientific applications, while the low cycle time is well suited to random access applications. The phased memory banks allow high communication rates through the I/O section and provide low read/store times for vector registers.

Table 1-1 summarizes the features of the CPU memory section. CPU Memory is described in detail in section 2.

Table 1-1. Characteristics of the CPU memory section

- From 0.25M to 4M words of bipolar integrated circuit memory
- 64 data bits and 8 error correction bits per word
- 8 or 16 interleaved banks
- 4 CP bank cycle time
- 1 word per CP transfer rate to B, T, and V registers
- 1 word per 2 CP transfer rate to A and S registers
- 4 words per CP transfer rate to instruction buffers
- Single error correction/double error detection (SECDED)

CONTROL SECTION

The control section performs all decisions related to instruction issue and coordinates the activities for the three types of processing: address, scalar, and vector.

The control section executes 128 basic instruction codes as either 16-bit (1 parcel) or 32-bit (2 parcel) instructions and provides for register reservation, memory field protection, memory access, and interrupt control.

Table 1-2 summarizes features of the control section. This section is described in greater detail in section 3.

Table 1-2. Characteristics of the CPU control section

- 12.5 nanosecond clock period (CP)
- 4 instruction buffers of 64 16-bit parcels each
- 128 basic instruction codes
- Program exchange mechanism
- Error/monitor interrupt flags
- Memory and program field protection

COMPUTATION SECTION

The computation section contains registers and functional units that operate together to execute a program of instructions stored in memory.

Eight address (A) registers are used to store 24-bit integers or addresses. Sixty-four intermediate address (B) registers store data for use by the A registers. Eight scalar (S) registers store 64-bit operands for scalar operations. Sixty-four intermediate (T) registers store data temporarily for the S registers. Eight vector (V) registers are made up of 64 elements in each register. Each element stores one 64-bit operand. The vector registers are used in vector processing.

A series of operands can be an ordered set, called a vector. A vector instruction operates on a series of operands, doing the same function repetitively, and producing a series of results. Scalar processing starts an instruction, handles one operand or operand pair, then stops the operation. The main advantage of vector over scalar processing is the elimination of the instruction start-up time for all but the first operand.

Arithmetic operations are integer or floating-point. Integer arithmetic is performed in twos complement mode. Floating-point quantities have signed magnitude representation.

Floating-point instructions provide for addition, subtraction, multiplication, and reciprocal approximation. The reciprocal approximation instructions allow a floating-point divide operation using a multiple instruction sequence. These instructions produce 64-bit results.

Integer or fixed-point operations are provided as follows: integer addition, integer subtraction, and integer multiplication. An integer multiply operation produces a 24-bit result; additions and subtractions produce either 24-bit or 64-bit results. No integer divide instruction is provided; the operation is accomplished through a software algorithm using floating-point hardware.

The instruction set includes Boolean operations for OR, AND, equivalence, and exclusive OR and for a mask-controlled merge operation. Shift operations allow the manipulation of either 64-bit or 128-bit operands to produce 64-bit results. With the exception of 24-bit integer arithmetic, most operations are implemented in vector as well as scalar instructions. The integer product is a scalar instruction designed for index calculation. Full indexing capability allows the programmer to index throughout memory in either scalar or vector modes. The index may be positive or negative in either mode. This allows matrix operations in vector mode to be performed on rows or the diagonal as well as conventional column-oriented operations.

Population and parity counts are provided for both vector and scalar operations. Additionally, scalar operations may include leading zero counts.

Table 1-3 summarizes the characteristics of the CPU computation section.

Table 1-3. Characteristics of CPU computation section

- Integer and floating-point arithmetic
- Twos complement integer arithmetic
- Sign and magnitude floating-point arithmetic
- Address, scalar, and vector processing modes
- Thirteen functional units
- Eight 24-bit address (A) registers
- Sixty-four 24-bit intermediate address (B) registers
- Eight 64-bit scalar (S) registers
- Sixty-four 64-bit intermediate scalar (T) registers
- Eight 64-element vector (V) registers, 64 bits per element

INPUT/OUTPUT SECTION

One or two Memory Channels transfer data between Central Memory and the Buffer I/O Processor. Each channel is 64 bits wide and uses 8 check bits with each word. Data words are transferred in blocks of 16 under control of Data Ready and Data Transmit control signals. A maximum transfer rate of approximately 850 Mbits per second is possible on this channel.

Normal input and output communication with the CPU is over 12 full duplex 16-bit channel pairs. Associated with each channel are control lines that indicate the presence of data on the channel.

Table 1-4 summarizes features of the channels in the CPU input/output section. Channels are described in detail in section 5.

Table 1-4. Characteristics of the CPU input/output section

Up to twelve I/O channel pairs; 50 or 160 Mbits/s maximum rate
- Four channel groups containing either 6 input or 6 output channels
- Channel groups served equally by memory (scans each group every 4 CPs)
- Channel priority resolved within channel groups
- 16 data bits, 3 control bits, and 4 parity bits in each direction
- Lost data detection
One or two Memory Channels; approx. 800 Mbits/s maximum rate each
- 64 data bits, 3 control bits, and 8 check bits in each direction

CENTRAL MEMORY SECTION 2

INTRODUCTION

The Central Memory of the CPU consists of 8 or 16 banks of bipolar LSI memory. Four memory sizes are available:

524,288 words - 8 banks,
1,048,576 words - 8 banks,
2,097,152 words - 8 banks,
4,194,304 words - 16 banks.

The banks are independent of each other. A 262,144-word 8-bank memory is no longer available.

MEMORY CYCLE TIME

The memory cycle time is 4 CPs (50 ns). The access time, that is, the time required to fetch an operand from memory to an operational register, is 11 CPs (137.5 ns).

MEMORY ACCESS

The Central Memory of the CPU is shared by the computation section and the I/O section. A single port access is provided.

Because of the interleaving scheme used to address the independent banks, it is possible to reference memory every clock period with a new request. It is not possible, however, to reference any one bank sooner than its 4-CP cycle time. Trying to reference a bank more often than every 4 CPs causes memory conflicts. These conflicts are handled in an orderly, predictable manner.

Block transfers require all memory requests to be completed before the block transfers can issue. Once issued, they inhibit all other memory requests. Multiple block transfers cannot issue without allowing one waiting I/O reference to complete. The maximum duration of a lockout caused by block transfers is one block length.

Vector block transfers may conflict with themselves. Therefore, the vector logic provides for identifying these conditions (speed control) and for slowing or disallowing the vector operations that would be affected by the slowed memory referencing rate. The vector logic identifies 1/4 speed (4 CPs), 1/2 speed (2 CPs), and full speed (1 CP) data rates from memory.

Fetch operations bring instructions from Central Memory to the instruction buffers. Fetch operations require completion of all other types of memory references before the fetch operations reference memory. Once the fetch request is honored, all other types of memory reference are inhibited.

Exchange operations require memory to be quiet before referencing memory. After the exchange has issued, all other memory references are inhibited.

Scalar and I/O memory references are examined in three registers for possible memory conflicts. These three registers contain the low-order bits of each of the referenced memory addresses. These registers, plus the address register, represent the 4 CPs between referencing any one bank. The first register is rank A, the second is rank B, and the third is rank C. At each clock period, the contents of the registers are shifted down one rank until they are discarded unless a conflict arises, in which case the conflicting address is held in rank B until the conflict is resolved.

I/O requests are tested against ranks A, B, and C. Coincidence with rank A, B, or C disallows the request. An I/O request that is disallowed must wait 8 CPs before it can request again.

The following conditions must be present for an I/O memory request to be processed:

1. I/O request
2. No coincidence in rank A, B, or C
3. No scalar memory reference in CP 2 of its sequence (scalar priority over I/O)
4. No fetch request
5. No 176, 177, or 034 through 037 instruction in progress
6. No exchange sequence or request
7. No 033 request (not a memory conflict)

Scalar instruction memory requests are tested in ranks A, B, and C for memory conflicts. Scalar instructions have priority over I/O requests arriving in memory in the same clock period.

A scalar conflict in rank A (CP 2 of a scalar instruction) causes a hold storage on this instruction for 3 CPs. At the same time, a hold issue signal blocks the issue of another scalar reference instruction. The only memory conflict that may occur in rank A is a scalar reference conflicting with a previous I/O reference. It is not possible for a scalar to conflict with a scalar in rank A because it takes 2 CPs to issue a scalar reference instruction.

A scalar conflict in rank B (CP 3) causes a hold storage on this instruction for 2 CPs. Also, a hold issue signal blocks issue of another scalar reference instruction.

A scalar conflict in rank C (CP 4) causes a hold storage on this instruction for 1 CP. There is also a hold issue signal, which blocks issue of another scalar reference instruction.

The Memory Channel shares the same access with the normal I/O channels, but the normal I/O channels have priority. The Memory Channel operates in blocks of 16 words with a 1-CP pause between blocks to allow other memory operations to break the Memory Channel transfer.

Under normal operating conditions on codes performing a mix of vector and scalar instructions, the memory access supports four disk and three interface channel pairs without degrading the CPU computation rate. However, a single program requiring memory access continuously will be measurably degraded by maximum I/O transfer conditions. This degradation is caused by the delays imposed on the issue of vector memory instructions because block transfers require memory quiet before issue.

MEMORY ORGANIZATION

Central Memory is organized into 8 or 16 banks to minimize memory conflicts and to exploit the speed of the memory chips. In a 16-bank machine, each bank occupies half a column and contains 72 modules. Each module contributes one data or check bit to each 72-bit word in the bank; a memory word consists of 64 data bits and 8 check bits.

The 8-bank organization is standard on the 8-column mainframes. The 16-bank phasing is standard on the S/4200 and larger mainframes. A maintenance feature for all 16-bank systems permits them to operate with only 8 banks using either the left or right half of memory. This is accomplished by installing two special modules and setting the bank select switch (on the Power Distribution Unit) to the left or right banks.

MEMORY ADDRESSING

A word in a 16-bank memory is addressed in a maximum of 22 bits as shown in figure 2-1. The low-order 4 bits specify one of the 16 banks. The next field specifies an address within the chip. The high-order bits specify one of the chips on the module.

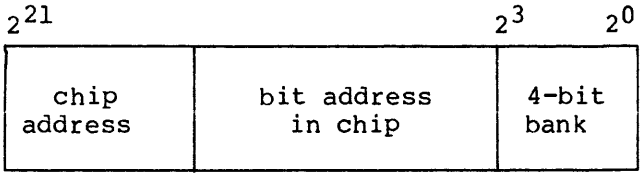


Figure 2-1. Memory address (16 banks)

A word in an 8-bank memory is addressed in a maximum of 21 bits as shown in figure 2-2. In this case, the low-order 3 bits specify one of the eight banks. The next field specifies an address within the chip. The high-order bits specify one of the chips on the module.

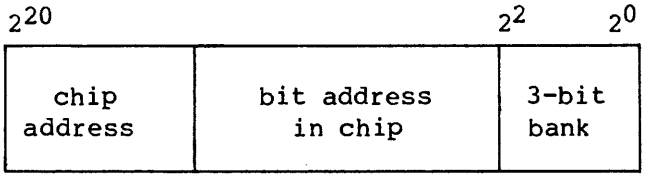


Figure 2-2. Memory address (8 banks)

SPEED CONTROL

For vector read and vector store instructions, the low-order 4 bits of (Ak) determines speed control (table 2-1).

Table 2-1. Vector memory rate x 80 x 10^6 references per second

Phasing	Increment or multiple in (Ak)							
	1-3	4	5-7	8	9-11	12	13-15	16
8-bank	1	1/2	1	1/4	1	1/2	1	1/4
16-bank	1	1	1	1/2	1	1	1	1/4

For eight banks, incrementing by eight places causes successive references in the same bank so that a word is transferred every 4 CPs. If (Ak) is incremented by 4, an 8-bank memory transfers words every 2 CPs.

8-BANK PHASING

A 16-bank system can be readily modified to run on either the right or left 8 banks for maintenance purposes. This is accomplished by replacing two modules and setting the bank select switch on the Power Distribution Unit to the right or left banks. If the situation warrants it, the machine can continue running on one half of memory while repairs are made to the other half.

The effect of 8-bank phasing on instruction fetches is a predictable increase of 4 CPs for filling an instruction buffer. Otherwise, the amount of performance degradation for 8 banks as compared with 16 banks is not readily predictable since it largely results from an increase of memory conflicts.

For other differences, refer to the preceding paragraphs on Memory Addressing and Speed Control.

MEMORY ERROR CORRECTION

An error correction and detection network between the CPU and memory assures that the data written into memory can be returned to the CPU with consistent precision (figure 2-3).

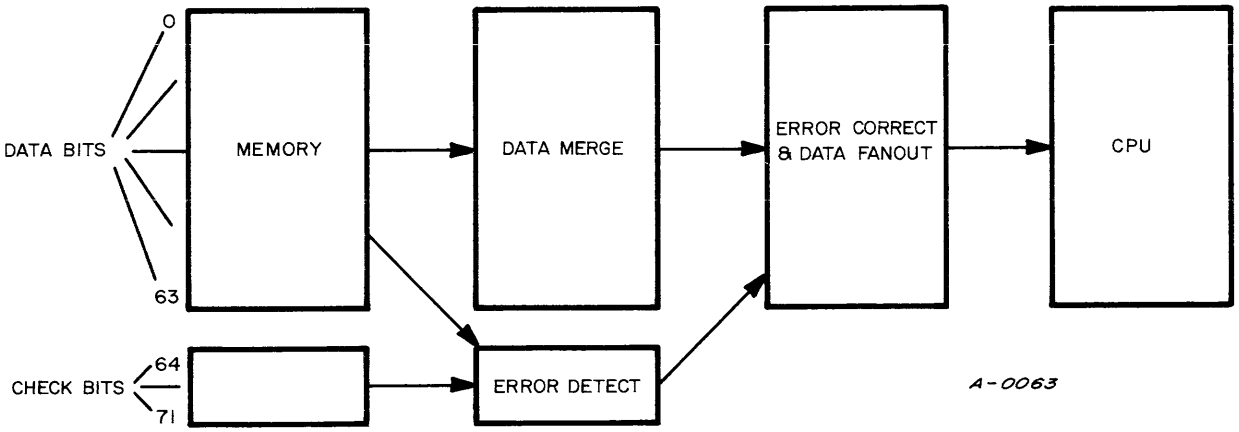


Figure 2-3. Memory data path with SECDED

The network operates on the basis of single error correction, double error detection (SECDED). If 1 bit of a data word is altered, the single error alteration is automatically corrected before passing the data word to the computer. If 2 bits of the same data word are altered, the double error is detected but not corrected. In either case, the CPU may be interrupted depending on interrupt options selected to prevent incorrect data from contaminating a job. For 3 or more bits in error, results are ambiguous.

The SECDED error processing scheme is based on error detection and correction codes devised by R. W. Hamming[§]. An 8-bit check byte is appended to the 64-bit data word before the data is written in memory. The 8 check bits are each generated as even parity bits for a specific group of data bits. Figure 2-4 shows the bits of the data word used to determine the state of each check bit. An X in the horizontal row indicates that data bit contributes to the generation of that check bit. Thus, check bit 2^{64} is the bit making group parity even for the group of bits 2^{1}, 2^{3}, 2^{5}, 2^{7}, 2^{9}, 2^{11}, 2^{13}, 2^{15}, 2^{17}, 2^{19}, 2^{21}, 2^{23}, 2^{25}, 2^{27}, 2^{29}, and 2^{31} through 2^{55}.

The 8 check bits are stored in memory at the same location as the data word. When read from memory, the same 72-bit matrix of figure 2-4 is used to generate a new set of parity bits, which are even parity bits of the data word, and the old check bits. The resulting 8 parity bits are called syndrome[§§] bits (S bits), shown as bits 64 through 71 in figure 2-4. The states of these S bits are all symptoms of any error that occurred. If all syndrome bits are 0, no memory error occurred.

Any change of state of 1 data bit will cause an odd number of syndrome bits to be set to 1. An error in two columns changes the parity states of an even number of bit groups. Therefore, a double error appears as an even number of syndrome bits set to 1.

The matrix is designed so that SECDED decodes the syndrome bits and determines the error condition using the following five rules:

1. If all syndrome bits are 0, no error occurred.
2. If only 1 syndrome bit is 1, the associated check bit is in error.

---

§ Hamming, R.W., "Error Detection and Correcting Codes", Bell System Technical Journal, 29, No. 2, pp. 147-160 (April, 1950).

§§ Syndrome: Any set of characteristics regarded as identifying a certain type, condition, etc. Websters New World Dictionary.

3. If more than 1 syndrome bit is 1 and the parity of all syndrome bits S0 through S7 is even, then a double error (or an even number of bit errors) occurred within the data bits or check bits.

4. If more than 1 syndrome bit is 1 and the parity of all syndrome bits is odd, then a single and correctable error is assumed to have occurred. The syndrome bits can be decoded to identify the bit in error.

5. If 3 or more memory bits are in error, the parity of all syndrome bits is odd and results are ambiguous.

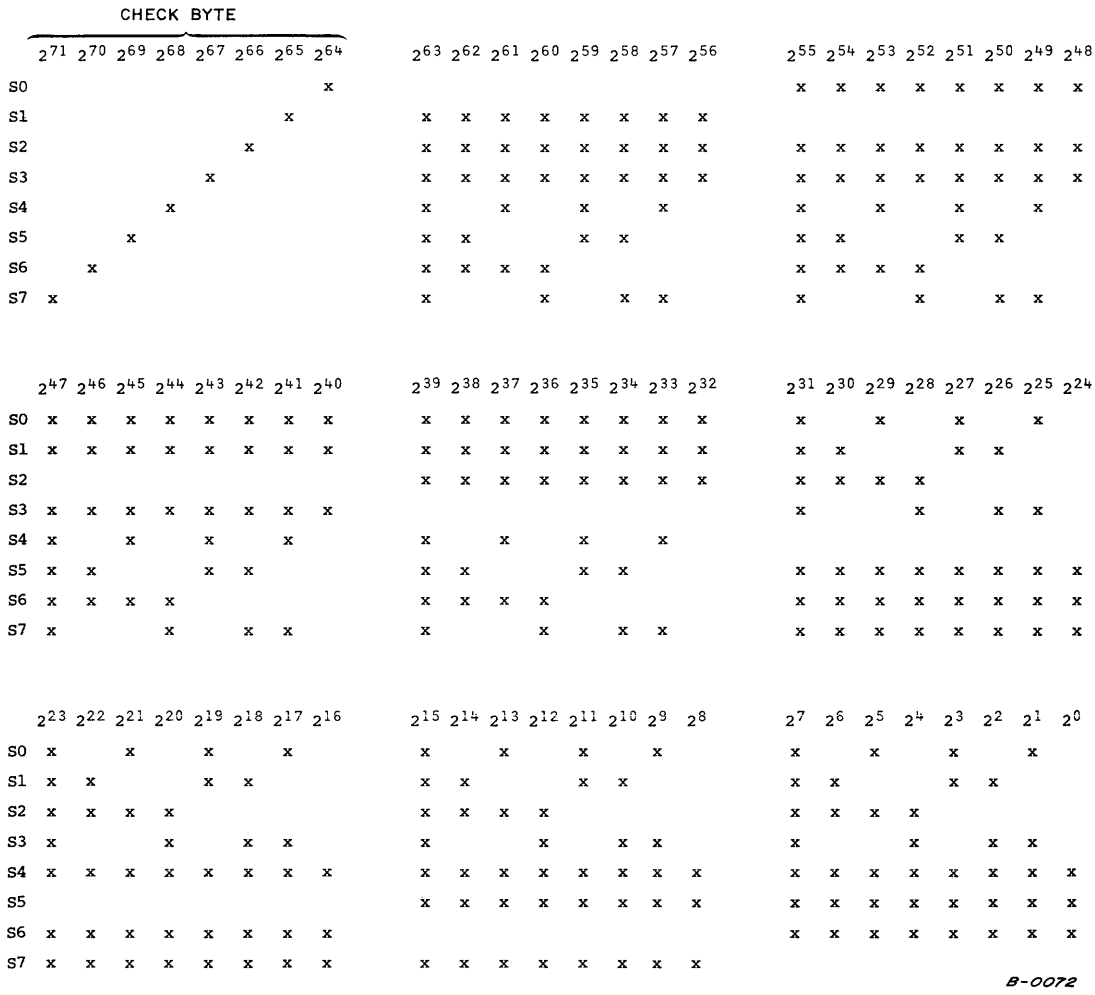


Figure 2-4. Error correction matrix

CPU CONTROL SECTION 3

INSTRUCTION ISSUE AND CONTROL

This section describes the instruction buffers and registers involved with instruction issue and control. Figure 3-1 illustrates the general flow of instruction parcels through the registers and buffers.

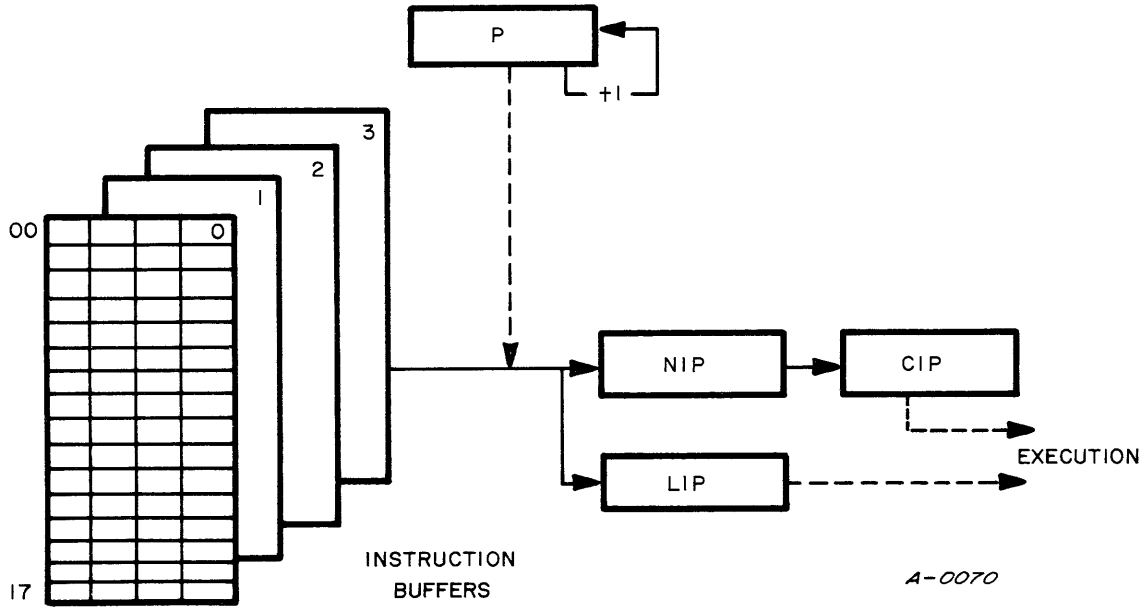


Figure 3-1. Instruction issue and control elements

P REGISTER

The P register is a 24-bit register which indicates the next parcel of program code to enter the next instruction parcel (NIP) register in a linear program sequence. The high-order 22 bits of the P register indicate the word address for the program word in memory. The low-order 2 bits indicate the parcel within the word. The contents of the P register are normally advanced by 1 as each parcel successfully enters the NIP register. The value in the P register normally corresponds to the parcel address for the parcel currently moving to the NIP register.

The P register is entered with new data on an instruction branch or on an exchange sequence. The contents of P are then advanced sequentially until the next branch or exchange sequence. The value in the P register is stored directly into the terminating exchange package during an exchange sequence.

The P register is not master cleared. An indeterminate value is stored in the terminating exchange package at address 0 during the deadstart sequence.

NIP REGISTER

The NIP (next instruction parcel) register is a 16-bit register that holds a parcel of program code prior to entering the CIP register. A parcel of program code that has entered the NIP register must be executed. There is no mechanism to discard it.

The NIP register is not master cleared. An undetermined instruction may issue during the master clear interval before the interrupt condition blocks data entry into the NIP register.

CIP REGISTER

The CIP (current instruction parcel) register is a 16-bit register that holds the instruction waiting to issue. If this instruction is a 2-parcel instruction, the CIP register holds the upper half of the instruction and the LIP holds the lower half. Once an instruction enters the CIP register, it must issue. Issue may be delayed until previous operations have been completed but then the current instruction waiting for issue must proceed. Data arrives at the CIP register from the NIP register. The indicators which make up the instruction are distributed to all modules which have mode selection requirements when the instruction issues.

The control flags associated with the CIP register are generally master cleared; the register itself is not. An undetermined instruction will issue during the master clear sequence.

LIP REGISTER

The LIP (lower instruction parcel) register is a 16-bit register that holds the lower half of a 2-parcel instruction at the time the 2-parcel instruction issues from the CIP register.

INSTRUCTION BUFFERS

The CPU has four instruction buffers, each of which holds 64 consecutive 16-bit instruction parcels (figure 3-2). Instruction parcels are held in the buffers prior to being delivered to the NIP or LIP registers.

The beginning instruction parcel in a buffer always has a word address that is a multiple of 20_8, (that is, a parcel address that is a multiple of 100_8). This allows the entire range of addresses for instructions in a buffer to be defined by the high-order 18 bits of the beginning parcel address. For each buffer, there is an 18-bit beginning address register that contains this value.

The beginning address registers are scanned each clock period. If the high-order 18 bits of the P register match one of the beginning addresses, an in-buffer condition exists and the proper instruction parcel is selected from the instruction buffer. An instruction parcel to be executed is normally sent to the NIP. However, the second half of a 2-parcel instruction is blocked from entering the NIP and is sent to the LIP instead, and is available when the upper half issues from the CIP. At the same time, a blank parcel is entered into the NIP.

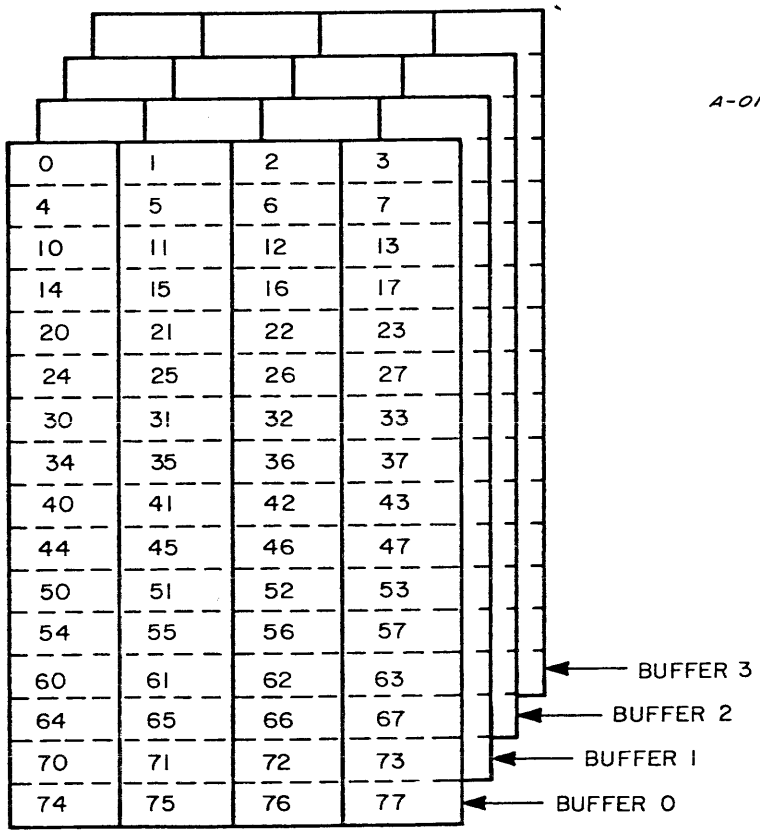


Figure 3-2 Instruction buffers

On an in-buffer conditon, if the instruction is in a different buffer than the previous instruction, a change of buffers occurs, normally necessitating a 2-CP delay of issue.

An out-of-buffer condition exists when the high-order 18 bits of the P register do not match any instruction buffer beginning address. When this conditon occurs, instructions must be loaded into one of the instruction buffers from memory before execution can continue. The instruction buffer that receives the instructions is determined by a 2-bit counter. Each occurrence of an out-of-buffer condition causes the counter to be incremented by 1 so that the buffers are selected in rotation.

Buffers are loaded from memory four words per CP, an operation that fully occupies memory. The first group of 16 parcels delivered to the buffer always contains the instruction required for execution. For this reason, the branch out-of-buffer time is an apparent constant 11 CPs for 16-bank memories and 15 CPs for 8-bank memories.

An instruction buffer is loaded with one word of instructions from each of the 16 memory banks or two words from each of 8 banks. The first four instruction parcels residing in an instruction buffer are always from bank 0.

An exchange sequence voids the instruction buffers by setting their beginning address registers to all ones. This prevents a match with the P register and causes the buffers to be loaded as needed.

Both forward and backward branching is possible within the buffers. A branch does not cause reloading of an instruction buffer if the instruction being branched to is within one of the buffers. Multiple copies of instruction parcels cannot occur in the instruction buffers. Because instructions are held in instruction buffers prior to issue, no attempt should be made to dynamically modify instruction sequences. As long as the unmodified instruction is in an instruction buffer, the modified instruction in memory will not be loaded into an instruction buffer.

Although optimization of code segment lengths for instruction buffers is not a prime consideration when programming the CPU, the number and size of the buffers and the capability for both forward and backward branching can be used to good advantage. Large loops containing up to 256 consecutive instruction parcels can be maintained in the four buffers, or as an alternative, a main program sequence in one or two of the buffers could make repeated calls to short subroutines maintained in the other buffers. The program and subroutines remain in the buffers undisturbed as long as no out-of-buffer condition causes a buffer to be reloaded.

EXCHANGE MECHANISM

Exchange mechanism refers to the technique employed in the CPU for switching instruction execution from program to program. This technique involves the use of blocks of program parameters known as exchange packages and a CPU operation referred to as an exchange sequence.

Throughout the discussion of the exchange package, an alternate bit position representation is used. The bits are numbered from left to right with bit 0 assigned to the 2^{63} bit position. This notation is for the convenience of CAL programmers.

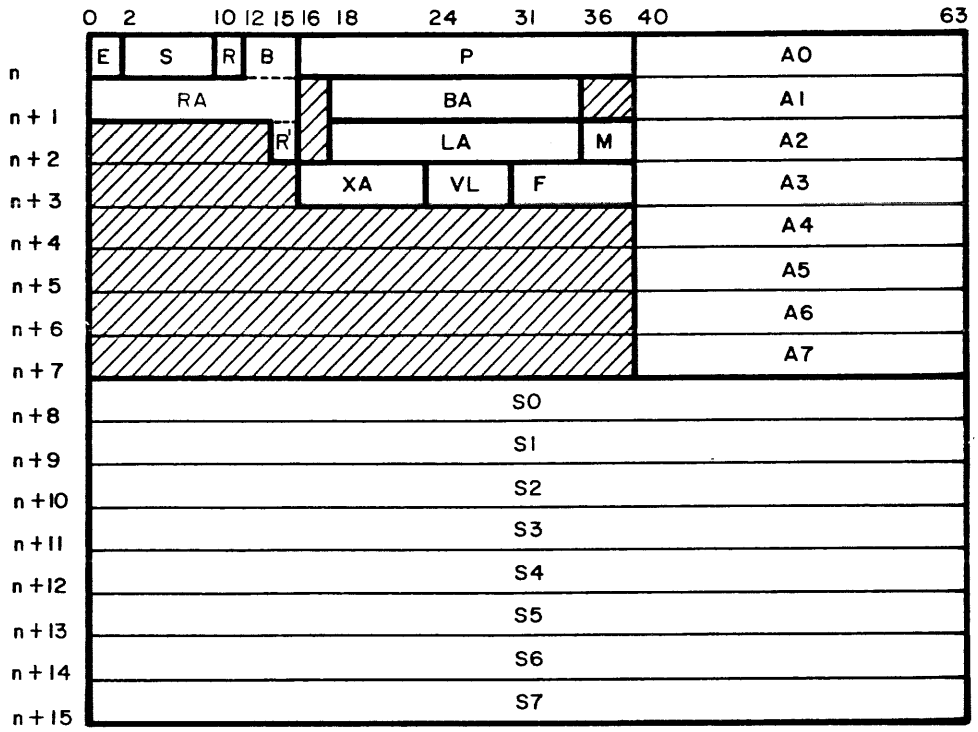
EXCHANGE PACKAGE

An exchange package (figure 3-3) is a 16-word block of data in memory which is associated with a particular computer program. It contains the basic parameters necessary to provide continuity from one execution interval for the program to the next. These parameters consist of the following:

Program address register (P) - 24 bits
Base address register (BA) - 18 bits
Limit address register (LA) - 18 bits
Mode register (M) - 4 bits
Exchange address register (XA) - 8 bits
Vector length register (VL) - 7 bits
Flag register (F) - 9 bits
Current contents of the eight A registers
Current contents of the eight S registers

The exchange package contents are arranged in a 16-word block. Data is swapped from memory to the computer operating registers and back to memory by the exchange sequence. This sequence exchanges the data in a currently active exchange package residing in the operating registers with an inactive exchange package in memory. The XA address of the currently active exchange package specifies the address of the inactive exchange package to be used in the swap. The data is exchanged and a new program execution interval is initiated by the exchange sequence.

The B register, T register, and V register contents are not swapped in the exchange sequence. The data in these registers must be stored and replaced as required by specific coding in the program which supervises the object program execution, or by any program that needs this data.



Registers

S Syndrome bits

R'RAB Read address for error

2^{21} 2^{19} 2^{0}

R'	RA	B

P Program address, 24 bits

BA Base address, 18 bits

LA Limit address, 18 bits

XA Exchange address, 8 bits

VL Vector length, 7 bits

E - Error type (bits 0,1)

10 Uncorrectable memory

01 Correctable memory

R - Read mode (bits 10,11)

00 Scalar

01 I/O

10 Vector

11 Fetch

Bit M - Modes

36 Interrupt on correctable memory error

37 Interrupt on floating-point

38 Interrupt on uncorrectable memory error

39 Monitor mode

F - Flags

31 PCI interrupt

32 MCU interrupt

33 Floating-point error

34 Operand range

35 Program range

36 Memory error

37 I/O interrupt

38 Error exit

39 Normal exit

Figure 3-3. Exchange Package

Memory error data

Two bits in the M (mode) register determine whether or not the exchange package contains data relevant to a memory error if one occurs prior to an exchange sequence. The bits are bit 36, the "Interrupt on correctable memory error bit" and bit 38, the "Interrupt on uncorrectable memory error bit". The error data, consisting of four fields of information, appears in the exchange package if bit 38 is set and an uncorrectable memory error is detected or if bit 36 is set and correctable memory error is encountered.

Error type (E) - The type of memory error encountered, uncorrectable or correctable, is indicated in bits 0 and 1 of the first word of the exchange package. Bit 0 is set for an uncorrectable memory error; bit 1 is set for a correctable memory error.

Syndrome (S) - The 8 syndrome bits used in detecting a memory data error are returned in bits 2 through 9 of the first word of the exchange package. Refer to section 2 for additional information.

Read mode (R) - This field indicates the read mode in progress when a memory data error occurred and consists of bits 10 and 11 of the first word of the exchange package. These bits assume the following values:

00 Scalar (includes memory references with A, B, S, or T registers, or exchange sequence)

01 I/O

10 Vector

11 Instruction fetch

These bits are not valid for range errors.

Read address (R'RAB) - The R'RAB field contains the address at which a memory data error occurred. Bits 12 through 15 (B) of the first word of the exchange package contain bits 2^3 through 2^0 of the address and may be considered as the bank address; bits 0 through 15 (RA) of the second word of the exchange package contain bits 2^{19} through 2^4 of the address. Bits 2^{14} and 2^{15} of the third word of the exchange package (R') contain bits 2^{21} (or 0) and bit 2^{20} of the address.

EXCHANGE REGISTERS

Three special registers are instrumental in the exchange mechanism: the exchange address (XA) register, the mode (M) register, and the flag (F) register.

XA Register

The XA (exchange address) register specifies the first word address of a 16-word exchange package loaded by an exchange operation. The register contains the high-order 8 bits of a 12-bit field that specifies the address. The low-order bits of the field are always 0; an exchange package must begin on a 16-word boundary. The 12-bit limit requires that the absolute address be in the lower 4096 ($10,000_8$) words of memory.

When an execution interval terminates, the exchange sequence exchanges the contents of the registers with the contents of the exchange package at the beginning address (XA) in memory.

M Register

The M (mode) register is a 4-bit register that contains part of the exchange package for a currently active program. Bits are assigned as follows in word 2 of the exchange package:

Bit 36 Correctable memory error mode flag. When this bit is set, interrupts on correctable memory data errors are enabled.

Bit 37 Floating-point error mode flag. When this bit is set, interrupts on floating point errors are enabled.

Bit 38 Uncorrectable memory error mode flag. When this bit is set, interrupts on uncorrectable memory data errors are enabled.

Bit 39 Monitor mode flag. When this bit is set, all interrupts other than memory errors are inhibited.

The 4 bits are selectively set during an exchange sequence. Bit 37, the floating-point error mode flag, can be set or cleared during the execution interval for a program through use of the 0021 (EFI) and 0022 (DFI) instructions. The remaining bits are not altered during the execution interval for the exchange package and can be altered only when the exchange package is inactive in storage.

F Register

The F (flag) register is a 9-bit register that contains part of the exchange package for the currently active program. This register contains nine flags which are individually identified within the exchange package. Setting any of these flags causes interruption of program execution. When one or more flags are set, a request interrupt signal is sent to initiate an exchange sequence. The content of the F register is

stored along with the rest of the exchange package and the monitor program can analyze the nine flags for the cause of the interruption. Before the monitor program exchanges back to the package, it may clear the flags in the F register area of the package. If any bit is set, another exchange occurs immediately.

Any flag other than the memory error flag can be set in the F register only if the currently active exchange package is not in monitor mode. This means that these flags will set only if the low-order bit of the M register is 0. With the exception of the memory error flag, if the program is in monitor mode and the conditions for setting an F register are otherwise present, the flag remains cleared and no exchange sequence is initiated.

ACTIVE EXCHANGE PACKAGE

An active exchange package is an exchange package that is currently residing in the computer operating registers. The interval of time in which the exchange package is active is called the execution interval for the exchange package and also for the program with which it is associated. The execution interval begins with an exchange sequence in which the subject exchange package moves from memory to the operating registers. The execution interval ends as the exchange package moves back to memory in a subsequent exchange sequence.

EXCHANGE SEQUENCE

The exchange sequence is the vehicle for moving an inactive exchange package from memory into the operating registers and at the same time moving the currently active exchange package from the operating registers back into memory. This swapping operation is done in a fixed sequence when all computational activity associated with the currently active exchange package has stopped. The same 16-word block of memory is used as the source of the inactive exchange package and the destination of the currently active exchange package. The location of this block is specified by the content of the exchange address register and is a part of the currently active exchange package. The exchange sequence may be initiated in three different ways.

1. Deadstart sequence
2. Interrupt flag set
3. Program exit

Initiated by deadstart sequence

The deadstart sequence forces the exchange address register content to 0 and also forces a 000 code in the NIP register. These two actions cause the execution of a program error exit using memory address 0 as the location of the exchange package. The inactive exchange package at address 0 is then moved into the operating registers and a program is initiated using these parameters. The exchange package swapped out to address 0 is largely indeterminate as a result of the deadstart operation and is in effect discarded by the subsequent entry of new data at these storage addresses.

Initiated by interrupt flag set

An exchange sequence can be initiated by setting any one of the interrupt flags in the F register. One or more flags set results in a request interrupt signal which initiates an exchange sequence.

Initiated by program exit

Two program exit instructions cause the initiation of an exchange sequence. The timing of the instruction execution is the same in either case and the difference is only in which of the two flags in the F register is set. The two instructions are:

Program code 000 ERR - Error exit

Program code 004 EX - Normal exit

The two exits provide a means for a program to request its own termination. A non-monitor (object) program will usually use the normal exit instruction to exchange back to the monitor program. The error exit allows for termination of an object program that has branched into an unused area of memory or into a data area. The exchange address selected is the same as for a normal exit.

Each of these instructions has a flag in the F register. The appropriate flag is set providing the currently active exchange package is not in monitor mode. The inactive exchange package called in this case is normally one that executes in monitor mode and the flags are sensed for evaluation of the cause of program termination.

The monitor program selects an inactive exchange package for activation by setting the address of the inactive exchange package into the XA register and then executing a normal exit instruction.

Exchange sequence issue conditions

An exchange sequence initiated by an instruction other than 000 or 004 has the following hold issue conditions, execution time, and special cases: (The corresponding information for the 000 and 004 instructions is provided with the instruction descriptions in section 6 of this manual.)

Hold issue conditions:

Instruction buffer data invalid

NIP not blank

Wait exchange flag not set

S, V, or A registers busy

Execution time:

50 CPs; consists of an exchange sequence (36 CPs) and a fetch operation (14 CPs)

Special cases:

Block instruction issue

Block I/O references

Block fetch

EXCHANGE PACKAGE MANAGEMENT

Each 16-word exchange package resides in an area defined during system deadstart that must lie within the lower 4096 words of memory. The package at address 0 is that of the monitor program. Other packages provide for object programs and monitor tasks. These packages lie outside of the field lengths for the programs they represent as determined by the base and limit addresses for the programs. Only the monitor program has a field defined so that it can access all of memory including the exchange package areas. This allows the monitor program to define or alter all exchange packages other than its own when it is the currently active exchange package.

Proper management of exchange packages dictates that a non-monitor program always exchange back to the monitor program that exchanged to it. This ensures that the program information is always swapped back into its proper exchange package.

Consider the case where exchange packages exist for programs A, B, and C. Program A is the monitor program, program B is a user program, and program C is an interrupt-processing program.

The monitor program, A, begins an execution interval following deadstart. No interrupts can terminate its execution interval since it is in monitor mode. The monitor program voluntarily exits by issuing a 004 EX exit instruction. Before doing so, however, it sets the contents of the XA register to point to B's exchange package so that B will be the next program to execute and it sets the exchange address in B's exchange package to point back to the monitor.

The exchange sequence to B causes the exchange address from B's exchange package to be entered in the XA register. At the same time, the exchange address in the XA register goes to B's exchange package area along with all other program parameters for the monitor program. When the exchange is complete, program B begins its execution interval.

Suppose further that while B is executing, an interrupt flag sets initiating an exchange sequence. Since B cannot alter the XA register, the exit is back to the monitor program. Program B's parameters swap back into B's exchange package area; the monitor program parameters held in B's package during the execution interval swap back into the operating registers.

The monitor, upon resuming execution, determines that an interrupt has caused the exchange and sets the XA register to call the proper interrupt processor into execution. It does this by setting XA to point to the exchange package for program C. Then, it clears the interrupt and initiates execution of C by executing a 004 exit instruction. Depending on the design of the operating system, the interrupt-processing program could execute in monitor mode or in user mode.

Further guidance on exchange package management is contained in the Cray Research, Inc. CRAY-1 Operating System maintenance documentation.

MEMORY FIELD PROTECTION

Each object program at execution time has a designated field of memory holding instructions and data. The field limits are specified by the monitor program when the object program is loaded and initiated. The field may begin at any word address that is a multiple of 16 and may continue to another address that is one less than a multiple of 16. The field limits are contained in two registers, the base address register (BA) and the limit address register (LA), described later.

All memory addresses contained in the object program code are relative to the base address which begins the defined field. An object program cannot read or alter any memory location with an absolute address lower than the base address. Each object program reference to memory is checked against the limit and base addresses to determine if the address is within the bounds assigned. A memory write reference beyond the assigned field limits is allowed to issue, but no write occurs. A memory read reference beyond the assigned field limits issues and completes, but data consisting of all zeros is transferred to the appropriate registers.

BA REGISTER

The 18-bit BA register holds the base address of the user field during the execution interval for each exchange package. The contents of this register are interpreted as the high-order 18 bits of a 22-bit memory address. The low-order 4 bits of the address are assumed 0. Absolute memory addresses are formed by adding the product of 2^4 x (BA) to the relative address specified by the CPU instructions. The BA register always indicates a bank 0 memory address.

LA REGISTER

The 18-bit LA register holds the limit address of the user field during the execution interval for each exchange package. The contents of LA are interpreted as the high-order 18 bits of a 22-bit memory address. The low-order 4 bits of the address are assumed 0. The LA register always indicates a bank 0 memory address.

The final address that can be executed or referenced by a program is at $[(LA) \times 2^4] - 1$. Note that the (LA) is absolute, not relative; it is not added to (BA).

PROGRAM RANGE ERROR

The program range error flag sets if an out-of-range memory reference was for an instruction fetch. This could occur in a non-monitor mode program on a branch or jump instruction that calls for a program address that is above or below the limits. The program range error flag causes an error condition that terminates program execution. The monitor program should check the state of the program range error flag and take appropriate action, perhaps aborting the user program.

OPERAND RANGE ERROR

The operand range error flag sets if an out-of-range memory reference was called to read or write an operand for an A, B, S, T, or V register. The operand range error flag causes an error condition that terminates the user program execution. The monitor program should check the state of the operand range error flag and take appropriate action, perhaps aborting the user program.

REAL-TIME CLOCK

Programs can be timed precisely by using the clock period counter. This counter is advanced one count each clock period of 12.5 nanoseconds. Since the clock is advanced synchronously with program execution, it may be used to time the program to an exact number of clock periods.

Instructions used with the real-time clock are:

```
0014j0  RT  Enter the real-time clock register with (Sj)

072ixx  Si  RT  Transmit (RTC) to Si
```

The clock period counter is a 64-bit counter that can be read by a program through the use of the 072 instruction and can be reset only by the 0014j0 monitor instruction.

PROGRAMMABLE CLOCK

A programmable clock is incorporated to measure the duration of intervals accurately. A periodic interrupt can be generated with intervals selected under monitor program control. The clock frequency is 80 Mhz. Intervals from 12.5 nanoseconds to about 53.7 seconds are possible; however, intervals shorter than about 100 microseconds are not practical due to the monitor overhead involved in processing the interrupt.

INSTRUCTIONS

Supporting the programmable clock are four monitor mode instructions and two additional registers: the interrupt interval register (II) and the interrupt countdown counter (ICD).

0014j4 Enter interrupt interval (II) register with (Sj)

0014j5 Clear the programmable clock interrupt request

0014j6 Enable the programmable clock interrupt request

0014j7 Disable the programmable clock interrupt request

INTERRUPT INTERVAL REGISTER

The interrupt interval (II) register is a 32-bit register that can be loaded with a binary value equal to the number of clock periods that are to elapse between programmable clock interrupt requests. The interrupt interval is transferred from the lower 32 bits of the Sj register into both the interrupt interval and the interrupt countdown (ICD) counter when the 0014j4 instruction is executed.

This value is held in the II register and is transferred to the ICD each time the counter reaches 0 and generates an interrupt request. The content of the II register is changed only by another 0014j4 instruction.

INTERRUPT COUNTDOWN COUNTER

The interrupt countdown (ICD) counter is a 32-bit counter that is preset to the contents of the interrupt interval register when the 0014j4 instruction is executed. This counter runs continuously but counts down, decrementing by 1 each clock period until the content of the counter is 0. At this time, it sets the programmable clock interrupt request. The counter then samples the interval value held in the interrupt interval register and repeats the countdown to zero cycle, setting the programmable clock interrupt request at regular intervals determined by the interval value. When the programmable clock interrupt request is set, it remains set until a 0014j5 instruction, clear programmable clock interrupt request, is executed. A programmable clock interrupt request can be set only after the 0014j6 instruction has been executed to enable the interrupt. A programmable clock interrupt request only causes an interrupt when not in monitor mode; a request set in monitor mode is held until the system switches to user mode.

CLEAR PROGRAMMABLE CLOCK INTERRUPT REQUEST

Following a program interrupt interval, an active programmable clock interrupt request may be cleared by executing the 0014j5 clear programmable clock interrupt instruction.

Following any deadstart, the monitor program should ensure the state of the programmable clock interrupt by clearing programmable clock interrupt requests (0014j5) and disabling programmable clock interrupt requests (0014j7).

DEADSTART SEQUENCE

The deadstart sequence is that sequence of operations that starts a program running in the CPU after power has been turned off and then turned on again or whenever a new system is to be re-initialized in the CPU. All registers in the machine, all control latches, and all words in memory are assumed to be invalid after power has been turned on. The following sequence of operations to begin a program is initiated by the MCU or the I/O Subsystem.

1. Turn on master clear signal.
2. Turn on I/O clear signal.
3. Turn off I/O clear signal.
4. Load memory via MCU channel.
5. Turn off master clear signal.

The master clear signal halts all internal computation and forces the critical control latches to predetermined states. The I/O clear signal clears the input channel address register of the MCU channel and activates the MCU input channel. All other input channels remain inactive. The MCU or I/O Subsystem then loads an initial exchange package and monitor program. The exchange package must be located at address 0 in memory. Turning off the master clear signal initiates the exchange sequence to read this package and to begin execution of the monitor program. Subsequent actions are dictated by the design of the operating system.

CPU COMPUTATION SECTION 4

INTRODUCTION

The CPU section consists of operating registers and functional units which are associated with three types of processing: address, scalar, and vector. For address processing, there are two levels of 24-bit registers and two integer arithmetic functional units. For scalar processing, there are two levels of 64-bit scalar registers, four functional units dedicated solely to scalar processing and three floating-point units shared with the vector operations. For vector processing, there are a set of 64-element registers of 64 bits each, four functional units dedicated solely to vector applications, and three floating-point functional units supporting both scalar and vector operations.

Vector and scalar processing are performed on data while address processing operates on internal control information such as addresses and indexes. The flow of data in the computation section is generally from memory to registers and from registers to functional units. The flow of results is from functional units to registers and from registers to memory or back to functional units. Data flows along either the scalar or vector path depending on the mode of processing it is undergoing. An exception is that scalar registers can provide one of the operands required for vector operations performed in the vector functional units.

The flow of address information is from memory or from control registers to address registers. Information in the address registers can then be distributed to various parts of the control network for use in controlling the scalar, vector, and I/O operations. The address registers can also supply operands to two integer functional units. The units generate address and index information and return the result to the address registers. Address information can also be transmsitted to memory from the address registers.

OPERATING REGISTERS

Operating registers are a primary programmable resource of the CPU. They enhance the speed of the system by satisfying the heavy demands for data that are made by the functional units. A single functional unit may require one to three operands per clock period and may deliver results at

a rate of one per clock period. Moreover, multiple functional units can be in use concurrently. To meet these requirements, the CPU has five sets of registers: three primary sets and two intermediate sets.

The three primary sets of registers are address, scalar, and vector designated in this manual as A, S, and V, respectively. These registers are considered primary because functional units can access them directly.

For the scalar and address registers, an intermediate level of registers exists which is not accessible to the functional units. These registers act as buffers for the primary registers. Block transfers are possible between these registers and memory so that the number of memory reference instructions required for scalar and address operands can be greatly reduced. The intermediate registers that support scalar registers are referred to as T registers. The intermediate registers that support the address registers are referred to as B registers.

ADDRESS REGISTERS

The two types of address registers (figure 4-1) are designated A registers and B registers.

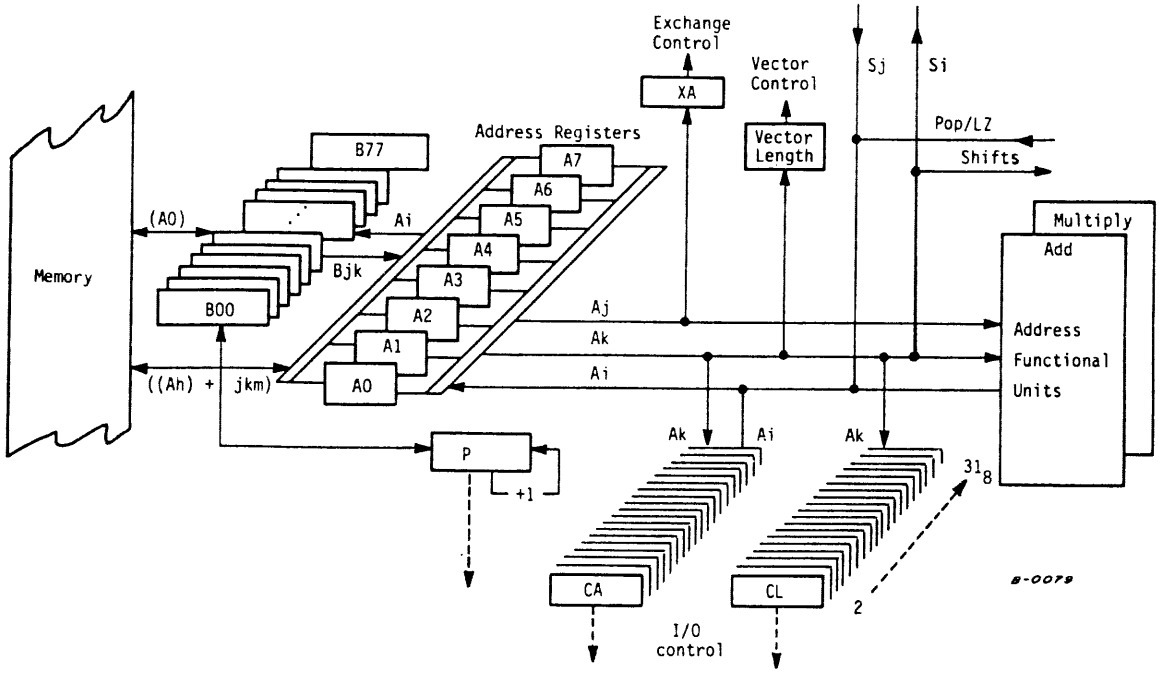


Figure 4-1. Address registers and functional units

A REGISTERS

The eight 24-bit A registers serve a variety of applications, but are primarily used as address registers for memory references and as index registers. They are used to provide values for shift counts, loop control, and channel I/O operations, and also to receive values of population count and leading zeros count. In address applications, they are used to index the base address for scalar memory references and for providing both a base address and an index increment for vector memory references.

The address functional units support address and index generation by performing 24-bit integer arithmetic on operands obtained from A registers and by delivering the results to A registers. There are several address adders devoted exclusively to calculations for memory references. These are not available to the program.

Data can move directly between memory and A registers or can be placed in B registers as an intermediate step. This allows buffering of the data between A registers and memory.

Data can also be transferred between A and S registers.

The vector length register and XA register are set by transmitting a value to it from an A register.

At most, one A register can be entered with data during each clock period. Issue of an instruction is delayed if it would have caused data to arrive at the A registers at the same time as data already being processed which is scheduled to arrive from another source.

When an instruction issues that delivers new data to an A register, a reservation is set for that register to prevent issue of instructions that use the register until the new data has been delivered.

In this manual, the A registers are individually referred to by the letter A and a numeric subscript in the range 0 through 7. Instructions reference A registers by specifying the subscript as the h, i, j, or k designator as described in part 2, section 6.

The only register to which an implicit reference is made is the A_0 register. The use of this register is implied in the following instructions:

```
010ijkm   JAZ
011ijkm   JAN
012ijkm   JAP
013ijkm   JAM

034ijk    Bjk,Ai ,A0
035ijk    ,A0 Bjk,Ai
036ijk    Tjk,Ai ,A0
037ijk    ,A0 Tjk,Ai
```

Refer to part 2, section 6 for additional information concerning the use of A registers by instructions.

B REGISTERS

There are sixty-four 24-bit B registers in the computation section. The B registers are used as intermediate storage for the A registers. Typically, the B registers contain data to be referenced repeatedly over a sufficiently long span so that it is not desirable to retain the data in either A registers or in memory. Examples of uses are loop counts, variable array base addresses, and dimensions.

The transfer of a value between an A register and a B register requires only 1 CP. A block of B registers may be transferred to or from memory at the maximum rate of one 24-bit value per clock period. No reservations are made for B registers and no instructions can issue during block transfers to and from B registers.

In this manual, B registers are individually referred to by the letter B and a 2-digit octal subscript in the range 00 through 77. Instructions reference B registers by specifying the B register number in the jk designator as described in part 2, section 6.

The only B register to which an implicit reference is made is the B_{00} register. On execution of the return jump instruction (007), register B_{00} is set to the next instruction parcel address (P) and a branch to an address specified by ijkm occurs. Upon receiving control, the called routine conventionally saves (B_{00}) so that the B_{00} register is free for the called routine to initiate return jumps of its own. When a called routine wishes to return to its caller, it restores the saved address and executes a 005000 instruction. This instruction, which is a branch to (B00), causes the address saved in B00 to be entered into P as the address of the next instruction parcel to be executed.

SCALAR REGISTERS

The two types of scalar registers (figure 4-2) are designated S registers and T registers.

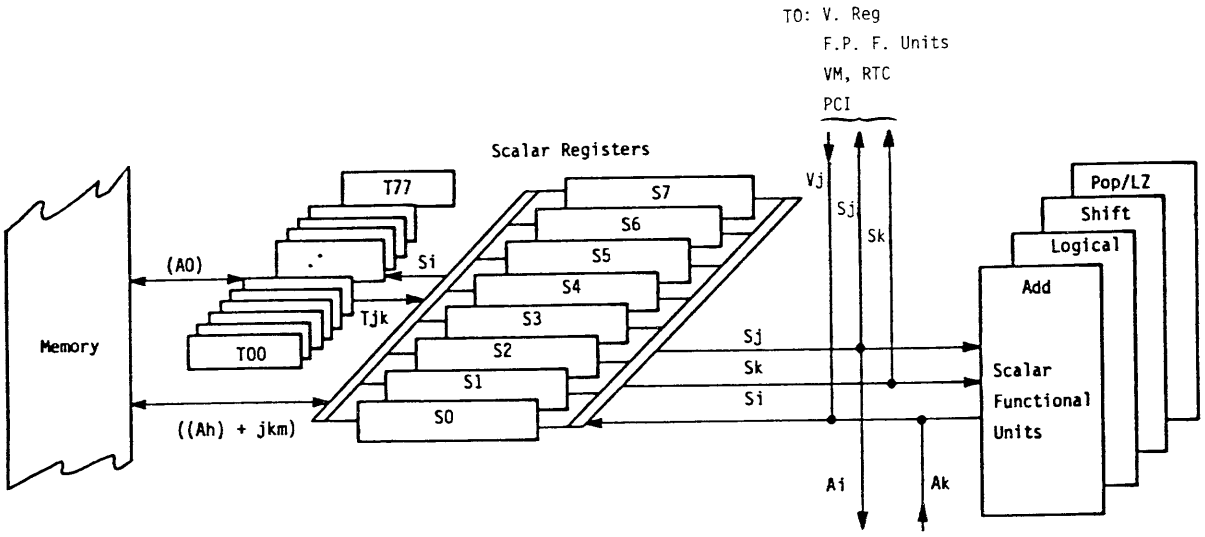


Figure 4-2. Scalar registers and functional units

S REGISTERS

The eight 64-bit S registers are the principal scalar registers for the CPU. These registers serve as the source and destination for operands in the execution of scalar arithmetic and logical instructions. The related functional units perform both integer and floating-point arithmetic operations.

S registers may furnish one operand in vector instructions. Single-word transmission of data between an S register and an element of a V register is also possible.

Data can move directly between memory and S registers or can be placed in T registers as an intermediate step. This allows buffering of scalar operands between S registers and memory.

Data can also be transferred between A and S registers.

Other uses of the S registers are the setting or reading of the vector mask (VM) register or the real-time clock (RTC) register, or setting the interrupt interval (II) register.

At most, one S register can receive data during each clock period. Issue of an instruction is delayed if it would have caused data to arrive at the S registers at the same time as data already being processed which is scheduled to arrive from another source.

When an instruction issues that delivers new data to an S register, a reservation is set for that register to prevent issue of instructions that use the register until the new data has been delivered.

In this manual, the S registers are individually referred to by the letter S and a numeric subscript in the range 0 through 7. Instructions reference S registers by specifying the subscript as the i, j, or k designator as described in part 2, section 6.

The only register to which an implicit reference is made is the S_0 register. The use of this register is implied in the following branch instructions.

014ijkm	JSZ
015ijkm	JSN
016ijkm	JSP
017ijkm	JSM

Refer to part 2, section 6 for additional information concerning the use of S registers by instructions.

T REGISTERS

There are sixty-four 64-bit T registers in the computation section. The T registers are used as intermediate storage for the S registers.

Data may be transferred between T and S registers and between T registers and memory. The transfer of a value between a T register and an S register requires only 1 CP. T registers reference memory through block read and block write instructions. Block transfers occur at a maximum rate of one word per clock period. No reservations are made for T registers and no instructions can issue during block transfers to and from T registers.

In this manual, T registers are referred to by the letter T and a 2-digit octal subscript in the range 00 through 77. Instructions reference T registers by specifying the octal subscript as the jk designator as described in part 2, section 6.

VECTOR REGISTERS

Figure 4-3 illustrates the registers and functional units used for vector operations.

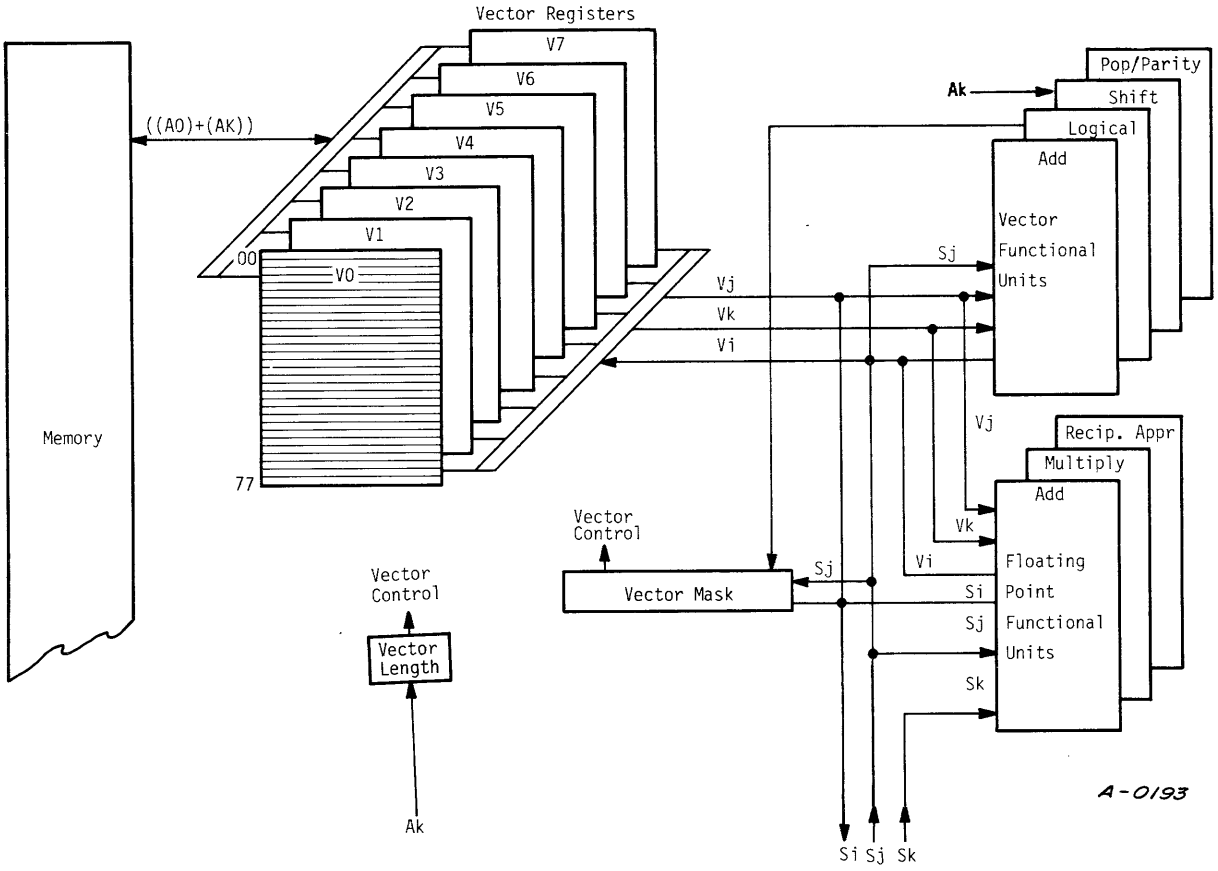


Figure 4-3. Vector registers and functional units

V REGISTERS

Eight V registers, each with 64 elements are the major computational registers of the CPU. Each element of a V register has 64 bits. When associated data is grouped into successive elements of a V register, the register quantity may be treated as a vector. Examples of vector quantities are rows or columns of a matrix or elements of a table.

Computational efficiency is achieved by identically processing each element of a vector. Vector instructions provide for the iterative processing of successive vector register elements. A vector operation begins by obtaining operands from the first element of one or more V

registers and delivering the result to the first element of a V register. Successive elements are provided each clock period and as each operation is performed, the result is delivered to successive elements of the result V register. The vector operation continues until the number of operations performed by the instruction equals a count specified by the content of the vector length (VL) register.

Since many vectors exceed 64 elements, a long vector is processed as one or more 64-element segments and a possible remainder of less than 64 elements. Generally, it is convenient to compute the remainder and process this short segment before processing the remaining number of 64-element segments. However, a programmer may choose to construct the vector loop code in any of a number of ways. The processing of long vectors in FORTRAN is handled by the compiler and is transparent to the programmer.

A result may be received by a V register and retransmitted as an operand to a subsequent operation in the same clock period. This use of a register as both a result and operand register allows for the "chaining" of two or more vector operations together. In this mode, two or more results may be produced per clock period. Chained operation is detected automatically by the CPU and is not explicitly specified by the programmer, although the programmer may reorder certain code segments in order to enable chained operation.

A conflict may occur between scalar and vector operations only for the floating-point operations and storage access. With the exception of these operations, the functional units are always available for scalar operations. A vector operation occupies the selected functional unit until the vector is processed.

Parallel vector operations may be processed in two ways:

1. Using different functional units and all different V registers

2. Using the result stream from one vector register simultaneously as the operand to another operation using a different functional unit (chain mode)

Parallel operations on vectors allow the generation of two or more results per clock period. Most vector operations use two vector registers as operands or one scalar and one vector register as operands. Exceptions are vector shifts, vector reciprocal, and the load or store instructions.

The contents of a V register are transferred to or from memory in a block mode by specifying a first word address in memory, an increment or decrement for the memory address, and a vector length. The transfer then proceeds beginning with the first element of the V register at a maximum rate of one word per clock period, depending upon bank conflicts.

Single-word data transfers are possible between an S register and an element of a V register.

In this manual, the V registers are individually referred to by the letter V and a numeric postscript in the range 0 through 7. Vector instructions reference V registers by specifying the postscript as the i, j, or k designator as described in section 6 of this manual.

Individual elements of a V register are designated in this manual by decimal numbers in the range 00 through 63. These appear as subscripts to vector register references. For example, $V6_{29}$ refers to element 29 of vector register 6.

V register reservations

The term "reservation" describes the register condition when a register is in use and therefore not available for use as a result or as an operand register for another operation. During execution of a vector instruction, reservations are placed on the operand V registers and on the result V register. These reservations are placed on the registers themselves, not on individual elements of the V register.

A reservation for a result register is lifted during "chain slot" time. Chain slot time is the clock period that occurs at functional unit time plus 2 CPs. During this clock period, the result is available for use as an operand in another vector operation. Chain slot time has no effect on the reservation placed on operand V registers. A V register may serve only one vector operation as the source of one or both operands.

No reservation is placed on the VL register during vector processing. If a vector instruction employs an S register, no reservation is placed on the S register. It may be modified in the next instruction after vector issue without affecting the vector operation. The length and scalar operand (if appropriate) of each vector operation is maintained apart from the VL register. Vector operations employing different lengths may proceed concurrently; however, the vector length should not normally be changed between operations that chain because chaining implies operations of the same length.

The A0 and Ak registers in a vector memory reference are treated in a similar fashion. They are available for modification immediately after use.

The vector store instruction (177) is blocked from chain slot execution.

A vector read cannot chain if speed control is in effect. Speed control is caused by bank conflict due to the increment, which varies between 16-bank and 8-bank machines. Speed control is in effect if the memory address increment is a multiple of eight on a 16-bank machine or is a multiple of four on an 8-bank machine.

VECTOR CONTROL REGISTERS

Two registers are associated with vector registers and provide control information needed in the performance of vector operations. They are the vector length (VL) register and the vector mask (VM) register.

VL register

The 7-bit vector length register can be set to 0 through 100_8 and specifies the length of all vector operations performed by vector instructions and the length of the vectors held by the V registers. This register controls the number of operations performed for instructions 140 through 177. The VL register may be set to an A register value using the 0020 instruction.

CAUTION

Cray Research cautions users against increasing VL between operations that may chain together. In some code sequences where the vector length is increased, unexpected results may occur.

Suppose, for example, that during a vector sequence the contents of VL are changed to a larger value and a second operation is initiated to chain to the first operation. The user may expect that the second operation will use the results of the first operation and the operands in the register unaltered by the first operation. However, when the instructions chain together, the second instruction does not receive the anticipated operands beyond the VL specified for the first operation. The user who intends to use the system in this manner must take care to avoid chained operations. Although there may be applications of the characteristic produced by chained operations with different contents for VL, Cray Research takes no responsibility for its use. Chained operation cannot be assured since I/O or other interrupts may "break" the chain.

VM register

The vector mask register has 64 bits, each of which corresponds to a word element in a vector register. Bit 0 corresponds to element 0, bit 63 to element 63. The mask is used in conjunction with vector merge and test instructions to allow operations to be performed on individual vector elements.

The vector mask register may be set from an S register through the 003 instrucion or may be created by testing a vector register for a condition using the 175 instrucion. The mask controls element selection in the vector merge instructions (146 and 147). The contents of VM may be sent to an S register with an 073 instruction.

FUNCTIONAL UNITS

Instructions other than simple transmits or control operations are performed by hardware organizations known as functional units. Each unit implements an algorithm or a portion of the instruction set. Units are independent with the exception of the Reciprocal Approximation and Vector Population Count units, which share some logic. A number of functional units can be in operation at the same time.

A functional unit receives operands from registers and delivers the result to a register when the function has been performed. The units operate essentially in 3-address mode with source and destination addressing limited to register designators.

All functional units perform their algorithms in a fixed amount of time; no delays are possible once the operands have been delivered to the unit. The amount of time required from delivery of the operands to the unit until completion of the calculation is termed the functional unit time and is measured in 12.5-nanosecond clock periods.

The functional units are fully segmented. This means that a new set of operands for unrelated computation may enter a functional unit each clock period even though the functional unit time may be more than 1 CP. This segmentation is possible by capturing and holding the information arriving at the unit or moving within the unit at the end of every clock period.

Thirteen functional units are identified in this manual and are arbitrarily described in four groups: address, scalar, vector, and floating-point. The first three groups each act in conjunction with one of the three primary register types, A, S, and V, to support the address, scalar, and vector modes of processing available in the CRAY-1. The fourth group, floating-point, supports either scalar or vector operations and accepts operands from or delivers results to S or V registers accordingly. In addition, for vector operations, memory acts like a fourteenth functional unit.

ADDRESS FUNCTIONAL UNITS

The address functional units perform 24-bit integer arithmetic on operands obtained from A registers and deliver the results to an A register. The arithmetic is twos complement.

Address add unit

The address add unit performs 24-bit integer addition and subtraction. The unit executes instructions 030 and 031. The addition and subtraction are performed in a similar manner. The twos complement subtraction for the 031 instruction occurs as follows. The ones complement of the Ak operand is added to the Aj operand. Then a 1 is added in the low order bit position of the result.

No overflow is detected in the functional unit.

The result register reservation time is 2 CPs.

Address multiply unit

The address multiply unit executes instruction 032 which forms a 24-bit integer product from two 24-bit operands. No rounding is performed. The result consists of the least significant 24 bits of the product.

This functional unit is designed to handle address manipulations that do not exceed its data capabilities. Therefore, the programmer must be careful when multiplying integers in the unit, because the unit does not detect overflow of the product and significant portions of the product could be lost.

The result register reservation time is 6 CPs.

SCALAR FUNCTIONAL UNITS

The scalar functional units perform operations on 64-bit operands obtained from S registers and in most cases deliver the 64-bit results to an S register. The exception is the population/leading zero count unit which delivers its 7-bit result to an A register.

Four functional units are exclusively associated with scalar operations and are described here. Three functional units are used for both scalar and vector operations and are described in the section, Floating-Point Functional Units.

Scalar add unit

The scalar add unit performs 64-bit integer addition and subtraction. It executes instructions 060 and 061. The addition and subtraction are performed in a similar manner. The twos complement subtraction for the 061 instruction occurs as follows. The ones complement of the Sk operand is added to the Sj operand. Then a 1 is added in the low-order bit position of the result.

No overflow is detected in the unit.

The result register reservation time is 3 CPs.

Scalar shift unit

The scalar shift unit shifts the entire 64-bit contents of an S register or shifts the double 128-bit contents of two concatenated S registers. Shift counts are obtained from an A register or from the jk portion of the instruction. Shifts are end off with zero fill. For a double shift, a circular shift is effected if the shift count does not exceed 64 and the i and j designators are equal and non-zero.

All A register shift counts are considered positive, unsigned integers. If any bit higher than 2^5 is set, the shifted result is all zeros.

The scalar shift unit executes instructions 052 through 057. Single-register shift instructions, 052 through 055, are executed in 2 CPs. Double-register shift instructions, 056 and 057, are executed in 3 CPs.

Scalar logical unit

The scalar logical unit performs bit-by-bit manipulation of 64-bit quantities obtained from S registers. It executes instructions 042 through 051, the mask and Boolean instructions. The 042-051 instructions execute in 1 CP.

Scalar population/parity/leading zero unit

This functional unit executes instructions 026 and 027. Instruction 026ij0 counts the number of bits in an S register having a value of 1 in the operand and executes in 4 CPs. The 026ij1 instruction returns a 1-bit population parity count of the Sj register's contents. Instruction 027, which counts the number of bits of 0 preceding a 1 bit in the operand, executes in 3 CPs. For these instructions, the 64-bit operand is obtained from an S register and the 7-bit result is delivered to an A register.

Most vector functional units perform operations on operands obtained from one or two V registers or from a V register and an S register. The reciprocal unit and the population/parity unit, which require only one operand, are exceptions. Results from a vector functional unit are delivered to a V register.

Successive operand pairs are transmitted from a vector register to a functional unit each clock period. The corresponding result arrives at a vector register n+2 CPs later, where n is the functional unit time and is constant for a given functional unit. The vector length determines the number of operand pairs to be processed by a functional unit.

Four functional units are exclusively associated with vector operations and are described in this subsection. Three functional units are associated with both vector operations and scalar operations and are described in the subsection entitled Floating-Point Functional Units. When a floating-point unit is used for a vector operation, the general description of vector functional units given in this subsection applies.

Vector functional unit reservation

A functional unit engaged in a vector operation remains busy during each clock period and may not participate in other operations. In this state, the functional unit is said to be reserved. Other instructions that require the same functional unit will not issue until the previous operation is completed. Only one functional unit of each type is available to the vector instruction hardware. When the vector operation completes, the reservation is dropped and the functional unit is then available for another operation.

The functional unit is reserved for (VL)+4 CP.

Vector add unit

The vector add unit performs 64-bit integer addition and subtraction for a vector operation and delivers the results to elements of a V register. The unit executes instructions 154 through 157. The addition and subtraction are performed in a similar manner. For the subtraction operations, 156 and 157, the Vk operand is complemented prior to addition and during the addition a 1 is added into the low order bit position of the result.

No overflow is detected by the unit.

The functional unit time for the vector add unit is 3 CPs; the chain slot time is 5 CPs.

Vector shift unit

The vector shift unit shifts the entire 64-bit contents of a V register element or the 128-bit value formed from two consecutive elements of a V register. Shift counts are obtained from an A register. Shifts are end-off with zero fill.

All shift counts are considered positive unsigned integers. If any bit higher than 2^5 is set, the shifted result is all zeros.

The vector shift unit executes instructions 150 through 153.

Functional unit time is 4 CPs; chain slot time is 6 CPs.

Vector logical unit

The vector logical unit performs bit-by-bit manipulation of 64-bit quantities for instructions 140 through 147. The unit also performs the logical operations associated with the vector mask instruction, 175. Because the 175 instruction uses the same functional unit as instructions 140 through 147, it cannot be chained with these logical operations.

Functional unit time is 2 CPs, chain slot time is 4 CPs.

Vector population/parity unit

The vector population count unit counts the 1 bits in each element of the source vector register. The total number of 1 bits is the population count. This population count may be an odd or an even number, as shown by the low-order bit of the population count.

The 174ij1 instruction (vector population count) and the 174ij2 (vector population count parity) use the same operation code as the vector reciprocal approximation instruction. Therefore, some of the restrictions for the reciprocal approximation unit also apply for the vector population instructions. The vector population count instruction delivers the total population count to elements of the destination V register.

The vector population count parity instruction delivers the low-order bit of the count to the destination V register.

The functional unit time is 6 CPs; chain slot time is 8 CPs.

Recursive characteristic of vector functional units

In a vector operation, the result register (designated by i in the instruction) is not normally the same V register as the source of either of the operands (designated by j or k). However, turning the output stream of a vector functional unit back into the input stream by setting i to the same register designator as j or k may be desirable under certain circumstances since it provides a facility for reducing 64 elements down to just a few. The number of terms generated by the partial reduction is determined by the number of values that can be in process in a functional unit at one time (i.e., functional unit time + 2 CP).

When the i designator is the same as the j or k designator, a recursive characteristic is introduced into the vector processing because of the handling of element counters. At the beginning of an operation for which i is the same as j or k, the element counters for both the operand register and the operand/result register are set to 0. Operand registers begin incrementing while the element counter for the result register is held at 0 until functional unit time +2 CP. However, when an operand register is the same as the result register, the element counter for the operand/result register is held at 0 and does not begin incrementing until the first result arrives from the functional unit at functional unit time + 2 CP. This counter then begins to advance by 1 each clock period.

Note that until functional unit time + 2, the initial contents of element 0 of the operand/result register are repeatedly sent to the functional unit. The element counter for the other operand register, however, immediately begins advancing by 1 on each successive clock period, sending the contents of elements 0, 1, 2, ... on successive clock periods.

Thus, the first functional unit time + 2 elements of the operand/result register contain results based on the contents of element 0 of the operand/result register and on successive elements of the other operand register. These functional unit time + 2 elements then provide one of the operands used in calculating the results for the next functional unit time + 2 elements. The third group contains results based on the results delivered to the second group, and so on until the final group of functional unit time + 2 elements is generated as determined by the vector length.

This recursive characteristic of vector processing is applicable to any vector operation, arithmetic or logical. The value initially placed in element 0 of the operand/result register may depend on the operation being performed. For example, when using the floating-point multiply unit recursively, element 0 of the operand/result register is usually set to an initial value of 1.0.

As an example, consider the summation of a vector of floating-point numbers with the following initial conditions for the vector operation:

- All elements of register V1 contain floating-point values.

- Register V2 provides one set of operands and receives the results. Element 0 of this register contains a 0 value.

- The vector length register (VL) contains 64.

A floating-point add instruction (171212) is then executed using register V1 for one operand and using register V2 as an operand/result register. This instruction uses the floating-point add unit which has a functional unit time of 6 CPs causing sums to be generated in groups of eight (functional unit time + 2 = 8). The final eight partial sums of the 64 elements of V1 are contained in elements 56 through 63 of V2.

Specifically, elements of V2 contain the following sums.

```
(V200) =                   = (V200)                                                                   + (V100)
(V201) =                   = (V200)                                                                   + (V101)
(V202) =                   = (V200)                                                                   + (V102)
(V203) =                   = (V200)                                                                   + (V103)
(V204) =                   = (V200)                                                                   + (V104)
(V205) =                   = (V200)                                                                   + (V105)
(V206) =                   = (V200)                                                                   + (V106)
(V207) =                   = (V200)                                                                   + (V107)

(V208) = (V200) + (V108) = (V200)                                                          + (V100) + (V108)
(V209) = (V201) + (V109) = (V200)                                                          + (V101) + (V109)
(V210) = (V202) + (V110) = (V200)                                                          + (V102) + (V110)
(V211) = (V203) + (V111) = (V200)                                                          + (V103) + (V111)
(V212) = (V204) + (V112) = (V200)                                                          + (V104) + (V112)
(V213) = (V205) + (V113) = (V200)                                                          + (V105) + (V113)
(V214) = (V206) + (V114) = (V200)                                                          + (V106) + (V114)
(V215) = (V207) + (V115) = (V200)                                                          + (V107) + (V115)

(V216) = (V208) + (V116) = (V200)                                                 + (V100) + (V108) + (V116)
(V217) = (V209) + (V117) = (V200)                                                 + (V101) + (V109) + (V117)
(V218) = (V210) + (V118) = (V200)                                                 + (V102) + (V110) + (V118)
(V219) = (V211) + (V119) = (V200)                                                 + (V103) + (V111) + (V119)
(V220) = (V212) + (V120) = (V200)                                                 + (V104) + (V112) + (V120)
(V221) = (V213) + (V121) = (V200)                                                 + (V105) + (V113) + (V121)
(V222) = (V214) + (V122) = (V200)                                                 + (V106) + (V114) + (V122)
(V223) = (V215) + (V123) = (V200)                                                 + (V107) + (V115) + (V123)

(V224) = (V216) + (V124) = (V200)                                        + (V100) + (V108) + (V116) + (V124)
(V225) = (V217) + (V125) = (V200)                                        + (V101) + (V109) + (V117) + (V125)
(V226) = (V218) + (V126) = (V200)                                        + (V102) + (V110) + (V118) + (V126)
(V227) = (V219) + (V127) = (V200)                                        + (V103) + (V111) + (V119) + (V127)
(V228) = (V220) + (V128) = (V200)                                        + (V104) + (V112) + (V120) + (V128)
(V229) = (V221) + (V129) = (V200)                                        + (V105) + (V113) + (V121) + (V129)
(V230) = (V222) + (V130) = (V200)                                        + (V106) + (V114) + (V122) + (V130)
(V231) = (V223) + (V131) = (V200)                                        + (V107) + (V115) + (V123) + (V131)

(V232) = (V224) + (V132) = (V200)                               + (V100) + (V108) + (V116) + (V124) + (V132)
(V233) = (V225) + (V133) = (V200)                               + (V101) + (V109) + (V117) + (V125) + (V133)
(V234) = (V226) + (V134) = (V200)                               + (V102) + (V110) + (V118) + (V126) + (V134)
(V235) = (V227) + (V135) = (V200)                               + (V103) + (V111) + (V119) + (V127) + (V135)
(V236) = (V228) + (V136) = (V200)                               + (V104) + (V112) + (V120) + (V128) + (V136)
(V237) = (V229) + (V137) = (V200)                               + (V105) + (V113) + (V121) + (V129) + (V137)
(V238) = (V230) + (V138) = (V200)                               + (V106) + (V114) + (V122) + (V130) + (V138)
(V239) = (V231) + (V139) = (V200)                               + (V107) + (V115) + (V123) + (V131) + (V139)

(V240) = (V232) + (V140) = (V200)                      + (V100) + (V108) + (V116) + (V124) + (V132) + (V140)
(V241) = (V233) + (V141) = (V200)                      + (V101) + (V109) + (V117) + (V125) + (V133) + (V141)
(V242) = (V234) + (V142) = (V200)                      + (V102) + (V110) + (V118) + (V126) + (V134) + (V142)
(V243) = (V235) + (V143) = (V200)                      + (V103) + (V111) + (V119) + (V127) + (V135) + (V143)
(V244) = (V236) + (V144) = (V200)                      + (V104) + (V112) + (V120) + (V128) + (V136) + (V144)
(V245) = (V237) + (V145) = (V200)                      + (V105) + (V113) + (V121) + (V129) + (V137) + (V145)
(V246) = (V238) + (V146) = (V200)                      + (V106) + (V114) + (V122) + (V130) + (V138) + (V146)
(V247) = (V239) + (V147) = (V200)                      + (V107) + (V115) + (V123) + (V131) + (V139) + (V147)

(V248) = (V240) + (V148) = (V200)             + (V100) + (V108) + (V116) + (V124) + (V132) + (V140) + (V148)
(V249) = (V241) + (V149) = (V200)             + (V101) + (V109) + (V117) + (V125) + (V133) + (V141) + (V149)
(V250) = (V242) + (V150) = (V200)             + (V102) + (V110) + (V118) + (V126) + (V134) + (V142) + (V150)
(V251) = (V243) + (V151) = (V200)             + (V103) + (V111) + (V119) + (V127) + (V135) + (V143) + (V151)
(V252) = (V244) + (V152) = (V200)             + (V104) + (V112) + (V120) + (V128) + (V136) + (V144) + (V152)
(V253) = (V245) + (V153) = (V200)             + (V105) + (V113) + (V121) + (V129) + (V137) + (V145) + (V153)
(V254) = (V246) + (V154) = (V200)             + (V106) + (V114) + (V122) + (V130) + (V138) + (V146) + (V154)
(V255) = (V247) + (V155) = (V200)             + (V107) + (V115) + (V123) + (V131) + (V139) + (V147) + (V155)

(V256) = (V248) + (V156) = (V200) + (V100) + (V108) + (V116) + (V124) + (V132) + (V140) + (V148) + (V156)
(V257) = (V249) + (V157) = (V200) + (V101) + (V109) + (V117) + (V125) + (V133) + (V141) + (V149) + (V157)
(V258) = (V250) + (V158) = (V200) + (V102) + (V110) + (V118) + (V126) + (V134) + (V142) + (V150) + (V158)
(V259) = (V251) + (V159) = (V200) + (V103) + (V111) + (V119) + (V127) + (V135) + (V143) + (V151) + (V159)
(V260) = (V252) + (V160) = (V200) + (V104) + (V112) + (V120) + (V128) + (V136) + (V144) + (V152) + (V160)
(V261) = (V253) + (V161) = (V200) + (V105) + (V113) + (V121) + (V129) + (V137) + (V145) + (V153) + (V161)
(V262) = (V254) + (V162) = (V200) + (V106) + (V114) + (V122) + (V130) + (V138) + (V146) + (V154) + (V162)
(V263) = (V255) + (V163) = (V200) + (V107) + (V115) + (V123) + (V131) + (V139) + (V147) + (V155) + (V163)
```

FLOATING-POINT FUNCTIONAL UNITS

The three floating-point functional units perform floating-point arithmetic for both scalar and vector operations. When executing a scalar instruction, operands are obtained from S registers and the result is delivered to an S register. When executing most vector instructions, operands are obtained from pairs of V registers or from a V register and an S register and the results are delivered to a V register. The reciprocal approximation unit, which requires only one input operand, is an exception.

A particular floating-point unit is reserved during the entire execution of the vector instruction.

Information on floating-point out-of-range conditions is contained in the subsection entitled Floating-Point Arithmetic, in this section.

Floating-point add unit

The floating-point add unit performs addition or subtraction of 64-bit operands in floating-point format. The unit executes instructions 062, 063, and 170 through 173.

A result is normalized even if the operands are unnormalized.

Out-of-range exponents are detected as described under Floating-Point Arithmetic.

Functional unit time is 6 CPs; chain slot time is 8 CPs.

Floating-point multiply unit

The floating-point multiply unit executes instructions 064 through 067 and 160 through 167. These instructions provide for full- and half-precision multiplication of 64-bit operands in floating-point format and for computing two minus a floating-point product for reciprocal iterations.

The half-precision product is rounded; the full-precision product is either rounded or unrounded.

Input operands are assumed to be normalized. The unit delivers a normalized result except that the result is not guaranteed to be correct if either input operand is not normalized.

Out-of-range exponents are detected as described under Floating-Point Arithmetic. However, if both operands have zero exponents, the result is considered as an integer product and is not normalized and is not considered out-of-range. This provides a fast method of computing a 48-bit integer product.

Functional unit time is 7 CPs; chain slot time is 9 CPs.

Reciprocal approximation unit

The reciprocal approximation unit finds the approximate reciprocal of a 64-bit operand in floating-point format. The unit executes instructions 070 and 174ij0. Since the Vector Population Count Unit shares some logic with this unit, the k designator must be 0 for the reciprocal approximation instruction to be recognized.

The input operand is assumed to be normalized and if so the result is normalized. The high-order bit of the coefficient is not tested but is assumed to be a 1. If it is not a 1, the result will be incorrect.

Functional unit time is 14 CPs; chain slot time is 16 CPs.

ARITHMETIC OPERATIONS

Functional units in the CPU perform either twos-complement integer arithmetic or floating-point arithmetic.

INTEGER ARITHMETIC

All integer arithmetic, whether 24 bits or 64 bits, is twos complement and is so represented in the registers as illustrated in figure 4-4. The address add unit and multiply units perform 24-bit arithmetic. The scalar add unit and the vector add unit perform 64-bit arithmetic.

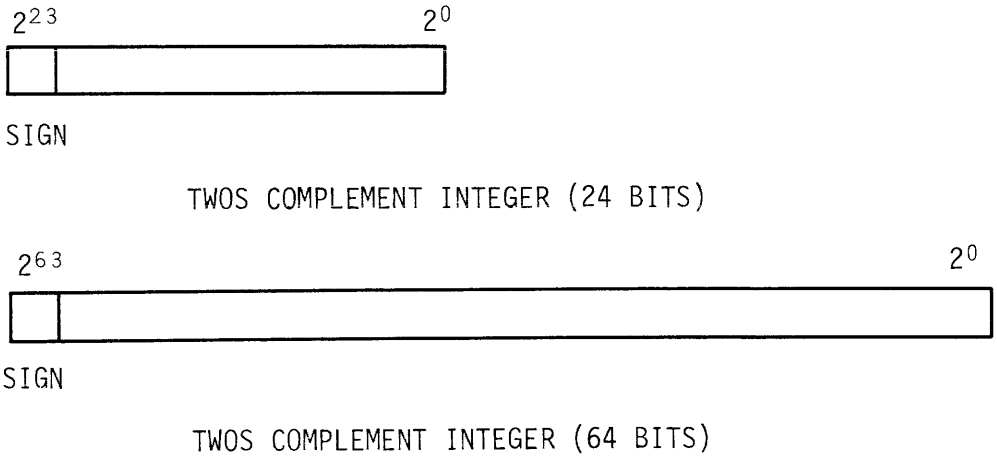


Figure 4-4. Integer data formats

Multiplication of two scaled integer operands may be accomplished by using the floating-point multiply instruction. The floating-point multiply unit recognizes the conditions where both operands have zero exponents as a special case and returns the high-order 48 bits of the product of the coefficients as the coefficient of the result and leaves the exponent field zero. See figure 4-6.

Division of integers would require that they first be converted to floating-point format and then divided using the floating-point units.

FLOATING-POINT ARITHMETIC

Floating-point numbers are represented in a standard format throughout the CPU. This format is a packed representation of a binary coefficient and an exponent (or power of two). The coefficient is a 48-bit signed fraction. The sign of the coefficient is separated from the rest of the coefficient as shown in figure 4-5. Since the coefficient is signed magnitude, it is not complemented for negative values.

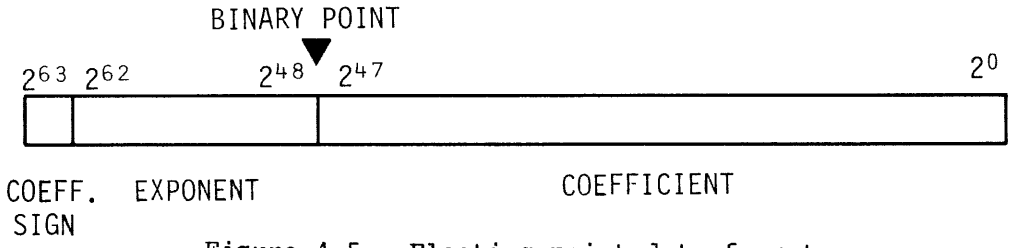


Figure 4-5. Floating-point data formats

The exponent portion of the floating-point format is represented as a biased integer in bits 2^{62} through 2^{48}. The bias that is added to the exponents is 40000_8. The positive range of exponents is 40000_8 through 57777_8. The negative range of exponents is 37777_8 through 20000_8. Thus, the unbiased range of exponents is the following:

$$2^{-20000_8} \text{ through } 2^{+17777_8}$$

In terms of decimal values, the floating-point format of the CRAY-1 allows the expression of numbers accurate to about 15 decimal digits in the approximate decimal range of 10^{-2466} through 10^{+2466}.

A zero value or an underflow result is not biased and is represented as a word of all zeros.

A negative 0 is not generated by any floating-point functional unit, except in the case of a negative 0 operand going to the floating-point multiply functional unit.

Normalized floating-point numbers

A non-zero floating-point number in packed format is normalized if the most significant bit of the coefficient is non-zero. This condition implies that the coefficient has been shifted to the left as far as possible and, therefore, the floating-point number has no leading zeros in the coefficient.

When a floating-point number has been created by inserting an exponent of 40060_8 into a word containing a 48-bit integer, the result should be normalized before being used in a floating-point operation. Normalization is accomplished by adding the unnormalized floating-point operand to 0. Since S0 provides a 64-bit zero when used in the Sj field of an instruction, an operand in Sk can be normalized using the 062i0k instruction. Si, which can be Sk, contains the normalized result.

The 170i0k instruction normalizes Vk into Vi if Vk and Vi are different registers.

Floating-point range errors

Overflow of the floating-point range is indicated by an exponent value of 60000_8 or greater in packed format. Underflow is indicated by an exponent value of 17777_8 or less in packed format. Detection of the overflow condition initiates an interrupt if the floating-point mode flag is set in the mode register and monitor mode is not in effect. The floating-point mode flag can be set or cleared by a user mode program.

Detection of floating-point error conditions by the floating-point units is described in the following paragraphs.

Floating-point add unit - A floating-point add range error condition is generated for scalar operands when the larger incoming exponent is greater than or equal to 60000_8. The floating-point error flag is set and an exponent of 60000_8 is sent to the result register along with the computed coefficient, as in the following example:

```
 60000.4   Range error
+57777.4
 60000.6   Result register
```

NOTE

If the operands to a floating-point add are identical except for sign, the error is suppressed, even though both have exponents greater than or equal to 60000_8.

An underflow condition occurs when the smaller incoming unnormalized exponent is less than or equal to 17777_8. This underflow condition is not detected, and the calculated result goes to the result register.

Floating-point multiply unit - The out-of-range conditions are tested before normalizing. In the floating-point multiply unit, if the exponent of either operand is greater than or equal to 60000_8 or if the sum minus 1 of the two exponents is greater than or equal to 60000_8, the floating-point error flag is set and an exponent of 60000_8 is sent to the result register along with the computed coefficient.

NOTE

If either of the operands is (plus, minus) zero, the error is suppressed, even though the other operand may be out of range.

An underflow condition is detected when the result exponent is less than or equal to 17777_8 and causes an all-zero exponent and coefficient to be returned to the result register.

Underflow is also generated when either, but not both, of the incoming exponents is 0. The condition where both exponents are equal to 0 is treated as an integer multiply and the result is treated normally with no normalization shift of the result allowed. The result is a 48-bit quantity starting with bit 2^{47}. When using this feature, one may consider the operands as 24-bit integers in bits 2^{47} through 2^{24} even though they are actually fractions with the binary point between bits 2^{48} and 2^{47}. In figure 4-6, operand 1 is 4 and operand 2 is 5 which produce a 48-bit result of 24_8. (In this case bit 2^{63} obeys the usual rules for multiplying signs, i.e. - * + = + * - = -, and - * - = + * + = +).

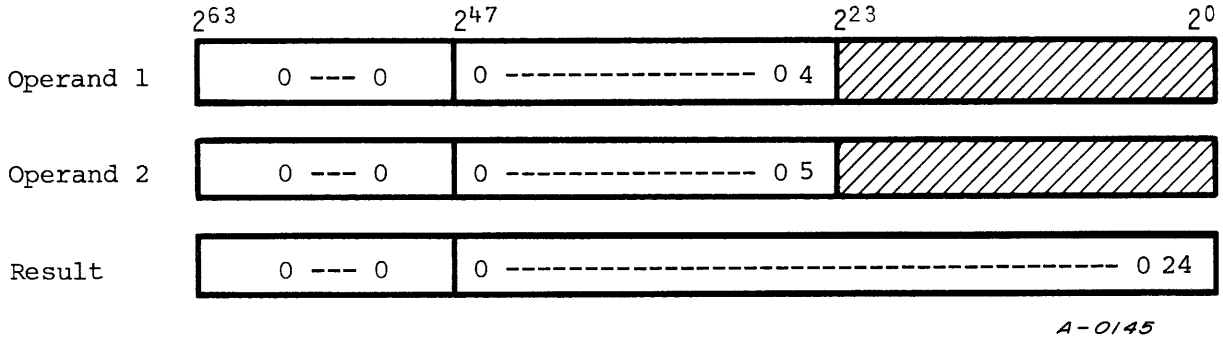


Figure 4-6. Integer multiply in floating-point multiply unit

Floating-point reciprocal approximation unit - For the floating-point reciprocal approximation unit, an incoming operand with an exponent less than or equal to 20001_8 or greater than or equal to 60000_8 causes a floating-point range error. The error flag is set and an exponent of 60000_8 is sent to the result register along with the computed coefficient.

Double-precision numbers

The CPU does not provide special hardware for performing double- or multiple-precision operations. Double-precision computations with 95-bit accuracy are available through software routines provided by Cray Research.

Addition algorithm

Floating-point addition or subtraction is performed in a 49-bit register (figure 4-7). Trial subtraction of the exponents occurs to select the operand to be shifted down for aligning the operands. The larger exponent operand carries the sign. The coefficient of the number with the smaller exponent is shifted right to align with the coefficient of the number with larger exponent. Bits shifted out of the register are lost; no round-up takes place. If the sum carries into the high-order bit, the low-order bit is discarded and an appropriate exponent adjustment is made. All results are normalized and if there is underflow, an all zero result is returned.

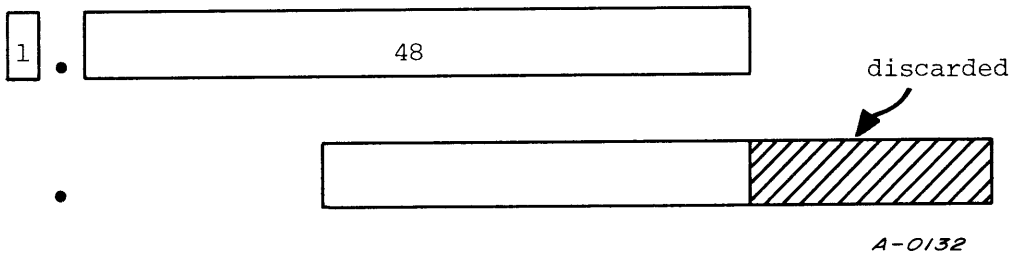


Figure 4-7. 49-bit floating-point addition

Multiplication algorithm

The floating-point multiply unit in the CPU has an input of 48 bits of coefficient into a multiply pyramid (figure 4-8). The pyramid truncates part of the low-order bits of the 96-bit product. To adjust for this truncation, a constant is unconditionally added above the truncation. The value of this constant is $9/2^{56}$. This is an average value determined by summing all carries produced by all possible combinations that could be truncated, and dividing the sum by the number of possible combinations. This averages to nine carries which are injected at the 2^{-56} position.

The errors due to this truncation and rounding are in the range:

$$-0.23 \times 2^{-48} \text{ to } +0.57 \times 2^{-48}$$

or $$-8.17 \times 10^{-16} \text{ to } +20.25 \times 10^{-16}.$$

The effect of this error is, at most, a round up of bit 2^{-48} of the result.

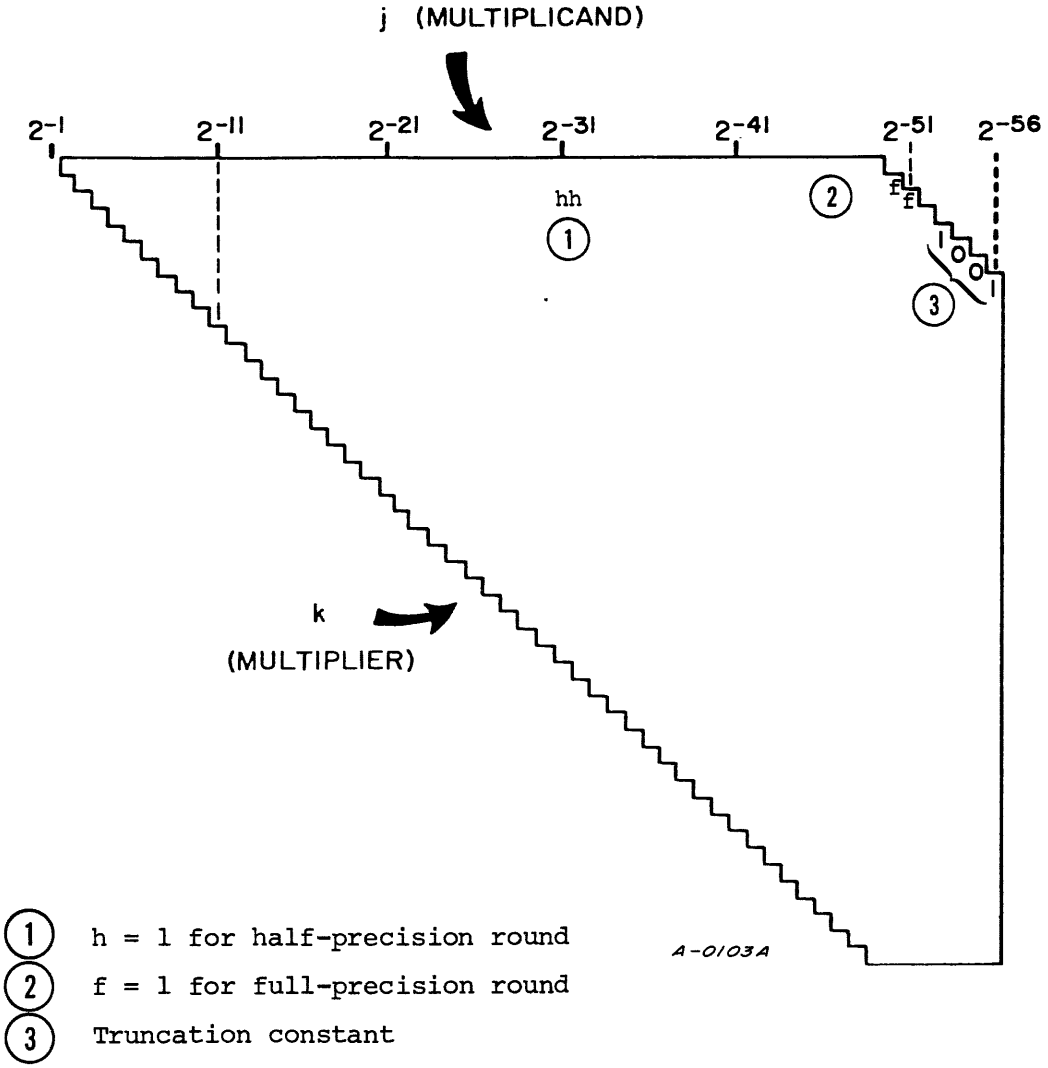


Figure 4-8. Floating-point multiply partial-product sums pyramid

The multiplication is commutative, that is, A times B equals B times A.

In a full-precision rounded multiply, 2 round bits are entered into the pyramid at bit position 2^{-50} and 2^{-51} and allowed to propagate up the pyramid.

For a half-precision multiply, round bits are entered into the pyramid at bit positions 2^{-32} and 2^{-31}. A carry resulting from this entry is allowed to propagate up and a 29-bit result (2^{-1} to 2^{-29}) is transmitted back.

Division algorithm

The CRAY-1 performs floating-point division by the method of reciprocal approximation. This facilitates the hardware implementation of a fully segmented functional unit. Operands may enter the reciprocal unit each clock period because of this segmentation. In vector mode, results are produced at a 1-CP rate. These results may be used in other vector operations during chaining because all functional units in the CRAY-1 have the same result rate. The reciprocal approximation is based on Newton's method.

Newton's method - The division algorithm is an application of Newton's method for approximating the real roots of an arbitrary equation $F(x) = 0$, for which $F(x)$ must be twice differentiable with a continuous second derivative. The method requires making an initial approximation (guess), x_0, sufficiently close to the true root, x_t, being sought (see figure 4-9). For a better approximation, a tangent line is drawn to the graph of $y = F(x)$ at the point $(x_0, F(x_0))$. The X intercept of this tangent line is the better approximation x_1. This may be again repeated using x_1 to find x_2, etc.

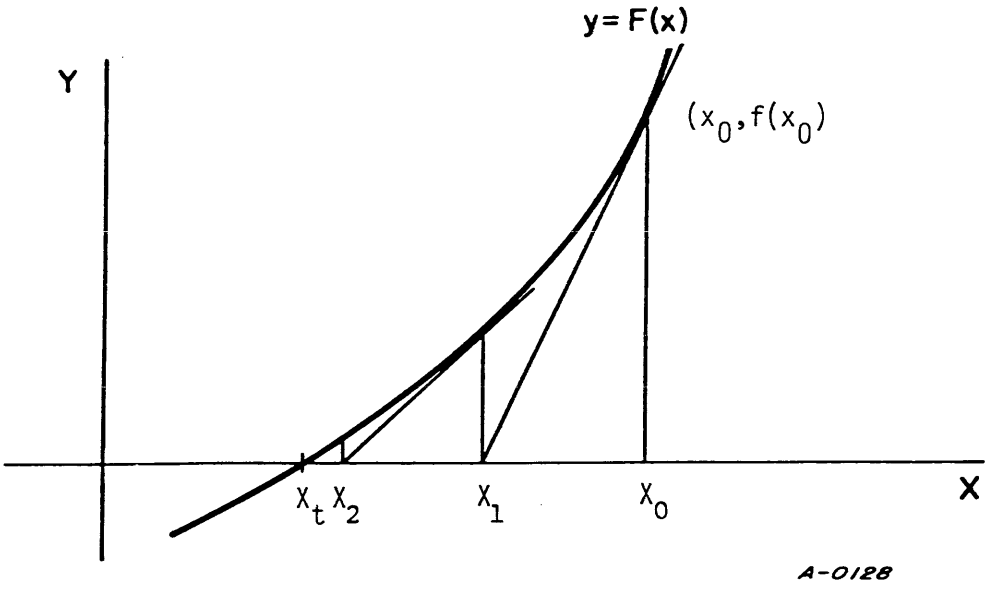


Figure 4-9. Newton's method

Derivation of the division algorithm

A definition for the derivative $F'(x)$ of a function $F(x)$ at point x_t is

$$F'(x_t) = \lim_{x \longrightarrow x_t} \frac{F(x) - F(x_t)}{x - x_t}$$

if this limit exists. If the limit does not exist, $F(x)$ is not differentiable at the point t.

For any point x_i near to x_t,

$$F'(x_t) \approx \frac{F(x_i) - F(x_t)}{x_i - x_t}$$ where $\approx$ means "approximately equal to".

This approximation improves as x_i approaches x_t. Let x_i stand for an approximate solution and let x_t stand for the true answer being sought. The exact answer is the value of x that makes $F(x)$ equal 0. This is the case for $x = x_t$, therefore $F(x_t)$ may be replaced by 0, giving:

$$F'(x_t) \approx \frac{F(x_i)}{x_i - x_t} \qquad (1)$$

Notice that $x_t - x_i$ is the correction that must be applied to an approximate answer, x_i, to give the right answer since $x_i + (x_t - x_i)$ equals x_t. Solving approximation (1) for $(x_t - x_i)$ gives:

$$x_t - x_i = \text{correction} \approx -\frac{F(x_i)}{F'(x_t)}.$$

If this quantity is substituted into the approximation, then:

$$x_t \approx (x_i + \text{correction}) = x_{i+1}.$$

This gives, then, an equation of the form:

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)}, \qquad (2)$$

where x_{i+1} is a better approximation than x_i to the true value, x_t, being sought. The exact answer is generally not obtained at once because the correction term is not generally exact. However, the operation may be repeated until the answer becomes sufficiently close for practical use.

To make use of Newton's method to find the reciprocal of a number B, simply use $F(x) = (1/x - B)$.

First calculating F'(x) we have:

where $F'(x) = \left(\frac{1}{x} - B\right)' = \left(\frac{-1}{x^2}\right)$. thus for any point $x_1 \neq 0$,

$$F'(x_1) = \frac{-1}{x_1^2} \cdot$$ Choosing for x, a value near $\frac{1}{B}$

and applying equation (2),

$$x_2 = x_1 - \frac{\frac{1}{x_1} - B}{-\frac{1}{x_1^2}},$$

$$x_2 = x_1 + x_1^2 \left(\frac{1}{x_1} - B\right),$$

$$x_2 = x_1 + x_1 - x_1^2 B,$$

$$x_2 = 2x_1 - x_1^2 B = x_1(2 - x_1 B).$$

This approximation technique using Newton's method is implemented in the CRAY-1. A hardware table look up provides an initial guess, x_0, to start the process.

$x_0(2 - x_0B)$	1st approximation, I1	Done in reciprocal unit
$x_1(2 - x_1B)$	2nd approximation, I2	
$x_2(2 - x_2B)$	3rd approximation, I3	
$x_3(2 - x_3B)$	4th approximation	Done with software

The CRAY-1 reciprocal approximation functional unit performs three iterations: I1, I2 and I3. I1 is accurate to 8 bits and is found after a table look-up to choose the initial guess, x_0. I2 is the second iteration and is accurate to 16 bits. I3 is the final (third) iteration answer of the reciprocal functional unit and its result is accurate to 30 bits.

A fourth iteration which uses a special instruction within the floating-point multiply functional unit to calculate the correction term, may be used to increase the accuracy of the reciprocal unit's answer to full precision.

The division algorithm that computes S_1/S_2 to full-precision requires four operations:

1. $S_3 = 1/S_2$ — Performed by the reciprocal approximation unit

2. $S_4 = (2 - (S_3 * S_2))$	Performed by the floating-point multiply unit in iteration mode
3. $S_5 = S_4 * S_3$	Performed by the floating-point multiply unit using full-precision. S_5 now equals $1/S_2$ to 48 bit accuracy.
4. $S_6 = S_5 * S_1$	Performed by the floating-point multiply unit using full-precision rounded

The reciprocal approximation at step 1 is correct to 30 bits. The additional Newton iteration (fourth iteration) at steps 2 and 3 increases this accuracy to 48 bits. This iteration answer is applied as an operand in a full-precision rounded multiply operation to obtain the quotient accurate to 48 bits. Additional iterations should not be attempted since erroneous results could be obtained.

Where 29 bits of accuracy is sufficient, the reciprocal approximation instruction may be used with the half-precision multiply to produce a half-precision quotient in only two operations.

1. $S_3 = 1/S_2$	Performed by the reciprocal approximation unit
2. $S_6 = S_1 * S_3$	Performed by the floating-point multiply unit in half-precision

The 19 low-order bits of the half-precision results are returned as zeros with a round applied to the low-order bit of the 29-bit result.

Another method of computing divisions is as follows:

1. $S_3 = 1/S_2$	Performed by the reciprocal approximation unit
2. $S_5 = S_1 * S_3$	Performed by the floating-point multiply unit
3. $S_4 = (2 - (S_3 * S_2))$	Performed by the floating-point multiply unit
4. $S_6 = S_4 * S_5$	Performed by the floating-point multiply unit

A scalar quotient is computed in 29 CPs since operations 2 and 3 issue in successive clock periods. With this method the full precision reciprocal $1/S_2$ is never formed.

A vector quotient using this procedure requires less than four vector times since operations 1 and 2 are chained together. This overlaps one of the multiply operations. (A vector time is 1 CP for each element in the vector.)

For example, two 60-element vectors are divided in 3 * 60 CPs plus overhead. (The overhead associated with the functional units for this case is 38 CPs).

LOGICAL OPERATIONS

The scalar and vector logical units perform bit-by-bit manipulation of 64-bit quantities. Operations provide for forming logical products, differences, sums and merges.

A logical product is the AND function:

```
operand 1   1 0 1 0
operand 2   1 1 0 0
result      1 0 0 0
```

A logical difference is the exclusive OR function:

```
operand 1   1 0 1 0
operand 2   1 1 0 0
result      0 1 1 0
```

A logical sum is the inclusive OR function:

```
operand 1   1 0 1 0
operand 2   1 1 0 0
result      1 1 1 0
```

A logical equivalence is the exclusive NOR function:

```
operand 1   1 0 1 0
operand 2   1 1 0 0
result      1 0 0 1
```

The merge uses two operands and a mask to produce results as follows:

```
operand 1   1 0 1 0 1 0 1 0
operand 2   1 1 0 0 1 1 0 0
mask        1 1 1 1 0 0 0 0
result      1 0 1 0 1 1 0 0
```

The bits of operand 1 pass where the mask bit is 1. The bits of operand 2 pass where the mask bit is 0.

CPU INPUT/OUTPUT SECTION 5

INTRODUCTION

The Input/Output section of the Central Processing Unit contains one Memory Channel and 12 I/O channel pairs. The Memory Channel has one input channel and one output channel. The 12 I/O channel pairs are composed of 12 input channels and 12 output channels. The I/O channels are 16 bits wide, while the Memory Channel is 64 bits wide.

MEMORY CHANNEL

The Memory Channel transfers data between the Central Memory and the Buffer I/O Processor (BIOP). It has two independent channels, one for input to Central Memory, and one for output from Central Memory. Each channel is 64 bits wide, and handles data at approximately 850 Mbits per second. Each channel uses an additional 8 check bits for single error correction/double error detection (SECDED), just as is used in Central Memory.

The CPU side of the channel uses a pair of 16-word buffers to stream the data out of Central Memory. As one buffer block is being sent to the I/O Processor, the other buffer is filling from Central Memory. Another pair of buffers is used to speed data into Central Memory.

At the I/O Processor end of the channels, the data passing into I/O Memory is double-buffered and disassembled into 16-bit parcels. The channel end passing data from I/O Memory simply assembles 64-bit words from the 16-bit parcels.

The instruction fetch, exchange sequence, and normal I/O channel memory requests all take precedence over a Memory Channel Central Memory request. Data is sent in blocks, with 16 words as the normal block length. Each block transfer keeps Central Memory busy for 7 CPs and locks out all other memory requests.

Between block transfers there is a 1-CP wait that allows any other active memory requests to take over Central Memory.

The Memory Channel is controlled by the Buffer I/O Processor. There are no CPU instructions for the Memory Channel. All data transfers between the Central Memory and the BIOP are initiated by the BIOP. All error handling is initiated by the BIOP. Since the Memory Channel is supervised by an I/O Processor, the programming details are contained in part 3, section 7.

I/O CHANNELS

The I/O channels have three basic types of control logic:

1. 16-bit asynchronous; used for Maintenance Control Unit interface or front-end interfaces; the standard CRAY-1 I/O channel

2. 16-bit high-speed asynchronous

3. 16-bit synchronous; used for disk storage access

Each type of I/O channel has the same electrical interface to the I/O cable but differs in timing, protocol, and data rates.

CHANNEL GROUPS

I/O channels are numbered 2 through 31_8 and are divided into four groups as follows:

Group 1	Input channels	2, 6, 12, 16, 22, 26
Group 2	Output channels	3, 7, 13, 17, 23, 27
Group 3	Input channels	4, 10, 14, 20, 24, 30
Group 4	Output channels	5, 11, 15, 21, 25, 31

I/O INSTRUCTIONS

The instructions used with I/O channels are:

0010jk	Set the current address (CA) register for the channel indicated by (Aj) to (Ak) and activate the channel
0011jk	Set the limit address (CL) register for the channel indicated by (Aj) to (Ak)

0012jx Clear the interrupt flag and error flag for the channel indicated by (Aj)

033i0x Transmit channel number to Ai

033ij0 Transmit address of channel (Aj) to Ai

033ij1 Transmit error flag of channel (Aj) to Ai

BASIC I/O CHANNEL OPERATION

Each input or each output channel directly accesses the Central Memory. Input channels store external data in memory and output channels read data from memory. A primary task of a channel is to convert 64-bit Central Memory words into 16-bit parcels or 16-bit parcels into 64-bit Central Memory words. Four parcels make up one Central Memory word, with bits of the parcels assigned to memory bit positions as shown in table 5-1. In both input and output operations, parcel 0 is always transferred first.

Each channel consists of a data channel (4 parity bits, 16 data bits, and 3 control lines), a 64-bit assembly or disassembly register, a channel current address register (CA), and a channel limit address register (CL).

The three control signals are Ready, Resume, and Disconnect. These control signals coordinate the transfer of parcels over the channels. The method of coordination varies among the types of channel; the different methods are explained later.

In addition to the three control signals, either the input or output channel of a pair has a Master Clear line.

Table 5-1. Channel word assembly/disassembly

Characteristic	Bit position	Number of bits	Comment
Channel data bits	$2^{15}-2^{0}$	16	Four 4-bit groups
Channel parity bits		4	One per 4-bit group
CRAY-1 word	$2^{63}-2^{0}$	64	
Parcel 0	$2^{63}-2^{48}$	16	First in or out
Parcel 1	$2^{47}-2^{32}$	16	Second in or out
Parcel 2	$2^{31}-2^{16}$	16	Third in or out
Parcel 3	$2^{15}-2^{0}$	16	Fourth in or out

I/O interrupts can be caused by the following:

- On all output channels, if (CA) becomes equal to (CL), then for each of the channel types on the transmission of the last four parcels:

16-bit asynchronous	Resume for last parcel transmitted sets interrupt
16-bit high-speed asynchronous	Resume for last four parcels transmitted sets interrupt
16-bit synchronous	Interrupt sets when last Ready is sent
16-bit asynchronous	Disconnect received and channel active, or CA = CL and channel active
16-bit high-speed asynchronous	Disconnect received and channel active
16-bit synchronous	Disconnect received and channel active and CA = CL

- External device disconnect received on any input channel and channel is active
- Channel error condition (described later in this section)

The number of the channel causing an interrupt can be determined by the use of a 033 instruction, which reads to Ai the highest priority channel number requesting an interrupt. The lowest numbered channel has the highest priority. The interrupt request continues until cleared by the monitor program at which time an interrupt from the next highest priority channel, if present, may be sensed.

INPUT CHANNEL PROGRAMMING

To start an input operation, the CPU program must perform the following steps:

1. Set the channel limit address to the last word address+1 (LWA+1). See figure 5-1.
2. Set the channel current address to the first word address (FWA).

Setting the current address causes the channel active flag to be set. The channel is then ready to receive data. When a 4-parcel word is assembled, the word is stored in memory at the address contained in the channel current address register. When the word is accepted by memory, the current address is advanced by 1.

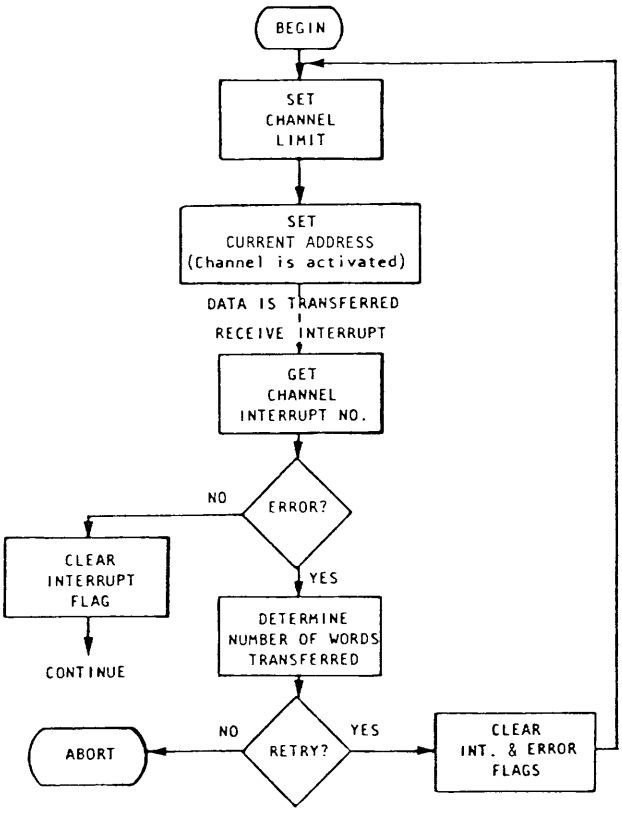


Figure 5-1. Basic I/O program flow chart

The external transmitting device sends a Disconnect pulse to indicate the end of the transfer. When the Disconnect is received, the channel interrupt flag sets and a test is performed to check for a partially assembled word. If the partial word is found, the valid portion of the word is stored in memory and the unreceived, low-order parcels are stored as zeros.

The interrupt flag sets when a Disconnect pulse is received or when an error condition is detected. Setting the interrupt flag deactivates the input channel.

Input channel error conditions

1. Parity error

   16-bit asynchronous channel - When a parcel in error occurs, the parity fault flag sets immediately. The parity fault flag does not generate an interrupt; it is saved and sets the error flag when a disconnect occurs. Therefore, the program should check the state of the error flag when an interrupt is honored.

   16-bit high-speed asynchronous channel - Same as for 16-bit asynchronous channel.

   16-bit synchronous channel - The parcel that contains the error sets the parity fault flag. The parity fault flag causes the subsequent parcels to be stored as all zero parcels. No interrupt is generated by the parity fault. When the data transfer is complete, and the channel goes inactive, the parity fault flag sets the channel error flag. The program should check the state of the error flag when an interrupt is honored.

2. Unexpected Ready pulse

   16-bit asynchronous channel - If a Ready pulse is received when the channel is not active, the ready condition is saved until the channel is activated. At this time a Resume pulse is sent. No error flag is set and no interrupt request is generated.

   If a Ready pulse is received when the memory reference for the previous four parcels is not yet complete, or is received when the channel is active but CA = CL (an extra Ready), the error flag is set. An interrupt request is generated, but no Resume is sent and the data is discarded. When servicing the I/O interrupt, if the channel error flag is set and CA is not equal to CL, a programmed master clear sequence (described later in this section) should be executed on the interrupting channel to clear the external device.

   16-bit high-speed asynchronous channel - If an unexpected Ready pulse is received during a memory reference, the normal burst of four Resume pluses is sent and the data is not sampled. The error flag is set and an interrupt is generated. If the channel is not active or CA = CL when the unexpected Ready pulse arrives, no Resume pulses are sent; the data is not sampled; and the error flag is set to generate an interrupt.

16-bit synchronous channel - A Ready signal is not expected when the channel is inactive, or when CA = CL, or after the first Ready but before the end of the transfer. If an unexpected Ready signal is received, the error flag is set and an interrupt is generated. No further data of the block is transferred. No Resume signal is returned in response to the unexpected Ready signal.

OUTPUT CHANNEL PROGRAMMING

To start an output operation, the CPU program must:

1. Set the channel limit address to the last word address+1 (LWA+1)

2. Set the channel current address to the first word address (FWA).

Setting the current address causes the channel active flag to be set. The channel reads the first word from memory addressed by the contents of the channel's current address register. When the word is received from memory, the channel advances the current address by 1 and starts the data transfer.

After each word is read from memory and the current address is advanced, the limit test is made. The test compares the contents of the channel's current address register and the channel's limit address register. If they are equal, the operation is complete as soon as the last parcel transfer is finished. Tables 5-3, 5-5, and 5-7 show the terminating sequence.

Output channel error condition

The interrupt flag also sets if an error is detected. The only error that an output channel detects is a Resume pulse received when the channel is not active. No external response is generated.

16-BIT ASYNCHRONOUS CHANNELS

Input channels

Table 5-2 illustrates a general view of an input signal sequence.

Data bits 2^0 through 2^{15} - Data bits 2^0, 2^1, ..., 2^{15} are signals carrying the 16-bit parcel of data from the external device to the CPU. They must all be valid within 80 nanoseconds after the leading edge of the Ready signal. Data bit signals must remain unchanged on the lines until the corresponding resume is received by the external device. Normally, data is sent coincident with the Ready pulse and is held until the subsequent Ready pulse.

Table 5-2. 16-bit asynchronous input channel signal exchange

	CPU		EXTERNAL
1.	Activate channel (Set CL and CA).		
2.		←	Data 2^{63}-2^{48} with Ready
3.	Resume →		
4.		←	Data 2^{47}-2^{32} with Ready
5.	Resume →		
6.		←	Data 2^{31}-2^{16} with Ready
7.	Resume →		
8.		←	Data 2^{15}-2^{0} with Ready
9.	Write word to memory and advance current address.		
10a.	Resume →		
10b.	If (CA) = (CL), go to 13.		
11.			If more data, go to 2.
12.		←	Disconnect (ignored if CA = CL or if channel not active
13.	Set interrupt and deactivate channel.		

Parity bits 0 through 3 - Parity bits 0 through 3 are each assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^0 - 2^3
Parity Bit 1	Data Bits 2^4 - 2^7
Parity Bit 2	Data Bits 2^8 - 2^{11}
Parity Bit 3	Data Bits 2^{12} - 2^{15}

Parity bits are sent from the external device to the CPU at the same time as the data bits. They are held stable in the same way as are the data bits.

Ready - The Ready signal sent to the CPU indicates that a parcel of data is being sent to the CPU input channel and may be sampled. The Ready signal is a pulse 50 $\pm$10 nanoseconds wide (at 50% voltage points). The leading edge of Ready at the CPU begins the timing for sampling the data bits.

Resume - Resume is sent from the CPU to the external device to show that the parcel was received and that the CPU is ready for the next data transmission. Resume is a pulse 50 $\pm$3 nanoseconds wide (at 50% voltage points).

Disconnect - This signal is sent from the external device to the CPU and means that the transmission from the external device is complete. It is sent after the Resume is received for the last Ready. Disconnect is a pulse 50 $\pm$10 nanoseconds wide (at the 50% voltage points).

Channel Master Clear - This signal may be programmed (see description of Programmed Master Clear later in this section) or may result from a Clear I/O Signal.

Output channels

Table 5-3 illustrates a general view of an output signal sequence.

Data bits 2^0 through 2^{15} - Data bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the CPU to an external device. They are all sent at the same time, within 5 nanoseconds of the leading edge of the Ready pulse. Data bit signals remain steady on the lines until the Resume pulse is received.

Parity bits 0 through 3 - Parity bits 0, 1, 2, and 3 are each assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^0 - 2^3
Parity Bit 1	Data Bits 2^4 - 2^7
Parity Bit 2	Data Bits 2^8 - 2^{11}
Parity Bit 3	Data Bits 2^{12} - 2^{15}

Parity bits are sent from the CPU to the external device at the same time as the data bits. They are held stable in the same way as are the data bits.

Table 5-3. 16-bit asynchronous output channel signal exchange

CPU	EXTERNAL
1. Activate channel (set CL and CA).	
2. Read word from memory and advance current address.	
3. Data 2^{63}-2^{48} with Ready ⟶	
4.	⟵ Resume
5. Data 2^{47}-2^{32} with Ready ⟶	
6.	⟵ Resume
7. Data 2^{31}-2^{16} with Ready ⟶	
8.	⟵ Resume
9. Data 2^{15}-2^{0} with Ready ⟶	
10.	⟵ Resume
11. If (CA) ≠ (CL), go to 2.	
12. Disconnect ⟶	
13. Set interrupt and deactivate channel.	

Ready - The Ready signal sent from the CPU to the external device indicates that the data is present and may be sampled. The Ready signal is a pulse 50 ±3 nanoseconds wide (at 50% voltage points). The leading edge of Ready may be used to time data sampling in the external device.

Resume - Resume is sent from the external device to the CPU to show that the parcel was received and that the external device is ready for the next parcel transmission. Resume is a pulse 50 ±10 nanoseconds wide (at 50% voltage points).

Disconnect - Disconnect is a signal sent from the CRAY-1 to the external device that means the transmission from the CRAY-1 is complete. It is sent after the CPU has received the Resume from the last Ready. The Disconnect is a pulse 50 ±3 nanoseconds wide (at 50% voltage points).

Cabling Restrictions - The normal length of cable for the 16-bit asynchronous channel is 70 feet (21.3 meters). This assumes a 10-foot (305-cm) drop cable at the CPU, a 50-foot (15.3-meter) data cable, and another 10-foot (305-cm) drop cable at the external device. When used with a Cray Research, Inc. front-end interface, the cable length increases to 300 feet (91.5 meters).

16-BIT HIGH-SPEED ASYNCHRONOUS CHANNELS

Input channels

Table 5-4 illustrates a general view of an input signal sequence.

Data bits 2^0 through 2^{15} - Data bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data to the CPU. The data lines must be stable no later than 80 nanoseconds after the leading edge of the associated Ready pulse and must be held stable until at least 120 nanoseconds after the leading edge of the same Ready. Note that if the device is transmitting at the maximum allowable rate, it is normal for a data parcel to overlap the subsequent Ready pulse. Typically, data is transmitted 50 nanoseconds after the leading edge of Ready and held until 50 nanoseconds after the leading edge of the following Ready pulse.

Parity bits 0 through 3 - Parity bits 0, 1, 2, and 3 are each a parity bit assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^0 - 2^3
Parity Bit 1	Data Bits 2^4 - 2^7
Parity Bit 2	Data Bits 2^8 - 2^{11}
Parity Bit 3	Data Bits 2^{12} - 2^{15}

Table 5-4. 16-bit high-speed asynchronous input channel signal exchange

	CPU	EXTERNAL
1.	Activate channel (Set CL and CA).	
2.	Resume ⟶	
3.	Resume ⟶	
4.	Resume ⟶	
5.	Resume ⟶	If done, go to 11.
6.		⟵ Data 2^{63}-2^{48} with Ready
7.		⟵ Data 2^{47}-2^{32} with Ready
8.		⟵ Data 2^{31}-2^{16} with Ready
9.		⟵ Data 2^{15}-2^{0} with Ready
10.	Write word to memory and advance current address; go to 2.	
11.		⟵ Disconnect
12.	Set interrupt and deactivate channel.	

Parity bits are sent from the external device to the CPU at the same time as the data bits. They are held stable in the same way as are the data bits.

Ready - The Ready signal sent to the CPU indicates that data will soon be sent to the CPU input channel and may be sampled. The Ready signal is a pulse 50 ±10 nanoseconds wide (at the 50% voltage points) sent in groups of four. The leading edge of Ready at the CPU begins the timing for sampling the data bits.

The first Ready pulse of a group may be transmitted by the device as soon as it detects the leading edge of the first Resume pulse for that group. The time from the leading edge of one Ready pulse to the leading edge of the following Ready pulse in the same group must be greater than 90 nanoseconds.

Resume - This signal is sent to the external device to show that the CPU is ready for the next data transmission. Resume is a pulse 50 ±3 nanoseconds wide (at the 50% voltage points) sent in groups of four.

For any group of Resume pulses, the time from the leading edge of one Resume to the leading edge of the next Resume is 100 ±3 nanoseconds.

Disconnect - This signal is sent from the external device to the CPU and indicates that the transmission from the external device is complete. It is sent after the last Ready. The Input Disconnect pulse must be transmitted no earlier than 20 nanoseconds after the leading edge of the final Ready pulse. Disconnect is a pulse 50 ±10 nanoseconds wide (at the 50% voltage points).

Output channels

Table 5-5 illustrates a general view of an output signal sequence.

Data bits 2^0 through 2^{15} - Data bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the CPU to an external device. They are all sent at the same time, within 5 nanoseconds of the leading edge of the Ready pulse. Data bits remain steady on the lines until the next parcel is sent, or until the Resume pulse is received, whichever occurs first.

Parity bits 0 through 3 - Parity bits 0, 1, 2, and 3 are each assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^0 - 2^3
Parity Bit 1	Data Bits 2^4 - 2^7
Parity Bit 2	Data Bits 2^8 - 2^{11}
Parity Bit 3	Data Bits 2^{12} - 2^{15}

Parity bits are sent from the CPU to the external device at the same time as the data bits. They are held stable in the same way as are the data bits.

Channel Master Clear - The Channel Master Clear may be programmed (see description of Programmed Master Clear later in this section) or may result from a Clear I/O signal. The Master Clear signal may be used by the external devices for control purposes or may be ignored.

Ready - The Ready signal sent from the CPU to the external device indicates that the data is present and may be sampled. The Ready signal is a pulse 50 ±3 nanoseconds wide (at the 50% voltage points) sent in groups of four. For any group of Ready pulses, the time from the leading edge of one Ready to the leading edge of the next Ready is 100 ±3 nanoseconds. The leading edge of Ready may be used to time data sampling in the external device.

Table 5-5. 16-bit high-speed asynchronous output channel signal exchange

CPU	EXTERNAL
1. Activate channel (set CL and CA).	
2. Read word from memory and advance current address.	
3. Data 2^{63}-2^{48} with Ready ⟶	
4. Data 2^{47}-2^{32} with Ready ⟶	
5. Data 2^{31}-2^{16} with Ready ⟶	
6. Data 2^{15}-2^{0} with Ready ⟶ (with Disconnect if this is the last word)	
7.	⟵ Resume
8. If (CA) ≠ (CL), go to 2.	
9. Set interrupt and deactivate channel.	

Resume - Resume is sent from the external device to the CPU to show that the 64-bit word of four parcels was received and that the external device is ready for the next word (four parcels). Resume is a pulse 50 ±10 nanoseconds wide (at the 50% voltage points). The pulse must be received at the CPU no earlier than 230 nanoseconds after the leading edge of the first Ready pulse is transmitted.

Disconnect - Disconnect is a signal sent from the CPU to the external device that means the transmission from the CPU is complete. It is sent with the last Ready ±3 nanoseconds. The Disconnect pulse is 50 ±3 nanoseconds wide (at the 50% voltage points).

Cabling Restrictions - The 16-bit high-speed asynchronous channel can handle a cable length of up to 70 feet (21.3 meters). This assumes a 10-foot (305-cm) drop cable at the CPU, a 50-foot (15.3-meter) data cable, and another 10-foot (305-cm) drop cable at the external device.

Input channels

Table 5-6 illustrates a general view of an input signal sequence.

Data bits 2^0 through 2^{15} - Data bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the external device to the CPU. They are all valid within 5 nanoseconds of each other. Data bit signals must remain unchanged on the lines until the next parcel is sent. Data lines must be stable at the CPU backpanel within 90 nanoseconds after the corresponding Resume pulse is transmitted from the CPU backpanel.

Parity bits 0 through 3 - Parity bits 0, 1, 2, and 3 are each assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^0 - 2^3
Parity Bit 1	Data Bits 2^4 - 2^7
Parity Bit 2	Data Bits 2^8 - 2^{11}
Parity Bit 3	Data Bits 2^{12} - 2^{15}

Parity bits are sent from the external device to the CPU at the same time as the data bits. They are held stable in the same way as are the data bits.

Ready - The Ready signal is a block ready in response to the first resume of a block. The Ready signal is a pulse 50 $\pm$10 nanoseconds wide (at the 50% voltage points). It is sent from the external device to the CPU. Ready occurs within 5 nanoseconds of the leading edges of the first parcel of data bits.

Resume - Resume is sent from the CPU to the external device to initiate the synchronous data transfer and to time the sending of data at the CPU. The first Resume pulse is 50 $\pm$3 nanoseconds wide (at the 50% voltage points). Following the first Resume, which awaits a Ready response, the signal is sent in one group of three Resumes (150 $\pm$7 ns) followed by as many groups of four Resumes (200 $\pm$9 ns) as required to complete the block transfer.

Disconnect - Disconnect is a signal sent from the external device to the CPU indicating that transmission from the external device is complete. It is sent with parcel 2 of the last data word or at any later time. Disconnect is a pulse 50 $\pm$10 nanoseconds wide (at the 50% voltage points).

Block length restrictions - The input channel has no restrictions on block length. The disk controller, which is the only device connected to this type of channel, has rigid restrictions on its block lengths. Input transmissions are limited to 1 or 4 or 512 64-bit words.

Table 5-6. 16-bit synchronous input channel signal exchange

CPU		EXTERNAL
1.	Activate channel (set CL and CA).	
2.	Resume ⟶	
3.		⟵ Data 2^{63}-2^{48} with Ready
4.	Resume ⟶ (150 ns pulse)	
5.	Resume ⟶ (150 ns pulse)	⟵ Data 2^{47}-2^{32}, no Ready
6.	Resume ⟶ (150 ns pulse)	⟵ Data 2^{31}-2^{16}, no Ready
7.		⟵ Data 2^{15}-2^{0} , no Ready
8.	Write word to memory; advance current address.	
9.	If last word, go to 16.	
10.	Resume ⟶ (200 ns pulse)	
11.	Resume ⟶ (200 ns pulse)	⟵ Data 2^{63}-2^{48}, no Ready
12.	Resume ⟶ (200 ns pulse)	⟵ Data 2^{47}-2^{32}, no Ready
13.	Resume ⟶ (200 ns pulse)	⟵ Data 2^{31}-2^{16}, no Ready
14.		⟵ Data 2^{15}-2^{0} , no Ready
15.	Go to 8.	
16.	Wait for Disconnect.	⟵ If last word, Disconnect.
17.	Set interrupt and deactivate channel.	

Clock - A Clock signal is supplied over a separate cable (one per DCU cabinet) to the external device from the CPU. This Clock signal synchronizes signals at the external device interface connector.

Cabling restrictions - The synchronous channels use a fixed length cable providing constant propagation time for the signals. This cable delay is designed into the control logic; therefore, the cable length and propagation speed cannot be changed. The total cable length between the CPU and the external device is 17 feet (518 cm). The cable run for a synchronous channel uses one 10 foot (305 cm) drop cable at the CPU and one 7 foot (213 cm) length of data cable at the external device.

Output channels

Table 5-7 illustrates a general view of an output signal sequence.

Data bits 2^0 through 2^{15} - Data bits 2^0, 2^1, ..., 2^{15} are signals carrying a 16-bit parcel of data from the CPU to the external device. They are sent with the leading edge of the Ready pulse $\pm$ 5 nanoseconds. Data bit signals remain unchanged on the lines until the next parcel is sent.

Parity bits 0 through 3 - Parity bits 0, 1, 2, and 3 are each assigned to a 4-bit group of data bits. The parity bits are set or cleared to give the bit group odd parity. Bit assignments are as follows:

Parity Bit 0	Data Bits 2^0 - 2^3
Parity Bit 1	Data Bits 2^4 - 2^7
Parity Bit 2	Data Bits 2^8 - 2^{11}
Parity Bit 3	Data Bits 2^{12} - 2^{15}

Parity bits are sent from the CPU to the external device at the same time as the data bits. They are held stable in the same way as are the data bits.

Channel Master Clear - The Channel Master Clear may be programmed (see description of Programmed Master Clear later in this section) or may be the result of a Clear I/O signal. The programmed Master Clear is a static signal sent from the CPU to an external device. The Master Clear signal may be used by the external device for control purposes or it may be ignored.

Ready - The Ready signal is sent from the CPU to the external device to indicate that the data is valid. The first Ready signal is a pulse 50 $\pm$3 nanoseconds wide (at the 50% voltage points). Following the first Ready, which awaits a Resume response, the signal is sent in one group of three Readies (150 $\pm$7 ns) followed by as many groups of four Readies (200 $\pm$9 ns) as required to complete the block transfer.

Resume - Resume is sent from the external device to the CPU in response to the first Ready signal. The Resume pulse is 50 $\pm$10 nanoseconds wide (at the 50% voltage points).

Table 5-7. 16-bit synchronous output channel signal exchange

CPU	EXTERNAL
1. Activate channel (set CL and CA).	
2. Read word from memory and advance current address.	
3. Data 2^{63}-2^{48} with Ready ⟶ (With Disconnect if last word)	
4.	⟵ Resume
5. Data 2^{47}-2^{32} with Ready ⟶	150 ns Ready pulse
6. Data 2^{31}-2^{16} with Ready ⟶	
7. Data 2^{15}-2^{0} with Ready ⟶	
8. If (CA) = (CL), go to 15.	
9. Read word from memory and advance current address.	
10. Data 2^{63}-2^{48} with Ready ⟶ (with Disconnect if (CA) = (CL))	200 ns Ready pulse
11. Data 2^{47}-2^{32} ⟶	
12. Data 2^{31}-2^{16} ⟶	
13. Data 2^{15}-2^{0} ⟶	
14. If (CA) ≠ (CL), go to 9.	
15. Set interrupt and deactivate channel.	

Disconnect - Disconnect is a signal sent from the CPU to the external device indicating that the transmission from the CPU is complete. It is sent with parcel 0 of the last 64-bit data word. Disconnect is a pulse 50 ±3 nanoseconds wide (at the 50% voltage points).

Block length restrictions - The output channel has no restrictions on block length. The disk controller, which is the only device connected to this type of channel, has rigid restrictions on its block lengths. Output transmissions are limited to 1 or 512 64-bit words.

Clock - A Clock signal is supplied over a separate cable (one per DCU cabinet) to the external device from the CPU. This Clock signal synchronizes signals at the external device interface connector.

Cabling restrictions - The 16-bit synchronous channels use a fixed length cable providing a constant propagation time for the signals. This cable delay is designed into the control logic; therefore, the cable length and propagation speed cannot be changed. The total cable length between the CPU and the external device is 17 feet (518 cm). The cable run for a synchronous channel uses one 10-foot (305 cm) drop cable at the CPU and one 7-foot (213 cm) length of data cable at the external device.

PROGRAMMED MASTER CLEAR TO EXTERNAL DEVICE

The CPU contains a mechanism for sending a Master Clear signal to an external device. The Master Clear is sent by either the input channel or the output channel as follows:

- Asynchronous channels - Master Clear sent on input channel
- High-speed asynchronous channels - Master Clear sent on output channel
- Synchronous channels - Master Clear sent on output channel

Sequence for asynchronous channels

The external Master Clear sequence for 16-bit normal-speed asynchronous channels is as follows:

1. 0012jk — Clear output channel to ensure that CPU activity on the channel pair has stopped.
2. 0012jk — Clear input channel to ensure that external activity on the channel pair has stopped.
3. 0011jk — Set the input channel limit to an arbitrary value.
4. 0010jk — Set the input channel current address equal to the same value. This initiates the Master Clear signal.
5. 0012jk — Clear the input channel. This stops the input channel activity just initiated.
6. Delay 1 — Device dependent. This determines the duration of the Master Clear signal.

7. 0011jk Set the input channel limit. This value may be the same value as used in steps 3 and 4. This turns off the Master Clear signal.

8. Delay 2 Device dependent. This allows time for initialization activities in the attached device to complete.

For Cray Research front-end interfaces, delays 1 and 2 should both be a minimum of 80 CPs.

Sequence for high-speed asynchronous and synchronous channels

The external Master Clear sequence for high-speed asynchronous and synchronous channels is as follows:

1. 0012jk Clear output channel interrupt to ensure that CPU activity on the channel pair has stopped.

2. 0012jk Clear input channel interrupt to ensure that external activity on the channel pair has stopped.

3. 0011jk Set the output channel limit to an arbitrary value.

4. 0010jk Set the output channel current address equal to the same value. This initiates the Master Clear signal.

5. 0012jk Clear the output channel. This stops the output channel activity just initiated.

6. Delay 1 Device dependent. This determines the duration of the Master Clear signal.

7. 0011jk Set the output channel limit. This value may be the same value as used in steps 3 and 4. This turns off the Master Clear signal.

8. Delay 2 Device dependent. This allows time for initialization activities in the attached device to complete.

9. Read disk subsystem status (high-speed synchronous channel only). A subsystem status should be taken and discarded to remove any false status left by the Master Clear sequence.

For the synchronous channel, delay 1 should be a minimum of 1 CP and delay 2, a minimum of 20 CPs.

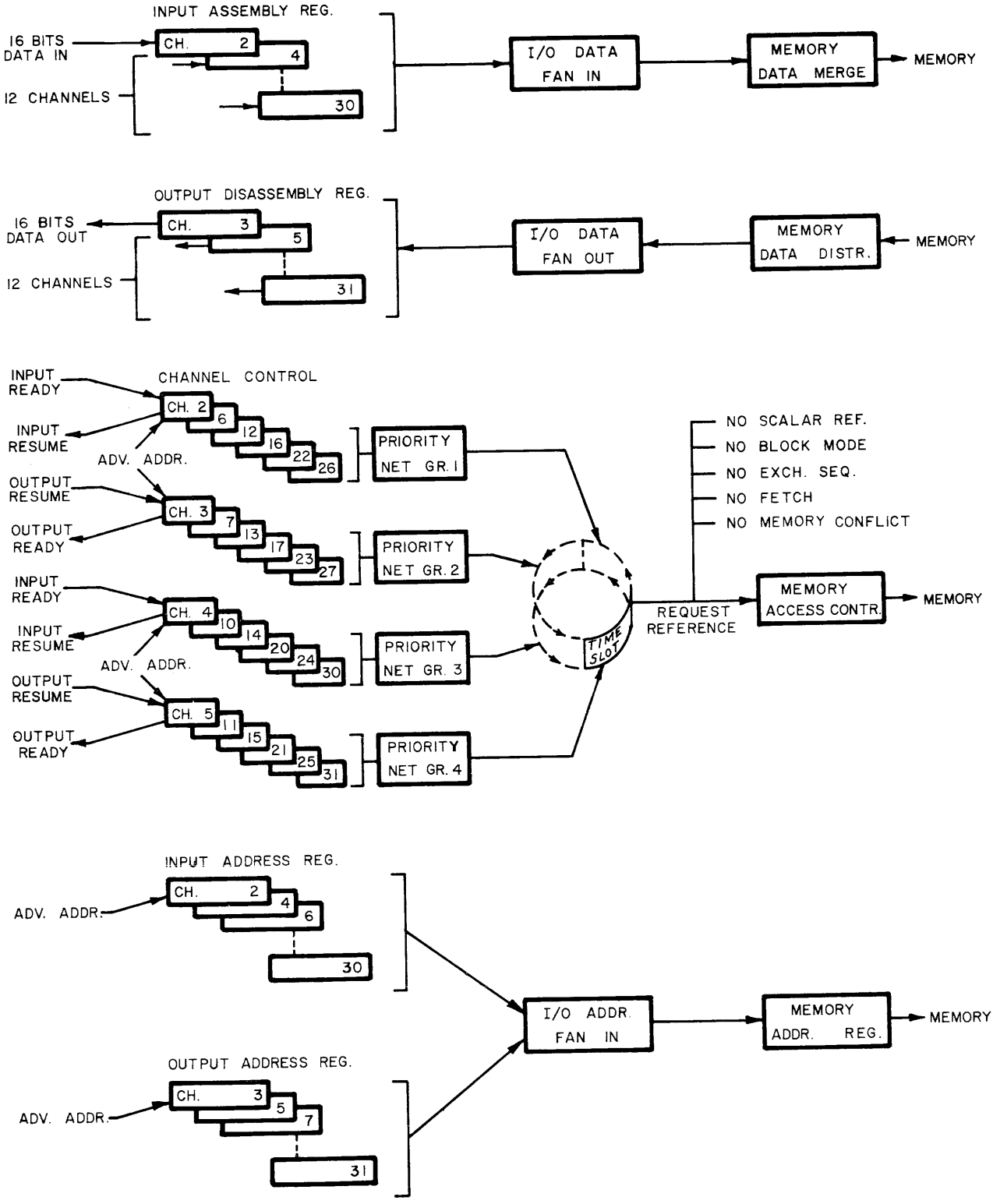


Figure 5-2. Channel I/O control

MEMORY ACCESS

Each of the four channel groups is assigned a time slot (figure 5-2), which is scanned once every 4 CPs for a memory request. The lowest-numbered channel in the group has the highest priority. A memory request, whether accepted or rejected, causes the requesting channel to miss the next time slot. Therefore, any given channel can request a memory reference only once every 8 CPs. However, another channel in the same group as a channel that has just made a memory request can cause a memory request 4 CPs later. During the next 3 CPs, the scanner allows requests from the other three channel groups. Therefore, it is possible to have an I/O memory request every clock period.

I/O LOCKOUT

An I/O memory request can be locked out by a transfer using the B, T, or V registers. Multiple transfers of these types cannot issue without allowing one waiting I/O reference to complete. The maximum duration of a lockout caused by these types of transfers is one block length.

Exchange sequences and instruction fetch sequences can also cause lockouts.

MEMORY BANK CONFLICTS

Memory bank conflicts are tested for CPU scalar references and I/O memory references. All other memory references (block transfers, exchange sequences, instruction fetch sequences) delay issue until all memory banks are quiet. When a block transfer, exchange sequence, or instruction fetch sequence has issued, all other memory references are locked out.

Each memory bank can accept a new request every 4 CPs. To test for a memory bank conflict, the 4 low-order bits[§] of the memory address move through three registers staying 1 CP in each register. The first register is rank A, the second is rank B, and the third is rank C. In the fourth clock period, the address is placed in the memory address register.

I/O MEMORY CONFLICTS

Before testing for a memory bank conflict, a check is made to ensure that no block transfer, exchange sequence, or instruction fetch sequence is in progress, and that no address or scalar instruction requiring a memory reference is in its second clock period of execution. If any of these conditions exist, the I/O request is blocked and must be resubmitted 8 CPs later. The 4 low-order address bits[§] of an I/O reference are tested against address bits in ranks A, B, and C. Coincidence with rank A, B, or C disallows the I/O request. These ranks may be holding previous scalar or I/O memory requests. An I/O request that is disallowed must wait 8 CPs before it can request again.

I/O MEMORY REQUEST CONDITIONS

The following conditions must be present for an I/O Memory request to be processed:

1. I/O request
2. No coincidence in rank A, B, or C
3. No address or scalar instruction requiring a memory reference in CP 2 of execution
4. No fetch request
5. No 176, 177 or 034 through 037 in process
6. No exchange sequence
7. No 033 request

I/O MEMORY ADDRESSING

All I/O memory references are absolute. The current address and limit address registers are 22 bits, allowing I/O access to all of memory. Setting of the current address and limit address registers is limited to monitor mode. I/O Memory reference addresses are not checked for range errors.

---

§ Three bits for 8-bank phasing; refer to part 2, section 2 of this manual.

CPU INSTRUCTIONS 6

INSTRUCTION FORMAT

Each instruction is either a 1-parcel (16-bit) instruction or a 2-parcel (32-bit) instruction. Instructions are packed four parcels per word. Parcels in a word are numbered from left to right as 0 through 3 and can be addressed in branch instructions. A 2-parcel instruction may begin in any parcel of a word and may span a word boundary. A 2-parcel instruction that begins in the fourth parcel of a word ends in the first parcel of the next word. No padding to word boundaries is required.

Figure 6-1 illustrates the general form of instructions:

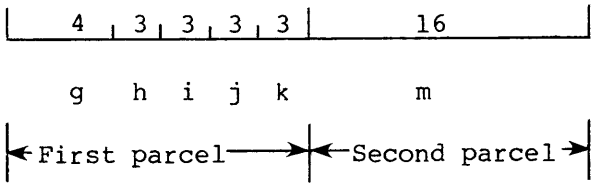


Figure 6-1. General form for instructions

Five variants of this general format use the fields in different ways. Two of these variant forms are 2-parcel formats, two are 1-parcel formats, and one is either a 1-parcel or a 2-parcel format.

ARITHMETIC, LOGICAL FORMAT

For arithmetic and logical instructions, a 7-bit operation code (gh) is followed by three 3-bit address fields. The first field, i, designates the result register. The j and k fields designate the two operand registers or are combined to designate a 6-bit B or T register address. This format is shown in figure 6-2.

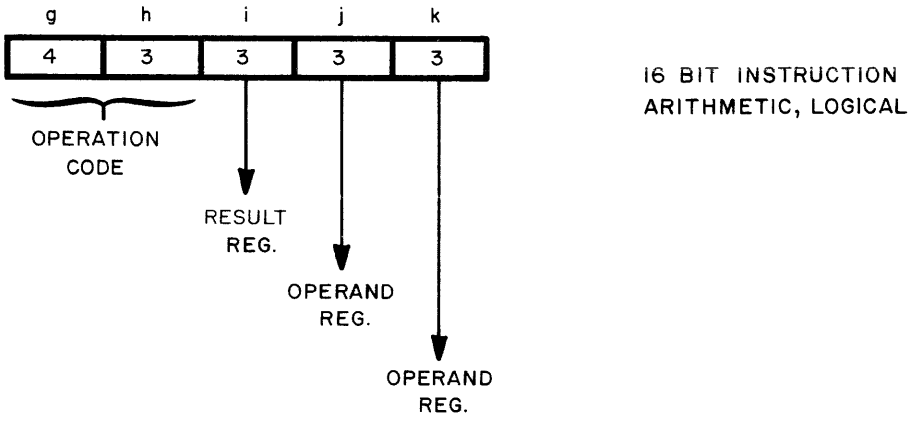


Figure 6-2. Format for arithmetic and logical instructions

SHIFT, MASK FORMAT

The shift and mask instructions consist of a 7-bit operation code (gh) followed by a 3-bit field and a 6-bit field. The 3-bit i field designates the result and operand registers. The 6-bit combined jk field specifies a shift or mask count. This format is shown in figure 6-3.

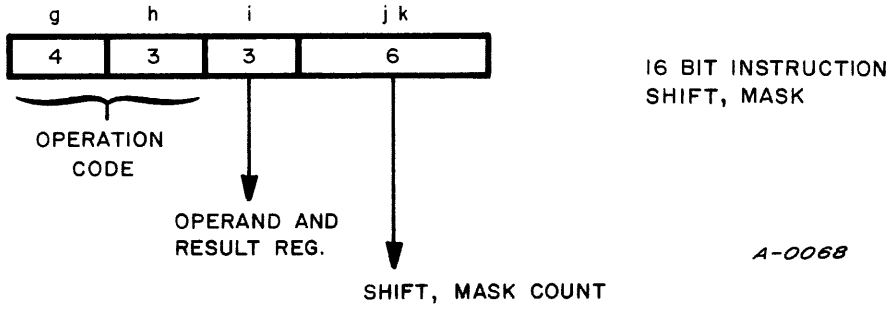


Figure 6-3. Format for shift and mask instructions

IMMEDIATE CONSTANT FORMAT

The instructions that enter immediate constants into A registers have either a 1-parcel or a 2-parcel form. Only the 2-parcel form exists

for entering immediate constants into S registers. For the 1-parcel form, the j and k fields are combined to give a 6-bit quantity. For the 2-parcel form, the j, k, and m fields are combined to give a 22-bit quantity. In either form, a 7-bit operation code (gh) and a 3-bit result field designating a result register precede the immediate constant. The instruction format for immediate constant instructions is shown in figure 6-4.

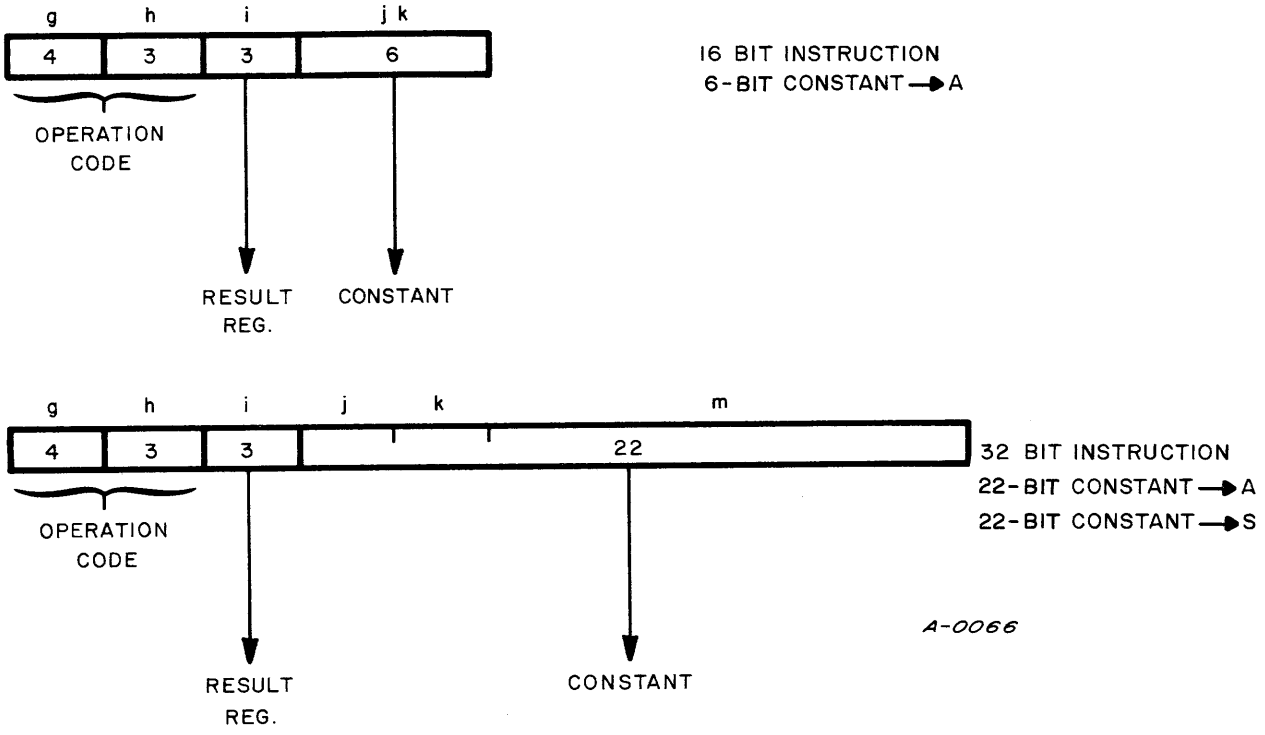


Figure 6-4. Format for immediate constant instructions

MEMORY TRANSFER FORMAT

Instructions that transfer data between the A or S registers and memory require a 32-bit format. For these instructions, a 4-bit operation code (g) is followed by two 3-bit fields and a 22-bit address field. The first 3-bit field (h) designates an index register (Ah).

When the h field is 0, the special value of 0 is considered to be the address index. Contents of Ah are not affected. The second 3-bit field (i) designates a result or source register. The 22-bit field formed by j, k, and m, specifies a memory address displacement value. Figure 6-5 illustrates the format of memory transfer instructions.

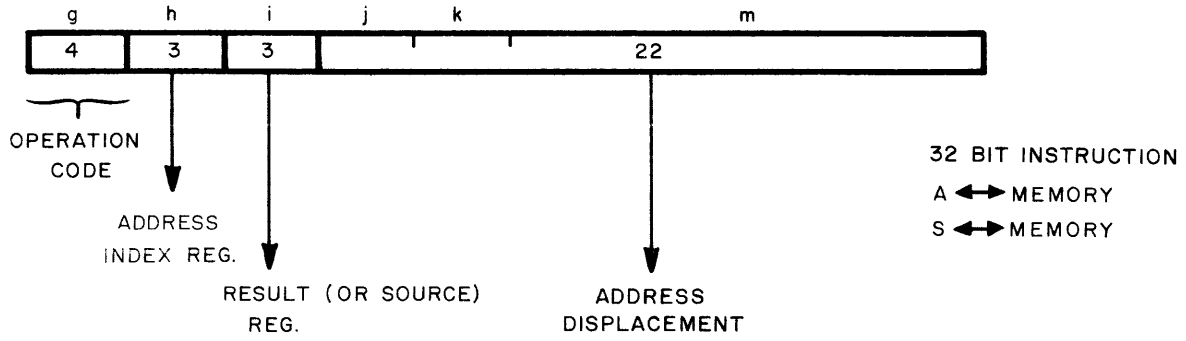


Figure 6-5. Format for memory transfer instructions

BRANCH FORMAT

In general, the branch instructions are 2-parcel instructions. A 7-bit operation code (gh) is followed by a 24-bit field formed by combining i, j, k, and m. The high-order bit of the i field is unused. The 24-bit field contains a parcel address and allows branching to a parcel. Figure 6-6 illustrates the 2-parcel format for branch instructions.

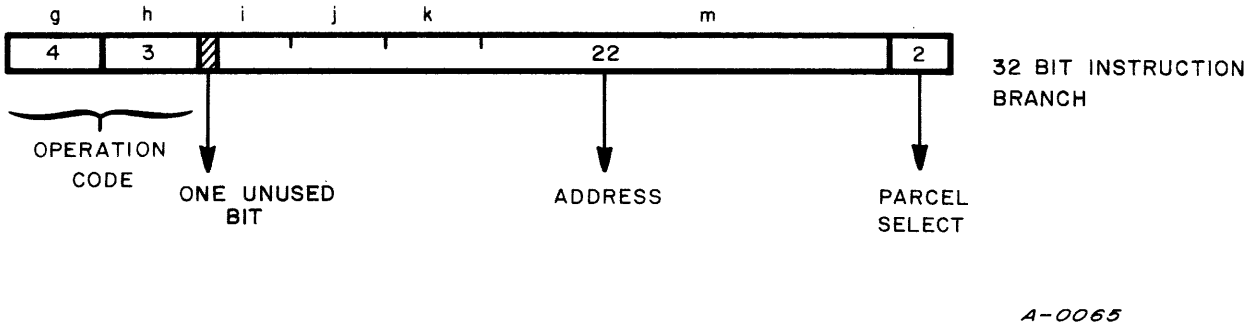


Figure 6-6. Two-parcel format for branch instructions

The unconditional branch to (Bjk) instruction uses only the upper parcel. For this instruction, there is a 7-bit operation code (gh) followed by a null i field and a combined jk field which specifies a B register that contains a parcel address. The m field is ignored.

SPECIAL REGISTER VALUES

The S_0 and A_0 registers provide special values if referenced in the j or k fields of an instruction. In these cases, the special value is used as the operand and the actual value of the S_0 or A_0 register is ignored. The special value is available regardless of existing A_0 or S_0 reservations. This use does not alter the actual value of the S_0 or A_0 register. If S_0 or A_0 is used in the i field, as the operand, the actual value of the register is provided.

Field	Operand value
Ai, i = 0	(A_0)
Aj, j = 0	0
Ak, k = 0	1
Si, i = 0	(S_0)
Sj, j = 0	0
Sk, k = 0	2^{63}
Ah, h = 0	0

INSTRUCTION ISSUE

Instructions are read a parcel at a time from the instruction buffers and delivered to the Next Instruction Parcel (NIP) register. The instruction is passed to the Current Instruction Parcel (CIP) register when the previous instruction issues. An instruction in CIP issues when the conditions in the functional units and registers are such that the functions required for execution may be performed without conflicting with a previously issued instruction. Instruction parcels may issue out of the CIP register at a maximum rate of one per clock period. Once an instruction has been delivered to the CIP, it must be completed in a fixed time frame following its final clock period in the CIP register. No delays are allowed from issue to delivery of data to the destination operating registers, except for scalar memory access instructions (10h and 12h).

Entry to the NIP is blocked for the second half of a 2-parcel instruction. The parcel is delivered to the Last Intruction Parcel (LIP) register, instead. The blank NIP for the second parcel is issued as a do-nothing instruction when it reaches the CIP.

When special register values (A_0 or S_0) are selected by an instruction for Ah, Aj, Ak, Sj or Sk, the normal "hold issue until operand ready" conditions do not apply. These values are always immediately available.

INSTRUCTION DESCRIPTIONS

This section contains detailed information about individual instructions or groups of related instructions. Descriptions are presented with the CRAY-1 Assembler Language (CAL) syntax and in the octal code sequence defined by the gh fields. Each subsection begins with boxed information consisting of the format and a brief summary of each instruction described in the subsection. The appearance of an m in a format designates that the instruction consists of two parcels. An x in the format signifies that the field containing the x is ignored during instruction execution.

Following the header information is a more detailed description of the instruction or instructions, including a list of hold issue conditions, execution time, and special cases. Hold issue conditions refer to those conditions that delay issue of an instruction until the conditions are met.

Instruction issue time assumes that if an instruction issues at CPn, the next instruction may issue at CPn + issue time if its own issue conditions have been met.

CAL Syntax	Description	Octal Code
ERR	Error exit	000000
ERR exp[§]	Programmer coded error exit	000ijk

A 000 instruction is treated as an error condition and an exchange sequence occurs. The content of the instruction buffers is voided by the exchange sequence. If monitor mode is not in effect, the error exit flag in the F register is set. All instructions issued prior to this instruction are run to completion. When the results of previously issued instructions have arrived at the operating registers, an exchange occurs to the exchange package designated by the contents of the XA register. The program address stored in the exchange on the terminating exchange sequence is advanced by one count from the address of the error exit instruction. The error exit instruction is not generally used in program code. Its purpose is to halt execution of an incorrectly coded program that branches into an unused area of memory or into a data area.

Hold issue conditions

034 - 037 in process

Exchange in process

Execution time

Instruction issue 50 CPs; this time includes an exchange sequence (36 CPs) and an instruction fetch operation (14 CPs).

Special cases

Inhibit instruction issue

Begin exchange sequence

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
CA,Aj Ak	Set the current address (CA) register for the channel indicated by (Aj) to (Ak) and activate the channel	0010jk
CL,Aj Ak	Set the limit address (CL) register for the channel indicated by (Aj) to (Ak)	0011jk
CI,Aj	Clear the interrupt flag and error flag for the channel indicated by (Aj)	0012jx
XA Aj	Enter the XA register with (Aj)	0013jk

These instructions are privileged to monitor mode and provide operations useful to the operating system. Functions are selected through the i designator. The instructions are treated as pass instructions if the monitor mode bit is not set.

When the i designator is 0, 1, or 2, the instruction controls the operation of the I/O channels. Each channel has two registers that direct the channel activity. The CA register for a channel contains the address of the current channel word. The CL register specifies the limit address. In programming the channel, the CL register is initialized first and then setting CA activates the channel. As the transfer continues, CA is incremented toward CL. When (CA) equal (CL), the transfer is complete for words at initial (CA) through (CL)-1. When the j designator is 0 or when the content of Aj is less than 2 or greater than 25, the functions are executed as pass instructions. When the k designator is 0, CA or CL is set to 1.

When the i designator is 3, the instruction transmits bits 2^{11} through 2^{4} of (Aj) to the exchange address (XA) register. When the j designator is 0, the XA register is cleared.

Hold issue conditions

034 - 037 in process

Exchange in process

For 0010 and 0011, Aj or Ak reserved

For 0012 or 0013, Aj reserved

Execution time

Instruction issue, 1 CP

Special cases

If the program is not in monitor mode, the instruction becomes a no-op although all hold issue conditions remain effective.

For 0010, 0011, and 0012:
If j = 0, instruction is a no-op even in monitor mode
If (Aj) is less than 2 or (Aj) is greater than or equal to 31_8, the instruction is a no-op
If k = 0, CA or CL is set to 1

For 0013:
If j = 0, XA register is cleared

Correct priority interrupting channel number can be read (via 033 instruction) 2 CPs after issue of 0012 instruction.

CAL Syntax	Description	Octal Code
RT Sj	Enter the real-time clock register with (Sj)	0014j0
PCI Sj	Enter interrupt interval (II) register with (Sj)	0014j4
CCI	Clear the programmable clock interrupt request	0014j5
ECI	Enable programmable clock interrupt request	0014j6
DCI	Disable programmable clock interrupt requests	0014j7

These instructions perform specialized functions for managing the real-time and programmable clocks. They are privileged to monitor mode. The instructions are treated as pass instructions if the monitor mode bit is not set.

When the k designator is 0, the instruction loads the contents of the Sj register into the real-time clock (RTC) register. When the j designator is 0, the real-time clock register is cleared.

When the k designator is 4, this instruction loads the low-order 32 bits from the Sj register into both the Interrupt Interval (II) register and the Interrupt Countdown (ICD) counter. When the j designator is 0, II and ICD are cleared.

When the k designator is 5, this instruction clears the programmable clock interrupt request if the request was previously set by an interrupt countdown to 0.

When the k designator is 6, this instruction enables repeated programmable clock interrupt requests at a repetition rate determined by the value stored in the Interrupt Interval (II) register.

When the k designator is 7, this instruction disables repeated programmable clock interrupt requests until a 0014j6 instruction is executed to enable the requests.

Hold issue conditions

034 - 037 in process

Exchange in process

Sj reserved

Execution time

Instruction issue, 1 CP

Special cases

If the program is not in monitor mode, these instructions become no-ops but all hold issue conditions remain effective.

For 0014j0 and 0014j4, if j=0, (Sj)=0.

The instruction forms 0015, 0016, 0017 are not implementable in the CRAY-1/S hardware but they act as no-op instructions. There is no CAL syntax for them.

CAL Syntax	Description	Octal Code
VL Ak	Transmit (Ak) to VL register	0020xk
VL 1§	Transmit 1 to VL register	0020x0

This instruction enters the vector length (VL) register with a value determined by the contents of Ak. The low-order 7 bits of (Ak) are entered into the VL register. The number of operations performed in a vector instruction is determined by first subtracting 1 from the contents of VL and then adding 1 to the low-order 6 bits of the result.

For example:

if (VL)=100_8	then	$100_8-1 = 77_8$
	and	$77_8+1 = 100_8$
if (VL)=0	then	$0-1 = 177_8$
	and	$77_8 + 1 = 100_8$

Thus, the number of vector operations is 64 when the content of VL is 0 or 64.

Hold issue conditions

034 - 037 in process

Exchange in process

Ak reserved

Execution time

Instruction issue 1 CP

VL register ready 1 CP

Special cases

Maximum vector length is 64

(Ak) = 1 if k = 0

(VL) = Ø if k = 0 and (Ak) = 0

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
EFI	Enable interrupt on floating-point error	0021xx
DFI	Disable interrupt on floating-point error	0022xx

These instructions set (0021xx) or clear (0022xx) the floating-point mode flag in the M register. They do not check the previous state of the flag (there is no way of testing the flag).

When set, the floating-point mode flag enables interrupts on floating-point range errors as described in section 4.

Hold issue conditions

034 - 037 in process

Exchange in process

Ak reserved

Execution time

Instruction issue 1 CP

Special cases

The 0023, 0024, 0025, 0026 and 0027 instructions are not implemented but act as no-ops. There is no CAL Syntax for them.

CAUTION

The operating system may have status bits that reflect whether interrupts on floating-point range errors are enabled or disabled. If so, those software status bits need to be modified to agree with the floating-point mode flag.

CAL Syntax	Description	Octal Code
VM Sj	Transmit (Sj) to VM register	003xjx
VM 0§	Clear VM register	003x0x

This instruction enters the vector mask (VM) register with the contents of Sj. The VM register is cleared if the j designator is 0. This instruction is used in conjunction with the vector merge instructions (146 and 147) in which an operation is performed depending on the contents of VM.

Hold issue conditions

034 - 037 in process

Exchange in process

Sj reserved

003 in process - unit busy 3 CPs

14x in process - unit busy (VL) + 4 CPs

175 in process - unit busy (VL) + 4 CPs

Execution time

Instruction issue 1 CP

VM ready in 3 CPs except for use in 073 instruction

VM ready in 6 CPs for 073 instruction

Special cases

(Sj) = 0 if j = 0

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
EX	Normal exit	004xxx
EX exp[§]	Normal exit, programmer encoded	004ijk

This instruction causes an exchange sequence. The contents of the instruction buffers are voided by the exchange sequence. If monitor mode is not in effect, the normal exit flag in the F register is set. All instructions issued prior to this instruction are run to completion. When all results have arrived at the operating registers as a result of previously issued instructions, an exchange sequence occurs to the exchange package designated by the contents of the XA register. The program address stored in the exchange package is advanced one count from the address of the normal exit instruction. This instruction is used to issue a monitor request from a user program.

Hold issue conditions

034 - 037 in process

Exchange in process

Execution time

Instruction issue 50 CPs; this time includes an exchange sequence (36 CPs) and an instruction fetch operation (14 CPs)

Special cases

Inhibit instruction issue

Begin exchange sequence

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
J Bjk	Branch to (Bjk)	005xjk

This instruction sets the P register to the 24-bit parcel address specified by the contents of Bjk causing execution to continue at that address. The instruction is used to return from a subroutine.

Hold issue conditions

034 - 037 in process

Exchange in process

Execution time

Instruction issue:

Instruction parcel and following parcel both in a buffer and branch address in a buffer, 7 CPs

Instruction parcel and following parcel both in a buffer and branch address not in a buffer, 16 CPs

Parcel following instruction parcel not in a buffer and branch address in a buffer, 16 CPs

Parcel following instruction parcel not in a buffer and branch address not in buffer, 25 CPs

Special cases

The parcel following an 005 instruction is not used for branching; however, it can cause a delay of the 005 instruction if it is out of buffer. See execution times.

CAL Syntax	Description	Octal Code
J exp	Branch to ijkm	006ijkm

This 2-parcel instruction sets the P register to the parcel address specified by the low-order 24 bits of the ijkm field. Execution continues at that address. The high-order bit of the ijkm field is ignored.

Hold issue conditions

034 - 037 in process

Exchange in process

Execution time

Instruction issue:

Both parcels of instruction in the same buffer and branch address in a buffer, 5 CPs

Both parcels of instruction in the same buffer and branch address not in a buffer, 14 CPs

Both parcels of instruction in different buffers and branch address in a buffer, 7 CPs

Both parcels of instruction in different buffers and branch address not in a buffer, 16 CPs

Second parcel of instruction not in a buffer and branch address in a buffer, 16 CPs

Second parcel of instruction not in a buffer and branch address not in a buffer, 25 CPs

Special cases

None

CAL Syntax	Description	Octal Code
R exp	Return jump to ijkm; set B_{00} to (P)	007ijkm

This 2-parcel instruction sets register B_{00} to the address of the following parcel. The P register is then set to the parcel address specified by the low-order 24 bits of the ijkm field. Execution continues at that address. The high-order bit of the ijkm field is ignored. The purpose of this instruction is to provide a return linkage for subroutine calls. The subroutine is entered via a return jump. The subroutine returns to the caller at the instruction following the call by executing a branch to the contents of the B_{00} register.

Hold issue conditions

034 - 037 in process

Exchange in process

Execution time

Instruction issue:

Both parcels of instruction in the same buffer and branch address in a buffer, 5 CPs

Both parcels of instruction in the same buffer and branch address not in a buffer, 14 CPs

Both parcels of instruction in different buffers and branch address in a buffer, 7 CPs

Both parcels of instruction in different buffers and branch address not in a buffer, 16 CPs

Second parcel of instruction not in a buffer and branch address in a buffer, 16 CPs

Second parcel of instruction not in a buffer and branch address not in a buffer, 25 CPs

Special cases

None

CAL Syntax	Description	Octal Code
JAZ exp	Branch to ijkm if (A_0) = 0	010ijkm
JAN exp	Branch to ijkm if (A_0) ≠ 0	011ijkm
JAP exp	Branch to ijkm if (A_0) positive	012ijkm
JAM exp	Branch to ijkm if (A_0) negative	013ijkm

These 2-parcel instructions test the contents of A_0 for the condition specified by the h field. If the condition is satisfied, the P register is set to the parcel address specified by the low-order 24 bits of the ijkm field and execution continues at that address. The high-order bit of the ijkm field is ignored. If the condition is not satisfied, execution continues with the instruction following the branch instruction.

Hold issue conditions

034 - 037 in process

Exchange in process

A_0 busy in previous 2 CPs

Execution time

Instruction issue:

Both parcels of instruction in the same buffer and branch address in a buffer, 5 CPs

Both parcels of instruction in the same buffer and branch address not in a buffer, 14 CPs

Both parcels of instruction in different buffers and branch address in a buffer, 7 CPs

Both parcels of instruction in different buffers and branch address not in a buffer, 16 CPs

Second parcel of instruction not in a buffer and branch address in a buffer, 16 CPs

010-013

Second parcel of instruction not in a buffer and branch address not in buffer, 25 CPs

Both parcels of instruction in the same buffer and branch not taken, 2 CPs

Both parcels of instruction in different buffers and branch not taken, 4 CPs

Second parcel of instruction not in a buffer and branch not taken, 13 CPs

Special case

$(A_0) = 0$ is considered a positive condition.

CAL Syntax	Description	Octal Code
JSZ exp	Branch to ijkm if $(S_0) = 0$	014ijkm
JSN exp	Branch to ijkm if $(S_0) \neq 0$	015ijkm
JSP exp	Branch to ijkm if (S_0) positive	016ijkm
JSM exp	Branch to ijkm if (S_0) negative	017ijkm

These 2-parcel instructions test the contents of S_0 for the condition specified by the h field. If the condition is satisfied, the P register is set to the parcel address specified by the low-order 24 bits of the ijkm field and execution continues at that address. The high-order bit of the ijkm field is ignored. If the condition is not satisfied, execution continues with the instruction following the branch instruction.

Hold issue conditions

034 - 037 in process

Exchange in process

S_0 busy in previous 2 CPs

Execution time

Instruction issue:

Both parcels of instruction in the same buffer and branch address in a buffer, 5 CPs

Both parcels of instruction in the same buffer and branch address not in a buffer, 14 CPs

Both parcels of instruction in different buffers and branch address in a buffer, 7 CPs

Both parcels of instruction in different buffers and branch address not in a buffer, 16 CPs

Second parcel of instruction not in a buffer and branch address in a buffer, 16 CPs

014-017

Second parcel of instruction not in a buffer and branch address in a buffer, 25 CPs

Both parcels of instruction in the same buffer and branch not taken, 2 CPs

Both parcels of instruction in different buffers and branch not taken, 4 CPs

Second parcel of instruction not in a buffer and branch not taken, 13 CPs

Special case

$(S_0) = 0$ is considered a positive condition.

CAL Syntax	Description	Octal Code
Ai exp	Transmit jkm to Ai	020ijkm
Ai exp	Transmit ones complement of jkm to Ai	021ijkm

The 020 instruction enters into Ai a 24-bit value that is composed of the 22-bit jkm field and 2 high-order bits of 0.

The 021 instruction enters into Ai a 24-bit value that is the complement of a value formed by the 22-bit jkm field and 2 high-order bits of 0. The complement is formed by changing all 1 bits to 0 and all 0 bits to 1. Thus, for the 021 instruction, the high-order 2 bits of Ai are set to 1 and the instruction provides a means of entering a negative value into Ai. The instructions are both 2-parcel instructions.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai reserved

Execution time

Instruction issue:

Both parcels in same buffer, 2 CPs

Both parcel in different buffers, 4 CPs

Second parcel not in a buffer, 13 CPs

Ai ready, 1 CP

Special cases

None

CAL Syntax	Description	Octal Code
Ai exp	Transmit jk to Ai	022ijk

This 1-parcel instruction enters the 6-bit quantity from the jk field into the low-order 6 bits of Ai. The high-order 18 bits of Ai are zeroed. No sign extension occurs.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai reserved

Execution time

Instruction issue, 1 CP

Ai ready, 1 CP

Special cases

None

CAL Syntax	Description	Octal Code
Ai Sj	Transmit (Sj) to Ai	023ijx

This instruction enters the low-order 24 bits of (Sj) into Ai. The high-order bits of (Sj) are ignored.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai reserved

Sj reserved

Execution time

Instruction issue, 1 CP

Ai ready, 1 CP

Special case

(Sj) = 0 if j = 0

CAL Syntax		Description	Octal Code
Ai Bjk	024ijk	Transmit (Bjk) to Ai	024ijk
Bjk Ai	025ijk	Transmit (Ai) to Bjk	025ijk

The 024 instruction enters the contents of Bjk into Ai.

The 025 instruction enters the contents of Ai into Bjk.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict (024 only)

Ai reserved

Execution time

For 024, Ai ready, 1 CP

Instruction issue for 024 or 025, 1 CP

Special cases

None

CAL Syntax	Description	Octal Code
Ai PSj	Population count of (Sj) to Ai	026ij0
Ai QSj	Population count parity of (Sj) to Ai	026ij1

The 026ij0 instruction counts the number of bits set to 1 in (Sj) and enters the result into the low-order 7 bits of Ai. The high-order 17 bits of Ai are zeroed.

The 026ij1 instruction counts the number of bits set to 1 in (Sj). Then, the low-order bit, which shows the odd/even state of the result is transferred to the low-order bit position of the Ai register. The high-order 23 bits are cleared. The actual population count is not transferred.

The instructions are executed in the population/leading zero count unit.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai reserved

Sj reserved

Execution time

Instruction issue, 1 CP

Ai ready, 4 CPs

Special case

(Ai) = 0 if j = 0

CAL Syntax	Description	Octal Code
Ai ZSj	Leading zero count of (Sj) to Ai	027ijx

This instruction counts the number of leading zeros in Sj and enters the result into the low-order 7 bits of Ai. The high-order 17 bits of Ai are zeroed.

The instruction is executed in the population/leading zero count unit.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai reserved

Sj reserved

Execution time

Instruction issue, 1 CP

Ai ready, 3 CPs

Special cases

(Ai) = 64 if j = 0

(Ai) = 0 if (Sj) is negative

CAL Syntax		Description	Octal Code
Ai	Aj+Ak	Integer sum of (Aj) and (Ak) to Ai	030ijk
Ai	Ak§	Transmit (Ak) to Ai	030i0k
Ai	Aj+1§	Integer sum of (Aj) and 1 to Ai	030ij0
Ai	Aj-Ak	Integer difference (Aj) less (Ak) to Ai	031ijk
Ai	-1§	Transmit -1 to Ai	031i00
Ai	-Ak§	Transmit the negative of (Ak) to Ai	031i0k
Ai	Aj-1§	Integer difference (Aj) less 1 to Ai	031ij0

These instructions are executed in the address add unit.

The 030 instruction forms the integer sum of (Aj) and (Ak) and enters the result into Ai. No overflow is detected.

The 031 instruction forms the integer difference of (Aj) and (Ak) and enters the result into Ai. No overflow is detected.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai, Aj, or Ak reserved

Execution time

Instruction issue, 1 CP

Ai ready, 2 CPs

§ Special CAL syntax form

030-031

Special cases

For 030:

(Ai)= (Ak)	if j = 0 and k ≠ 0
(Ai)= 1	if j = 0 and k = 0
(Ai)= (Aj)+1	if j ≠ 0 and k = 0

For 031:

(Ai)= -(Ak)	if j = 0 and k ≠ 0
(Ai)= -1	if j = 0 and k = 0
(Ai)= (Aj)-1	if j ≠ 0 and k = 0

CAL Syntax	Description	Octal Code
Ai Aj*Ak	Integer product of (Aj) and (Ak) to Ai	032ijk

This instruction forms the integer product of (Aj) and (Ak) and enters the low-order 24 bits of the result into Ai. No overflow is detected.

This instruction is executed in the address multiply unit.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai, Aj, or Ak reserved

Execution time

Instruction issue, 1 CP

Ai ready, 6 CPs

Special cases

(Ai) = 0 if j = 0

(Ak) = 1 if k = 0

Thus

(Ai) = (Aj) if $j \neq 0$ and k = 0

CAL Syntax		Description	Octal Code
Ai	CI	Channel number of highest priority interrupt request to Ai	033i0x
Ai	CA,Aj	Current address of channel (Aj) to Ai	033ij0
Ai	CE,Aj	Error flag of channel (Aj) to Ai	033ij1

This instruction enters channel status information into Ai. The j and k designators and the contents of Aj define the desired information.

The channel number of the highest priority interrupt request is entered into Ai when the j designator is 0. The contents of Aj specifies a channel number when the j designator is nonzero. The value of the current address (CA) register for the channel is entered into Ai when the k designator is 0. The error flag for the channel is entered into the low-order bit of Ai when the k designator is 1. The high-order bits of Ai are cleared. The error flag can be cleared only in monitor mode using the 0012 instruction.

The 033 instruction does not interfere with channel operation.

Hold issue conditions

034 - 037 in process

Exchange in process

A register access conflict

Ai reserved

Aj reserved

Execution time

Instruction issue, 1 CP

Ai ready, 4 CPs

Special cases

(Ai) = highest priority channel causing interrupt if (Aj) = 0

(Ai) = current address of channel (Aj) if

(Aj) ≠ 0 and k = 0

(Ai) = I/O error flag of channel (Aj) if

(Aj) ≠ 0 and k = 1

(Ai) = 0 if (Aj) = 1

2 CPs must elapse after an 0012jx instruction issue before issuing an 033i0x instruction.

CAL Syntax	Description	Octal Code
Bjk,Ai ,A0	Block transfer (Ai) words from memory starting at address (A_0) to B registers starting at register jk	034ijk
Bjk,Ai 0,A0§	Block transfer (Ai) words from memory starting at address (A_0) to B registers starting at register jk	034ijk
,A0 Bjk,Ai	Block transfer (Ai) words from B registers starting at register jk to memory starting at address (A_0)	035ijk
0,A0 Bjk,Ai§	Block transfer (Ai) words from B registers starting at register jk to memory starting at address (A_0)	035ijk
Tjk,Ai ,A0	Block transfer (Ai) words from memory starting at address (A_0) to T registers starting at register jk	036ijk
Tjk,Ai 0,A0§	Block transfer (Ai) words from memory starting at address (A_0) to T registers starting at register jk	036ijk
,A0 Tjk,Ai	Block transfer (Ai) words from T registers starting at register jk to memory starting at address (A_0)	037ijk
0,A0 Tjk,Ai§	Block transfer (Ai) words from T registers starting at register jk to memory starting at address (A_0)	037ijk

These instructions perform block transfers between memory and B or T registers.

In all of the instructions, the amount of data transferred is specified by the low-order 7 bits of (Ai). See special cases for details.

The first register involved in the transfer is specified by jk. Successive transfers involve successive B or T registers until B_{77} or T_{77} is reached. Since processing of the registers is circular, B_{00} will be processed after B_{77} and T_{00} will be processed after T_{77} if the count in (Ai) is not exhausted.

§ Special CAL syntax form

The first memory location referenced by the transfer instruction is specified by (A_0). The A_0 register contents are not altered by execution of the instruction. Memory references are incremented by 1 for successive transfers.

For transfers of B registers to memory, each 24-bit value is right adjusted in the word, the high-order 40 bits are zeroed. When transferring from memory to B registers, only the low-order 24 bits are transmitted; the high-order 40 bits are ignored.

Hold issue conditions

A0 through A_7 reserved (034, 036)

A0 Ai, or S_0 through S_7 reserved (035, 037)

Block sequence flag set (034 - 037, 176, 177)

034 - 037 in process

Exchange in process

Scalar reference in CP 2

Rank B data valid

Fetch request in previous clock period

I/O memory request

Execution time

For 034, 036:

Instruction issue 14 CPs + (Ai) if (Ai) $\neq$ 0; 5 CPs if (Ai) = 0

For 035, 037:

Instruction issue 6 CPs + (Ai) if (Ai) $\neq$ 0; 7 CPs if (Ai) = 0

Special cases

1. Block all issues when in process.
2. Block all I/O references.

3. (Ai) = 0 causes a zero-block transfer.

   (Ai) in the range greater than 100_8 and less than 200_8 causes a wrap-around condition.

   If (Ai) is greater than 177_8, bits 2^7 through 2^{23} are truncated. The block length is equal to the value of 2^0 through 2^6.

4. (A_0) is used as the block length if i = 0.

CAL Syntax	Description	Octal Code
Si exp	Transmit jkm to Si	040ijkm
Si exp	Transmit complement of jkm to Si	04lijkm

These 2-parcel instructions provide for entering immediate values into an S register.

The 040 instruction enters into Si a 64-bit value that is composed of the 22-bit jkm field and 42 high-order bits of 0.

The 04l instruction enters into Si a 64-bit value that is the complement of a value formed by the 22-bit jkm field and 42 high-order bits of 0. The complement is formed by changing all 1 bits to 0 and all 0 bits to 1. Thus, for the 04l instruction, the high-order 42 bits of Si are set to 1 and the instruction provides for entering a negative value into Si.

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si reserved

Execution time

Instruction issue:

Both parcels in same buffer, 2 CPs

Both parcels in different buffers, 4 CPs

Second parcel not in a buffer, 13 CPs

Si ready, 1 CP

Special cases

None

CAL Syntax	Description	Octal Code
Si <exp	Form exp = 64-jk bits of ones mask in Si from right	042ijk
Si # >exp§	Form exp = jk bits of zeros mask in Si from left	042ijk
Si 1§	Enter 1 into Si	042i77
Si -1§	Enter -1 into Si	042i00
Si >exp	Form exp = jk bits of ones mask in Si from left	043ijk
Si # <exp§	Form exp = 64-jk bits of zeros mask in Si from left	043ijk
Si 0§	Clear Si	043i00

The 042 instruction generates a mask of 64-jk ones from right to left in Si. For example, if jk = 0, Si contains all 1 bits and if jk = 77_8, Si contains zeros in all but the low-order bit.

The 043 instruction generates a mask of jk ones from left to right in Si. For example, if jk = 0, Si contains all 0 bits and if jk = 77_8, Si contains ones in all but the low-order bit.

These instructions are executed in the scalar logical unit.

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si reserved

Execution time

Instruction issue, 1 CP

Si ready, 1 CP

Special cases

None

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
Si Sj&Sk	Logical product of (Sj) and (Sk) to Si	044ijk
Si Sj&SB§	Sign bit of (Sj) to Si	044ij0
Si SB&Sj§	Sign bit of (Sj) to Si (j ≠ 0)	044ij0
Si #Sk&Sj	Logical product of (Sj) and complement of (Sk) to Si	045ijk
Si #SB&Sj§	(Sj) with sign bit cleared to Si	045ij0
Si Sj Sk	Logical difference of (Sj) and (Sk) to Si	046ijk
Si Sj SB§	Toggle sign bit of (Sj), then enter into Si	046ij0
Si SB Sj§	Toggle sign bit of (Sj); then enter into Si (j ≠ 0)	046ij0
Si #Sj Sk	Logical equivalence of (Sk) and (Sj) to Si	047ijk
Si #Sk§	Transmit ones complement of (Sk) to Si	047i0k
Si #Sj\SB§	Logical equivalence of (Sj) and sign bit to Si	047ij0
Si #SB\Sj§	Logical equivalence of (Sj) and sign bit to Si (j ≠ 0)	047ij0
Si #SB§	Enter ones complement of sign bit into Si	047i00
Si Sj!Si&Sk	Scalar merge	050ijk
Si Sj!Si&SB§	Scalar merge of (Si) and sign bit of (Sj) to Si	050ij0
Si Sj!Sk	Logical sum of (Sj) and (Sk) to Si	051ijk
Si Sk§	Transmit (Sk) to Si	051i0k
Si Sj!SB§	Logical sum of (Sj) and sign bit to Si	051ij0
Si SB!Sj§	Logical sum of (Sj) and sign bit to Si (j ≠ 0)	051ij0
Si SB§	Enter sign bit into Si	051i00

§ Special CAL syntax

These instructions are executed in the scalar logical unit.

The 044 instruction forms the logical product (AND) of (Sj) and (Sk) and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of (Sj) and (Sk) are 1 as in the following example:

```
(Sj) = 1 1 0 0
(Sk) = 1 0 1 0
(Si) = 1 0 0 0
```

(Sj) is transmitted to Si if the j and k designators have the same non-zero value. Si is cleared if the j designator is 0. The sign bit of (Sj) is extracted into Si if the j designator is non-zero and the k designator is 0.

The 045 instruction forms the logical product (AND) of (Sj) and the complement of (Sk) and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of (Sj) and the complement of (Sk) are 1 as in the following example:

```
(Sj)  = 1 1 0 0
(Sk)  = 1 0 1 0
(Sk') = 0 1 0 1
(Si)  = 0 1 0 0
```

where (Sk') = complement of (Sk)

Si is cleared if the j and k designators have the same value or if the j designator is 0. (Sj) with the sign bit cleared is transmitted to Si if the j designator is non-zero and the k designator is 0.

The 046 instruction forms the logical difference (exclusive OR) of (Sj) and $\overline{(Sk)}$ and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of (Sj) and (Sk) are different as in the following example:

```
(Sj) = 1 1 0 0
(Sk) = 1 0 1 0
(Si) = 0 1 1 0
```

Si is cleared if the j and k designators have the same non-zero value. (Sk) is transmitted to Si if the j designator is 0 and the k designator is non-zero. The sign bit of (Sj) is complemented and the result is transmitted to Si if the j designator is non-zero and the k designator is 0.

The 047 instruction forms the logical equivalence of (Sj) and (Sk), and enters the result into Si. Bits of Si are set to 1 when the corresponding bits of (Sj) and (Sk) are the same as in the following example:

```
(Sj) = 1 1 0 0
(Sk) = 1 0 1 0
(Si) = 1 0 0 1
```

Si is set to all ones if the j and k designators have the same non-zero value. The complement of (Sk) is transmitted to Si if the j designator is 0 and the k designator is non-zero. All bits except the sign bit of (Sj) are complemented and the result is transmitted to Si if the j designator is non-zero and the k designator is 0. The result is the complement of that produced by the 046 instruction.

The 050 instruction merges the contents of (Sj) with (Si) depending on the ones mask in Sk. The result is defined by the following Boolean equation:

$$(Si) = (Sj)(Sk) + (Si)(Sk')$$

where Sk' is the complement of Sk as illustrated:

```
(Sk)  = 1 1 1 1 0 0 0 0
(Sk') = 0 0 0 0 1 1 1 1
(Si)  = 1 1 0 0 1 1 0 0
(Sj)  = 1 0 1 0 1 0 1 0
(Si)  = 1 0 1 0 1 1 0 0
```

The 050 instruction is intended for merging portions of 64-bit words into a composite word. Bits of Si are cleared when the corresponding bits of Sk are 1 if the j designator is 0 and the k designator is non-zero. The sign bit of (Sj) replaces the sign bit of Si if the j designator is non-zero and the k designator is 0. The sign bit of Si is cleared if the j and k designators are both 0.

The 051 instruction forms the logical sum (inclusive OR) of (Sj) and (Sk) and enters the result into Si. Bits of Si are set when 1 of the corresponding bits of (Sj) and (Sk) is set as in the following example:

```
(Sj) = 1 1 0 0
(Sk) = 1 0 1 0
(Si) = 1 1 1 0
```

044-051

(Sj) is transmitted to Si if the j and k designators have the same non-zero value. (Sk) is transmitted to Si if the j designator is 0 and the k designator is non-zero. (Sj) with the sign bit set to 1 is transmitted to Si if the j designator is non-zero and the k designator is 0. A ones mask consisting of only the sign bit is entered into Si if the j and k designators are both 0.

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si, Sj, and Sk reserved

Execution time

Instruction issue, 1 CP

Si ready, 1 CP

Special cases

(Sj) = 0 if j = 0

(Sk) = 2^{63} if k = 0

CAL Syntax	Description	Octal Code
S0 Si <exp	Shift (Si) left exp = jk places to S_0	052ijk
S0 Si >exp	Shift (Si) right exp = 64-jk places to S_0	053ijk
Si Si <exp	Shift (Si) left exp = jk places to Si	054ijk
Si Si >exp	Shift (Si) right exp = 64-jk places to Si	055ijk

These instructions are executed in the scalar shift unit. They shift values in an S register by an amount specified by jk. All shifts are end-off with zero fill.

The 052 instruction shifts (Si) left jk places and enters the result into S_0.

The 053 instruction shifts (Si) right by 64-jk places and enters the result into S_0.

The 054 instruction shifts (Si) left jk places and enters the result into Si.

The 055 instruction shifts (Si) right by 64-jk places and enters the result into Si.

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si reserved

S_0 reserved (052 and 053 only)

Execution time

Instruction issue, 1 CP

For 052, 053, S_0 ready, 2 CPs

For 054, 055, Si ready, 2 CPs

Special cases

None

CAL Syntax	Description	Octal Code
Si Si,Sj<Ak	Shift (Si) and (Sj) left by (Ak) places to Si	056ijk
Si Si,Sj<1§	Shift (Si) and (Sj) left one place to Si	056ij0
Si Si<Ak§	Shift (Si) left (Ak) places to Si	056i0k
Si Sj,Si>Ak	Shift (Sj) and (Si) right by (Ak) places to Si	057ijk
Si Sj,Si>1§	Shift (Sj) and (Si) right one place to Si	057ij0
Si Si>Ak§	Shift (Si) right (Ak) places to Si	057i0k

These instructions are executed in the scalar shift unit. They shift 128-bit values formed by logically joining two S registers. Shift counts are obtained from register Ak. A shift of one place occurs if the k designator is 0.

All shifts are end-off with zero fill if i ≠ j. The shift is circular if the shift count does not exceed 64 and the i and j designators are equal and nonzero. For both the 056 and 057 instructions, (Sj) are unchanged, provided i ≠ j.

The 056 instruction performs left shifts of (Si) and (Sj) with (Si) initially the most significant bits of the double register. The high-order 64 bits of the result are transmitted to Si. Si is cleared if the shift count exceeds 127. The 056 instruction produces the same result as the 054 instruction if the shift count does not exceed 63 and the j designator is 0.

The 057 instruction performs right shifts of (Sj) and (Si) with (Sj) initially the most significant bits of the double register. The low-order 64 bits of the result are transmitted to Si. Si is cleared if the shift count exceeds 127. The 057 instruction produces the same result as the 055 instruction if the shift count does not exceed 63 and the j designator is 0.

All shifts counts, (Ak), are considered positive. All 24 bits of (Ak) are used for the shift count.

§ Special CAL syntax form

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si, Sj, or Ak reserved

Execution time

Instruction issue, 1 CP

Si ready, 3 CPs

Special cases

(Sj) = 0 if j = 0

(Ak) = 1 if k = 0

Circular shift if i = j ≠ 0 and (Ak) less than 64.

CAL Syntax		Description	Octal Code
Si	Sj+Sk	Integer sum of (Sj) and (Sk) to Si	060ijk
Si	Sj-Sk	Integer difference of (Sj) and (Sk) to Si	061ijk
Si	-Sk§	Transmit negative of (Sk) to Si	061i0k

These instructions are executed in the scalar add unit.

The 060 instruction forms the integer sums of (Sj) and (Sk) and enters the result into Si. No overflow is detected.

The 061 instruction forms the integer difference of (Sj) and (Sk) and enters the result into Si. No overflow is detected.

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si, Sj, or Sk reserved

Execution time

Si ready, 3 CPs

Instruction issue, 1 CP

Special cases

(Si) = 2^{63} if j = 0 and k = 0

For 060:

(Si) = (Sk) if j = 0 and k ≠ 0

(Si) = (Sj) with 2^{63} complemented if j = 0 and k ≠ 0

For 061:

(Si) = -(Sk) if j = 0 and k ≠ 0

(Si) = (Sj) with 2^{63} complemented if j ≠ 0 and k = 0

§ Special CAL syntax form

CAL Syntax		Description	Octal Code
Si	Sj+FSk	Floating sum of (Sj) and (Sk) to Si	062ijk
Si	+FSk§	Normalize (Sk) to Si	062i0k
Si	Sj-FSk	Floating difference of (Sj) and (Sk) to Si	063ijk
Si	-FSk§	Transmit normalized negative of (Sk) to Si	063i0k

These instructions are performed by the floating-point add unit. Operands are assumed to be in floating-point format. The result is normalized even if the operands are not. Underflow and overflow conditions are described in section 4.

The 062 instruction forms the sum of the floating-point quantities in Sj and Sk and enters the normalized result into Si.

The 063 instruction forms the difference of the floating-point quantities in Sj and Sk and enters the normalized result into Si.

Hold issue conditions

034 - 037 in process

Exchange in process

Si register access conflict

Si, Sj, or Sk reserved

170 - 173 in process; unit busy (VL) + 4 CPs

Execution time

Instruction issue, 1 CP

Si ready, 6 CPs

§ Special CAL syntax form

062-063

Special cases

For 062:

(Si) = (Sk)	normalized if (Sk) exponent is valid, j = 0 and k ≠ 0
(Si) = (Sj)	normalized if (Sj) exponent is valid, j ≠ 0 and k = 0

For 063:

(Si) = -(Sk)	normalized if (Sk) exponent is valid, j = 0 and k ≠ 0. Sign of (Si) is opposite that of (Sk) if (Sk) ≠ 0.
(Si) = (Sj)	normalized if (Sj) exponent is valid, j ≠ 0 and k = 0

CAL Syntax		Description	Octal Code
Si	Sj*FSk	Floating-point product of (Sj) and (Sk) to Si	064ijk
Si	Sj*HSk	Half-precision rounded floating-point product of (Sj) and (Sk) to Si	065ijk
Si	Sj*RSk	Rounded floating-point product of (Sj) and (Sk) to Si	066ijk
Si	Sj*ISK	Reciprocal iteration; 2-(Sj)*(Sk) to Si	067ijk

These instructions are executed by the floating-point multiply unit. Operands are assumed to be in floating-point format. The result is not guaranteed to be normalized if the operands are not.

The 064 instruction forms the product of the floating-point quantities in Sj and Sk and enters the result into Si.

The 065 instruction forms the half-precision rounded product of the floating-point quantities in Sj and Sk and enters the result into Si. The low-order 19 bits of the result are cleared.

The 066 instruction forms the rounded product of the floating-point quantities in Sj and Sk and enters the result into Si.

The 067 instruction forms two minus the product of the floating-point quantities in Sj and Sk and enters the result into Si. This instruction is used in the divide sequence as described in section 4 under Floating-Point Arithmetic.

In the evaluation C = 2-B * A, B must be a reciprocal of A of less than 47 significant bits. Otherwise C will be in error. The reciprocal produced by the reciprocal approximation instruction meets this criterion.

Hold issue conditions

034 - 037 in process

Exchange in process

S register access conflict

Si, Sj, or Sk reserved

160 - 167 in process; unit busy (VL) + 4 CPs

Execution time

Instruction issue, 1 CP

Si ready, 7 CPs

Special cases

(Sj) = 0 if j = 0

(Sk) = 2^{63} if k = 0

If both exponent fields are 0, an integer multiply is performed. Correct integer multiply results and produced if the following conditions are met:

1. Both operand sign bits are 0.
2. The sum of the zero bits to the right of the least significant one bit in the two operands is greater than or equal to 48.

In this case the integer result obtained is the high-order 48 bits of the 96-bit product of the two operands.

CAL Syntax	Description	Octal Code
Si /HSj	Floating-point reciprocal approximation of (Sj) to Si	070ijx

This instruction is executed in the reciprocal approximation unit.

The instruction forms an approximation to the reciprocal of the normalized floating-point quantity in Sj and enters the result into Si. This instruction occurs in the divide sequence to compute the quotient of two floating-point quantities as described in part 2, section 4 under Floating-Point Arithmetic.

The reciprocal approximation instruction produces a result that is accurate to 30 bits. A second approximation may be generated to extend the accuracy to 48 bits using the reciprocal iteration instruction.

Hold issue conditions

034 - 037 in process

Exchange in process

Si or Sj reserved

174 in process; unit busy (VL) + 4 CPs

Execution time

Si ready, 14 CPs

Instruction issue, 1 CP

Special cases

(Si) is meaningless if (Sj) is not normalized; the unit assumes that bit 2^{47} of (Sj) = 1; no test is made of this bit.

(Sj) = 0 produces a range error; the result is meaningless.

(Sj) = 0 if j = 0

CAL Syntax		Description	Octal Code
Si	Ak	Transmit (Ak) to Si with no sign extension	071i0k
Si	+Ak	Transmit (Ak) to Si with sign extension	071i1k
Si	+FAk	Transmit (Ak) to Si as unnormalized floating-point number	071i2k
Si	0.6	Transmit constant 0.75 x 2^{48} to Si	071i3k
Si	0.4	Transmit constant 0.5 to Si	071i4k
Si	1.	Transmit constant 1.0 to Si	071i5k
Si	2.	Transmit constant 2.0 to Si	071i6k
Si	4.	Transmit constant 4.0 to Si	071i7k

These instructions perform functions that depend on the value of the j designator. The functions are concerned with transmitting information from an A register to an S register and with generating frequently used floating-point constants.

When the j designator is 0, the 24-bit value in Ak is transmitted to Si. The value is treated as an unsigned integer. The high-order bits of Si are cleared.

When the j designator is 1, the 24-bit value in Ak is transmitted to Si. The value is treated as a signed integer. The sign bit of Ak is extended to the high-order bit of Si.

When the j designator is 2, the 24-bit value in Ak is transmitted to Si as an unnormalized floating-point quantity. The result can then be added to 0 to normalize. For this instruction, the exponent in bits 2^{62} through 2^{48} is set to 40060_8. The sign of the coefficient is set according to the sign of Ak. If the sign bit of Ak is set, the twos complement of Ak is entered into Si as the magnitude of the coefficient and bit 2^{63} of Si is set for the sign of the coefficient.

A sequence of instructions which would be used to convert to floating-point format, an integer whose absolute value is less than 24 bits is:

```
(CAL)  A1   S1
       S1   +FA1
       S    +FS1     9 CPs required.
```

When the j designator is 3, the floating-point constant of 0.75×2^{48} is entered into Si (4006060000000000000_8). This constant is used to create floating-point numbers from integer numbers (positive and negative) whose absolute value is less than 47 bits. A sequence of instructions which would be used for conversion of an integer in S1 is:

```
(CAL)  S2  0.6
       S1  S2-S1
       S1  S2-FS1     11 CPs required.
```

When the j designator is 4, the floating-point constant 0.5 (= 0 40000 4000 0000 0000 0000_8) is entered into Si.

When the j designator is 5, the floating-point constant 1.0 (= 0 40001 4000 0000 0000 0000_8) is entered into Si.

When the j designator is 6, the floating-point constant 2.0 (= 0 40002 4000 0000 0000 0000_8) is entered into Si.

When the j designator is 7, the floating-point constant 4.0 (= 0 40003 4000 0000 0000 0000_8) is entered into Si.

Hold issue conditions

034 - 037 in process

Exchange in process

Si register access conflict

Si reserved

Ak reserved (all instructions)

Execution time

Instruction issue, 1 CP

Si ready, 2 CPs

071

Special cases

(Ak) = 1 if k = 0

(Si) = (Ak) if j = 0

(Si) = (Ak) sign extended if j = 1

(Si) = (Ak) unnormalized if j = 2

(Si) = 0.6×2^{60} (octal) if j = 3

(Si) = 0.4×2^{0} (octal) if j = 4

(Si) = 0.4×2^{1} (octal) if j = 5

(Si) = 0.4×2^{2} (octal) if j = 6

(Si) = 0.4×2^{3} (octal) if j = 7

CAL Syntax	Description	Octal Code
Si RT	Transmit (RTC) to Si	072ixx
Si VM	Transmit (VM) to Si	073ixx
Si Tjk	Transmit (Tjk) to Si	074ijk
Tjk Si	Transmit (Si) to Tjk	075ijk

These instructions transmit register values to Si except for instruction 075 which transmits (Si) to Tjk.

The 072 instruction enters the 64-bit value of the real-time clock into Si. The clock is incremented by 1 each clock period. The real-time clock can be set only by the monitor through use of the 0014 instruction.

The 073 instruction enters the 64-bit value of the vector mask (VM) register into Si. The VM register is usually read after having been set by the 175 instruction.

The 074 instruction enters the contents of Tjk into Si.

The 075 instruction enters the contents of Si into Tjk.

Hold issue conditions

034 - 037 in process

Exchange in process

Si register access conflict (072, 073, and 074 only)

Si reserved

For 073 only:

175 in process, VM busy (VL) + 6 CPs

003 in process, VM not available until 6 CPs after 003 issue

Execution time

Instruction issue, 1 CP

For 072 through 074, Si ready, 1 CP

For 075, Tjk ready, 1 CP

Special cases

None

CAL Syntax	Description	Octal Code
Si Vj,Ak	Transmit (Vj element (Ak)) to Si	076ijk
Vi,Ak Sj	Transmit (Sj) to Vi element (Ak)	077ijk
Vi,Ak 0§	Clear Vi element (Ak)	077i0k

These instructions transmit a 64-bit quantity between a V register element and an S register.

The 076 instruction transmits the contents of an element of register Vj to Si.

The 077 instruction transmits the contents of register Sj to an element of register Vi.

The low-order 6 bits of (Ak) determine the vector element for either instruction.

Hold issue conditions

034 - 037 in process

Exchange in process

Ak reserved

Si register access conflict (076 only)

For 076, Si and Vj reserved

For 077, Vi and Sj reserved

Execution time

Instruction issue, 1 CP

For 076, Si ready, 5 CPs

For 077, Vi ready, 1 CP

Special cases

(Sj) = 0 if j = 0

(Ak) = 1 if k = 0

§ Special CAL syntax form

CAL Syntax	Description	Octal Code
Ai exp,Ah	Read from ((Ah) + jkm) to Ai	10hijkm
Ai exp,0§	Read from (jkm) to Ai	100ijkm
Ai exp,§	Read from (jkm) to Ai	100ijkm
Ai ,Ah§	Read from (Ah) to Ai	10hi000
exp,Ah Ai	Store (Ai) to (Ah) + jkm	11hijkm
exp,0 Ai§	Store (Ai) to jkm	110ijkm
exp, Ai§	Store (Ai) to exp	110ijkm
,Ah Ai§	Store (Ai) to (Ah)	11hi000
Si exp,Ah	Read from ((Ah) + jkm) to Si	12hijkm
Si exp,0§	Read from (exp) to Si	120ijkm
Si exp,§	Read from (exp) to Si	120ijkm
Si ,Ah§	Read from (Ah) to Si	12hi000
exp,Ah Si	Store (Si) to (Ah) + jkm	13hijkm
exp,0 Si§	Store (Si) to exp	130ijkm
exp, Si§	Store (Si) to exp	130ijkm
,Ah Si§	Store (Si) to (Ah)	13hi000

These 2-parcel instructions transmit data between memory and an A register or an S register. The content of Ah (treated as a 22-bit signed integer) is added to the signed 22-bit integer in the jkm field to determine the memory address. If h is 0, (Ah) is 0 and only the jkm field is used for the address. The address arithmetic is performed by an address adder similar to but separate from the address add unit.

The 10h and 11h instructions transmit 24-bit quantities to or from A registers. When transmitting data from memory to an A register, the high-order 40 bits of the memory word are ignored. On a store from Ai into memory, the high-order 40 bits of the memory word are zeroed.

§ Special CAL syntax form

10h-13h

The 12h and 13h instructions transmit 64-bit quantities to or from register Si.

Hold issue conditions

034 - 037 in process

Exchange in process

Rank A bank conflict and unit busy 3 CPs

Rank B bank conflict and unit busy 2 CPs

Rank C bank conflict and unit busy 1 CP

Storage hold continuation

Ah reserved

For 10h only, Ai register access conflict

For 10h and 11h only, Ai reserved

For 12h and 13h only, Si reserved

For 12h only, Si register access conflict

Fetch request in previous clock period

176 in process unit busy (VL) + 4 CPs

177 in process unit busy (VL) + 5 CPs

Execution time

Instruction issue:

Both parcels in same buffer, 2 CPs

Parcels in different buffers, 4 CPs

Second parcel not in a buffer, 13 CPs

10h only, Ai ready, 11 CPs

12h only, Si ready, 11 CPs

Memory ready for next scalar read or store, 4 CPs

Special cases

For 10h, 12h only:

Rank A conflict, 3 CPs delay before Ai or Si ready

Rank B conflict, 2 CPs delay before Ai or Si ready

Rank C conflict, 1 CP delay before Ai or Si ready

For 12h only:

Hold storage, 1 CP delay if 070 register access conflict occurs (when the result entering coincides with a reciprocal approximation result entering Si).

CAL Syntax	Description	Octal Code
Vi Sj&Vk	Logical products of (Sj) and (Vk elements) to Vi elements	140ijk
Vi Vj&Vk	Logical products of (Vj elements) and (Vk elements) to Vi elements	141ijk
Vi Sj!Vk	Logical sums of (Sj) and (Vk elements) to Vi elements	142ijk
Vi Vk§	Transmit (Vk) elements to Vi elements	142i0k
Vi Vj!Vk	Logical sums of (Vj elements) and (Vk elements) to Vi elements	143ijk
Vi Sj/Vk	Logical differences of (Sj) and (Vk elements) to Vi elements	144ijk
Vi Vj/Vk	Logical differences of (Vj elements) and (Vk elements to Vi elements	145ijk
Vi 0§	Clear Vi elements	145iii
Vi Sj!Vk&VM	If VM bit = 1, transmit (Sj) to Vi elements If VM bit = 0, transmit (Vk elements) to Vi elements	146ijk
Vi #VM&Vk§	Vector merge of (Vk) elements and 0 to Vi elements	146i0k
Vi Vj!Vk&VM	If VM bit = 1, transmit (Vj elements) to Vi elements If VM bit = 0, transmit (Vk elements) to Vi elements	147ijk

These instructions are executed by the vector logical unit. The number of operations performed is determined by the contents of the VL register. All operations start with element 0 of the Vi, Vj, or Vk register and increment the element number by 1 for each operation performed. All results are delivered to Vi.

§ Special CAL syntax form

For instructions 140, 142, 144, and 146, a copy of the content of Sj is delivered to the functional unit where it is held as one of the operands until the completion of the operation. For instructions 141, 143, 145, and 147, all operands are obtained from V registers.

Instructions 140 and 141 form the logical products (AND) of pairs of operands and enter the result into Vi. Bits of an element of Vi are set to 1 when the corresponding bits of (Sj) or (Vj element) and (Vk element) are 1 as in the following:

```
(Sj) or (Vj element) = 1 1 0 0
(Vk element)         = 1 0 1 0
(Vi element)         = 1 0 0 0
```

The 142 and 143 instructions form the logical sums (inclusive OR) of pairs of operands and deliver the results to Vi. Bits of an element of Vi are set to 1 when one of the corresponding bits of (Sj) or (Vj element) and (Vk element) is 1 as in the following:

```
(Sj) or (Vj element) = 1 1 0 0
(Vk element)         = 1 0 1 0
(Vi element)         = 1 1 1 0
```

The 144 and 145 instructions form the logical differences (exclusive OR) of pairs of operands and deliver the results of Vi. Bits of an element are set to 1 when the corresponding bit of (Sj) or (Vj element) are different from (Vk element) as in the following:

```
(Sj) or (Vj element) = 1 1 0 0
(Vk element)         = 1 0 1 0
(Vi element)         = 0 1 1 0
```

The 146 and 147 instructions transmit operands to Vi depending on the contents of the vector mask register (VM). Bit 2^{63} of the mask corresponds to element 0 of a V register. Bit 2^{0} corresponds to element 63. Operand pairs used for the selection depend on the instruction. For the 146 instructions, the first operand is always (Sj), the second operand is (Vk element). For the 147 instruction, the first operand is (Vj element) and the second operand is (Vk element). If bit n of the vector mask is 1, the first operand is transmitted; if bit n of the mask is 0, the second operand, (Vk element), is selected.

Examples

1. Suppose that a 146 instruction is to be executed and the following register conditions exist:

```
(VL)   = 4
(VM)   = 0 60000 0000 0000 0000 0000
(S2)   = -1
(V600) = 1
(V601) = 2
(V602) = 3
(V603) = 4
```

   Instruction 146726 is executed. Following execution, the first four elements of V7 contain the following values:

```
(V700) = 1
(V701) = -1
(V702) = -1
(V703) = 4
```

   The remaining elements of V7 are unaltered.

2. Suppose that a 147 instruction is to be executed and the following register conditions exist:

```
(VL)   = 4
(VM)   = 0 600000 0000 0000 0000 0000
(V200) = 1      (V300) = -1
(V201) = 2      (V301) = -2
(V202) = 3      (V302) = -3
(V203) = 4      (V303) = -4
```

   Instruction 147123 is executed. Following execution, the first four elements of V1 contain the following values:

```
(V100) = -1
(V101) = 2
(V102) = 3
(V103) = -4
```

   The remaining elements of V1 are unaltered.

140-147

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vk reserved

14x in process, unit busy (VL) + 4 CPs

175 in process, unit busy (VL) + 4 CPs

003 in process, unit busy 3 CPs

For 140, 142, 144, 146 only, Sj reserved

For 141, 143, 145, 147 only, Vj reserved

Execution time

Instruction issue, 1 CP

Vi ready, 9 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 4 CPs if (VL) greater than 5

Vj or Vk ready, 5 CPs if (VL) is less than or equal to 5

Vj or Vk ready, (VL) CPs if (VL) greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 4 CPs

Special cases

(Sj) = 0 if j = 0

For 145 only, if i = j = k, (Vi) = 0.

CAL Syntax	Description	Octal Code
Vi Vj <Ak	Shift (Vj) elements left by (Ak) places to Vi elements	150ijk
Vi Vj <1§	Shift (Vj) elements left one place to Vi elements	150ij0
Vi Vj >Ak	Shift of (Vj) elements right by (Ak) places to Vi elements	151ijk
Vi Vj >1§	Shift (Vj) elements right one place to Vi elements	151ij0

These instructions are executed in the vector shift unit. The number of operations performed is determined by the contents of the VL register. Operations start with element 0 of the Vi and Vj registers and end with elements specified by (VL) - 1.

All shifts are end-off with zero fill. The shift count is obtained from (Ak) and elements of Vi are cleared if the shift count exceeds 63. All shift counts (Ak) are considered positive. All 24 bits of Ak are used for the shift count.

Unlike the 052-055 shift instructions, these instructions get the shift count from Ak, rather than the jk fields.

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vj reserved

Ak reserved

150 - 153 in process, unit busy (VL) + 4 CPs

§ Special CAL syntax form

150-151

Execution time

Instruction issue, 1 CP

Vi ready, 11 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 6 CPs if (VL) greater than 5

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 6 CPs

Special case

(Ak) = 1 if k = 0

CAL Syntax	Description	Octal Code
Vi Vj,Vj<Ak	Double shifts of (Vj elements) left (Ak) places to Vi elements	152ijk
Vi Vj,Vj<1§	Double shifts of (Vj elements) left one place to Vi elements	152ij0
Vi Vj,Vj>Ak	Double shifts of (Vj elements) right (Ak) places to Vi elements	153ijk
Vi Vj,Vj>1§	Double shifts of (Vj elements) right one place to Vi elements	153ij0

These instructions are executed in the vector shift unit. They shift 128-bit values formed by logically joining the contents of two elements of the Vj register. The direction of the shift determines whether the high-order bits or the low-order bits of the result are sent to Vi. Shift counts are obtained from register Ak.

All shifts are end-off with zero fill.

The number of operations is determined by the contents of the VL register.

The 152 instruction performs left shifts. The operation starts with element 0 of Vj being joined with element 1 and the resulting 128-bit quantity is then shifted left by the amount specified by (Ak). The 64 high-order bits remaining are transmitted to element 0 of Vi. Figure 6-7 illustrates this operation.

If (VL) is 1, element 0 is joined with 64 bits of 0 and only the one operation is performed. If (VL) is greater than 2, the operation continues by joining element 1 with element 2 and transmitting the 64-bit result to element 1 of Vi. This is illustrated in figure 6-8.

If (VL) is 2, however, element 1 is joined with 64 bits of 0 and only two operations are performed. In general, the last element of Vj as determined by (VL) is joined with 64 bits of zeros. Figure 6-9 illustrates this operation.

§ Special CAL syntax form

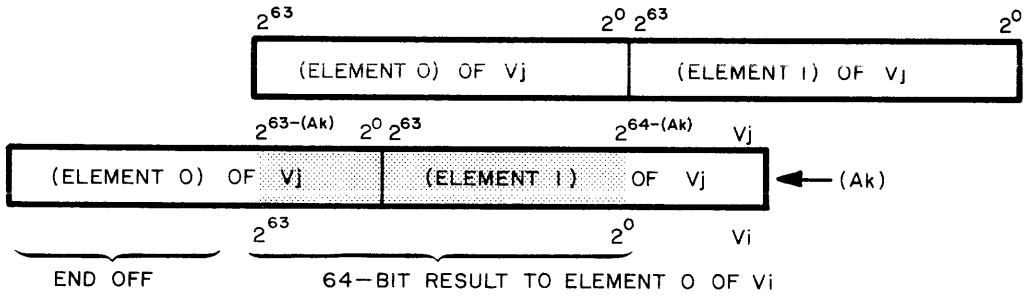


Figure 6-7. Vector left double shift, first element, VL greater than 1

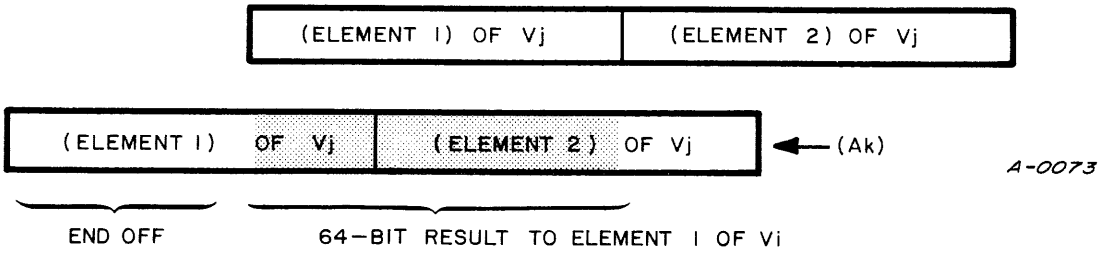


Figure 6-8. Vector left double shift, second element, VL greater than 2

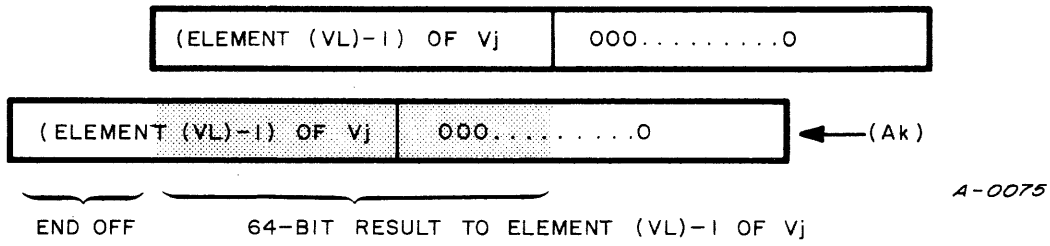


Figure 6-9. Vector left double shift, last element

If (Ak) is greater than 128, the result is all zeros. If (Ak) is greater than 64, the result register contains at least (Ak) - 64 zeros.

Example

Suppose that a 152 instruction is to be executed and the following register conditions exist:

```
(VL)   = 4
(Al)   = 3
(V400) = 0 00000 0000 0000 0000 0007
(V401) = 0 60000 0000 0000 0000 0005
(V402) = 1 00000 0000 0000 0000 0006
(V403) = 1 60000 0000 0000 0000 0007
```

Instruction 152541 is executed and following execution, the first four elements of V_5 contain the following values:

```
(V500) = 0 00000 0000 0000 0000 0073
(V501) = 0 00000 0000 0000 0000 0054
(V502) = 0 00000 0000 0000 0000 0067
(V503) = 0 00000 0000 0000 0000 0070
```

The 153 instruction performs right shifts. Element 0 of Vj is joined with 64 high-order bits of 0 and the 128 bit quantity is shifted right by the amount specified by (Ak). The 64 low-order bits of the result are transmitted to element 0 of Vi. Figure 6-10 illustrates this operation.

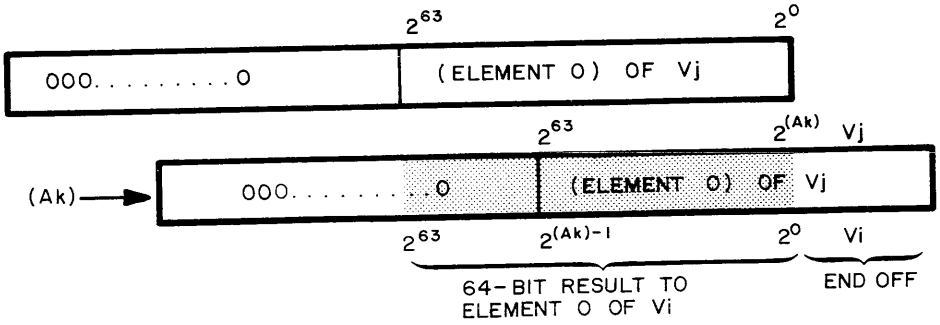


Figure 6-10. Vector right double shift, first element

If (VL) = 1, only the one operation is performed. In the general case, however, instruction execution continues by joining element 0 with element 1, shifting the 128-bit quantity by the amount specified by (Ak), and transmitting the result to element 1 of Vi. This operation is shown in figure 6-11.

152-153

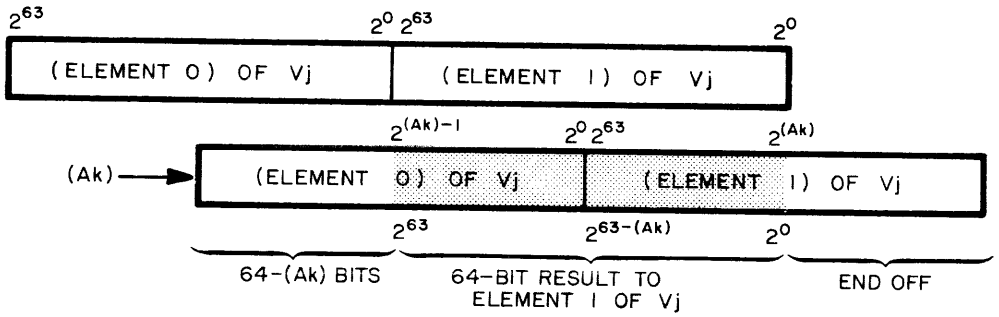


Figure 6-11. Vector right double shift, second element, VL greater than 1

The last operation performed by the instruction joins the last element of Vj as determined by (VL) with the preceding element. Figure 6-12 illustrates this operation.

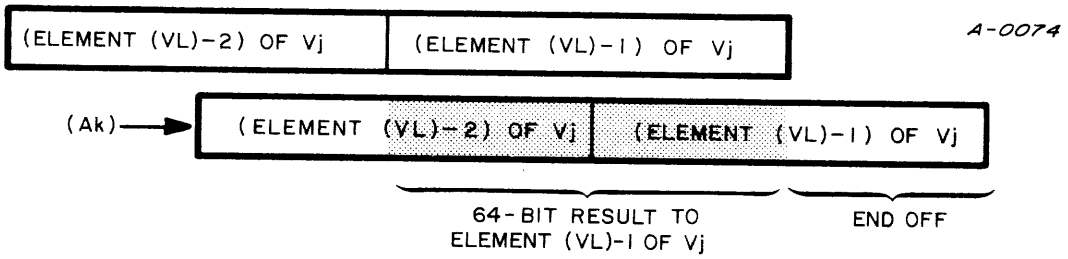


Figure 6-12. Vector right double shift, last operation

Example

Suppose that a 153 instruction is to be executed and the following register conditions exist:

```
(VL)   = 4
(A6)   = 3
(V200) = 0 00000 0000 0000 0000 0017
(V201) = 0 60000 0000 0000 0000 0006
(V202) = 1 00000 0000 0000 0000 0006
(V203) = 1 60000 0000 0000 0000 0007
```

Instruction 153026 is executed and following execution, register V0 contains the following values:

```
(V000) = 0 00000 0000 0000 0000 0001
(V001) = 1 66000 0000 0000 0000 0000
(V002) = 1 50000 0000 0000 0000 0000
(V003) = 1 56000 0000 0000 0000 0000
```

The remaining elements of V0 are unaltered.

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vj reserved

Ak reserved

150 - 153 in process, unit busy (VL) + 4 CPs

Execution time

Instruction issue, 1 CP

Vi ready, 11 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 6 CPs if (VL) is greater than 5

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 6 CPs

Special case

(Ak) = 1 if k = 0

CAL Syntax		Description	Octal Code
Vi	Sj+Vk	Integer sums of (Sj) and (Vk elements) to Vi elements	154ijk
Vi	Vj+Vk	Integer sums of (Vj elements) and (Vk elements) to Vi elements	155ijk
Vi	Sj-Vk	Integer differences of (Sj) and (Vk elements) to Vi elements	156ijk
Vi	-Vk§	Transmit negative of (Vk elements) to Vi elements	156i0k
Vi	Vj-Vk	Integer differences of (Vj elements) and (Vk elements) to Vi elements	157ijk

These instructions are executed by the vector add unit.

Instructions 154 and 155 perform integer addition. Instructions 156 and 157 perform integer subtraction. The number of additions or subractions performed is determined by the contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each operation performed. All results are delivered to elements of Vi. No overflow is detected.

Instructions 154 and 156 deliver a copy of (Sj) to the functional unit where the copy is retained as one of the operands until the vector operation completes. The other operand is an element of Vk. For instructions 155 and 157, both operands are obtained from V registers.

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vk reserved

154 - 157 in process, unit busy (VL) + 4 CPs

For 154 and 156 only, Sj reserved

For 156 and 157 only, Vj reserved

§ Special CAL syntax form

Execution time

Instruction issue, 1 CP

Vi ready, 10 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 5 CPs if (VL) is greater than 5

Vj or Vk ready, 5 CPs if (VL) is less than or equal to 5

Vj or Vk ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 5 CPs

Special cases

For 154, if j = 0, then (Sj) = 0 and (Vi element) = (Vk element)

For 156, if j = 0, then (Sj) = 0 and (Vi element) = -(Vk element)

CAL Syntax		Description	Octal Code
Vi	Sj*FVk	Floating-point products of (Sj) and (Vk elements) to Vi elements	160ijk
Vi	Vj*FVk	Floating-point products of (Vj elements) and (Vk elements) to Vi elements	161ijk
Vi	Sj*HVk	Half-precision rounded floating-point products of (Sj) and (Vk elements) to Vi elements	162ijk
Vi	Vj*HVk	Half-precision rounded floating-point products of (Vj elements) and (Vk elements) to Vi elements	163ijk
Vi	Sj*RVk	Rounded floating-point products of (Sj) and (Vk elements) to Vi elements	164ijk
Vi	Vj*RVk	Rounded floating-point products of (Vj elements) and (Vk elements) to Vi elements	165ijk
Vi	Sj*IVk	Reciprocal iterations; 2 - (Sj) * (Vk elements) to Vi elements	166ijk
Vi	Vj*IVk	Reciprocal iterations; 2 - (Vj elements) * (Vk elements) to Vi elements	167ijk

These instructions are executed in the floating-point multiply unit. The number of operations performed by an instruction is determined by the contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each successive operation.

Operands are assumed to be in floating-point format. Even-numbered instructions in the group deliver a copy of (Sj) to the functional unit where the copy is retained as one of the operands until the completion of the operation. The other operand is an element of Vk. For odd-numbered instructions in the group, both operands are obtained from V registers.

All results are delivered to elements of Vi. If either operand is not normalized, there is no guarantee that the products will be normalized.

Out-of-range conditions are described in section 4.

The 160 instruction forms the products of the floating-point quantity in Sj and the floating-point quantities in elements of Vk and enters the result into Vi.

The 161 instruction forms the products of the floating-point quantities in elements of Vj and Vk and enters the results into Vi.

The 162 instruction forms the half-precision rounded products of the floating-point quantity in Sj and the floating-point quantities in elements of Vk and enters the results into Vi. The low-order 19 bits of the result elements are zeroed.

The 163 instruction forms the half-precision rounded products of the floating-point quantities in elements of Vj and Vk and enters the results into Vi. The low-order 19 bits of the result elements are zeroed.

The 164 instruction forms the rounded products of the floating-point quantity in Sj and the floating-point quantities in elements of Vk and enters the results into Vi.

The 165 instruction forms the rounded products of the floating-point quantities in elements of Vj and Vk and enters the results into Vi.

The 166 instruction forms for each element, two minus the product of the floating-point quantity in Sj and the floating-point quantity in elements of Vk. It then enters the results into Vi. See the description of the 067 instruction for more details.

The 167 instruction forms for each element pair, two minus the product of the floating-point quantities in elements of Vj and Vk and enters the results into Vi. See the description of the 067 instruction for more details.

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vk reserved

16x in process, unit busy (VL) + 4 CPs

For 160, 162, 164, and 166, Sj reserved

For 161, 163, 165, and 167, Vj reserved

160-167

Execution time

Instruction issue, 1 CP

Vi ready, 14 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 9 CPS if (VL) is greater than 5

Vj or Vk ready, 5 CPS if (VL) is less than or equal to 5

Vj or Vk ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 9 CPs

Special case

(Sj) = 0 if j = 0

CAL Syntax		Description	Octal Code
Vi	Sj+FVk	Floating-point sums of (Sj) and (Vk elements) to Vi element	170ijk
Vi	+FVk§	Transmit normalized (Vk elements) to Vi elements	170i0k
Vi	Vj+FVk	Floating-point sums of (Vj elements) and (Vk elements) to Vi elements	171ijk
Vi	Sj-FVk	Floating-point differences of (Sj) and (Vk elements) to Vi elements	172ijk
Vi	-FVk§	Transmit normalized negatives of (Vk elements) to Vi elements	172i0k
Vi	Vj-FVk	Floating-point differences of (Vj elements) and (Vk elements) to Vi elements	173ijk

These instructions are executed by the floating-point add unit. Instructions 170 and 171 perform floating-point addition; instructions 172 and 173 perform floating-point subtraction. The number of additions or subtractions performed by an instruction is determined by the contents of the VL register. All operations start with element 0 of the V registers and increment the element number by 1 for each operation performed. All results are delivered to Vi. The results are normalized even if the operands are not.

Instructions 170 and 172 deliver a copy of (Sj) to the functional unit where it is retained as one of the operands until the completion of the operation. The other operand is an element of Vk. For instructions 171 and 173, both operands are obtained from V registers.

Out-of-range conditions are described in section 4.

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vk reserved

170 - 173 in process, unit busy (VL) + 4 CPs

For 170, 172, Sj reserved

For 171, 173, Vj reserved

§ Special CAL syntax form

170-173

Execution time

Instruction issue, 1 CP

Vi ready, 13 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 8 CPs if (VL) is greater than 5

Vj and Vk ready, 5 CPs if (VL) is less than or equal to 5

Vj and Vk ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 8 CPs

Special case

(Sj) = 0 if j = 0

CAL Syntax	Description	Octal Code
Vi /HVj	Floating-point reciprocal approximation of (Vj elements) to Vi elements	174ij0

This instruction is executed in the reciprocal approximation unit.

The instruction forms an approximate value of the reciprocal of the normalized floating-point quantity in each element of Vj and enters the result into elements of Vi. The number of elements for which approximations are found is determined by the contents of the VL register.

The 174 instruction occurs in the divide sequence to compute the quotients of floating-point quantities as described in section 4 under Floating-Point Arithmetic.

The reciprocal approximation instruction produces results that are accurate to 30 bits. A second approximation may be generated to extend the accuracy to 48 bits using the reciprocal iteration instruction.

Hold issue conditions

034 - 037 in process

Exchange in process

Vi or Vk reserved

174 in process, unit busy for (VL) + 4 CPs

Execution time

Instruction issue, 1 CP

Vi ready, 21 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 16 CPs if (VL) is greater than 5

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 16 CPs

Special case

(Vi element) is meaningless if (Vj element) is not normalized; the unit assumes that bit 2^{47} of (Vj element) is 1; no test of this bit is made.

CAL Syntax	Description	Octal Code
Vi PVj	Population count of (Vj elements) to Vi elements	174ij1
Vi QVj	Population count parity of (Vj elements) to Vi elements	174ij2

The 174ij1 instruction counts the number of bits set to 1 in each element of Vj and enters the results into corresponding elements of Vi. The results are entered into the low-order 7 bits of each Vi element; the remaining high-order bits of each Vi element are zeroed.

The 174ij2 instruction counts the number of bits set to 1 in each element of Vj. The least significant bit of each element result shows whether the result is an odd or an even number. Only the least significant bit of each element is transferred to the least significant bit position of the corresponding element of register Vi. The remainder of the element is set to zeros. The actual population count results are not transferred.

These instructions use the vector population count unit, which shares some logic with the reciprocal approximation functional unit.

Hold issue conditions

034-037 in process

Exchange in process

Vi reserved

Vk reserved

174 in process; unit busy for (VL) + 4 CPs

Execution time

Instruction issue, 1 CP

Vi ready, 13 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 8 CPs if (VL) is greater than 5

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

Unit ready, (VL) + 4 CPs

Chain slot ready, 8 CPs

CAL Syntax	Description	Octal Code
VM Vj,Z	VM = 1 where (Vj element) = 0	175xj0
VM Vj,N	VM = 1 where (Vj element) ≠ 0	175xj1
VM Vj,P	VM = 1 where (Vj element) positive, (bit 2^{63} = 0)	175xj2
VM Vj,M	VM = 1 where (Vj element) negative, (bit 2^{63} = 1)	175xj3

The 175xjk instruction creates a vector mask in VM based on the results of testing the contents of the elements of register Vj. Each bit of VM corresponds to an element of Vj. Bit 2^{63} corresponds to element 0; bit 2^{0} corresponds to element 63.

The type of test made by the instruction depends on the low-order 2 bits of the k designator. The high-order bit of the k designator is not interpreted.

If the k designator is 0, the VM bit is set to 1 when (Vj element) is 0 and is set to 0 when (Vj element) is nonzero.

If the k designator is 1, the VM bit is set to 1 when (Vj element) is nonzero and is set to 0 when (Vj element) is zero.

If the k designator is 2, the VM bit is set to 1 when (Vj element) is positive and is set to 0 when (Vj element) is negative. A zero value is considered positive.

If the k designator is 3, the VM bit is set to 1 when (Vj element) is negative and is set to 0 when (Vj element) is positive. A zero value is considered positive.

The number of elements tested is determined by the contents of the VL register. VM bits corresponding to untested elements of Vj are zeroed.

The 175 vector mask instruction provides a vector counterpart to the scalar conditional branch instructions.

The 175 vector mask instruction uses the vector logical unit.

175

Hold issue conditions

034 - 037 in process

Exchange in process

Vj reserved

14x in process, unit busy (VL) + 4 CPs

003 in process, VM busy 3 CPs

175 in process, unit busy (VL) + 4 CPs

Execution time

Instruction issue, 1 CP

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

VM ready except for 073 instruction, (VL) + 4 CPs

VM ready for 073 instruction, (VL) + 6 CPs

Special cases

k = 0 or 4, VM bit xx = 1 if (Vj element xx) = 0

k = 1 or 5, VM bit xx = 1 if (Vj element xx) $\neq$ 0

k = 2 or 6, VM bit xx = 1 if (Vj element xx) is positive

k = 3 or 7, VM bit xx = 1 if (Vj element xx) is negative

CAL Syntax	Description	Octal Code
Vi ,A0,Ak	Transmit (VL) words from memory to Vi elements starting at memory address (A_0) and incrementing by (Ak) for successive addresses	176ixk
Vi ,A0,1§	Transmit (VL) words from memory to Vi elements starting at memory address (A_0) and incrementing by 1 for successive addresses	176ix0
,A0,Ak Vj	Transmit (VL) words from Vj elements to memory starting at memory address (A_0) and incrementing by (Ak) for successive addresses	177xjk
,A0,1 Vj§	Transmit (VL) words from Vj elements to memory starting at memory address (A_0) and incrementing by 1 for successive addresses	177xj0

These instructions transfer blocks of data between V registers and memory.

The 176 instruction transfers data from memory to elements of register Vi.

The 177 instruction transfers data from elements of register Vj to memory.

Register elements begin with 0 and are incremented by 1 for each transfer. Memory addresses begin with (A_0) and are incremented by the contents of Ak. Ak contains a signed 22-bit integer which is added to the address of the current word to obtain the address of the next word. Ak may specify either a positive or negative increment allowing both forward and backward streams of reference. The two high-order bits of (Ak) are ignored.

The number of words transferred is determined by the contents of the VL register.

§ Special CAL syntax form

176-177

Hold issue conditions

034 - 037 in process

Exchange in process

A_0 reserved

Ak reserved where k = 1 through 7

Block sequence flag set (034 - 037, 176, 177)

Scalar reference (3 CPs maximum)

Rank B data valid

Fetch request in last clock period

For 176, vector register i reserved

For 177, vector register j reserved

I/O memory request

Execution time

For 176 (assuming no bank conflicts):

Instruction issue except for 034-037, 100-137, 176, 177, 1 CP

Instruction issue for above exceptions, (VL) + 4 CPs

Vi ready, 14 CPs if (VL) is less than or equal to 5

Vi ready, (VL) + 9 CPs if (VL) is greater than 5

For 177 (assuming no bank conflicts):

Instruction issue except for 034-037, 100-137, 176, 177, 1 CP

Instruction issue for above exceptions, (VL) + 5 CPs

Vj ready, 5 CPs if (VL) is less than or equal to 5

Vj ready, (VL) CPs if (VL) is greater than 5

Special cases

The increment, (Ak), = 1 if k = 0

Chain slot issue is 9 CPs if full speed for 176, blocked for 177

Inhibit I/O references

Inhibit 034 - 037, 100 - 137, 176, 177

Special cases (continued)

(Ak) determines speed control. Successive addresses are located in successive banks. References to the same bank can be made every 4 CPs or more. Incrementing (Ak) by 16 (16-bank memory) or 8 (8-bank memory) places successive memory references in the same bank, so a word is transferred every 4 CPs. If the address is incremented by 8 (16-bank memory) or 4 (8-bank memory), every other reference is to the same bank and words can transfer every 2 CPs. With any address incrementing that allows 4 CPs before addressing the same bank, one word can transfer each CP.

CPU INTERFACES 7

INTRODUCTION

The CRAY-1 Computer is designed for use with front-end computers in a network of computers. Front-end interfaces connect the CRAY-1 I/O channels to channels of other computers. In S Series systems that do not have an I/O Subsystem, the interfaces are connected to CPU I/O channels. The interfaces adapt the CPU I/O channels to handle differences in voltage levels, grounding requirements, word sizes, data rates, and protocols. Each interface is designed to accommodate a specific type of computer, interfacing it to one CPU low-speed asynchronous I/O channel pair.

Standard interface hardware exists for the Maintenance Control Unit (MCU). Cray Research also offers standard interfaces for a variety of computers produced by other manufacturers and is continually expanding the list of front-end intefaces that it offers.

This section describes the physical construction, cabling limitations, and operation common to the interfaces. Specific information for each interface is contained in separate documentation.

PHYSICAL DESCRIPTION

Each interface is housed in a stand-alone cabinet (figure 7-1) located near the host computer. The cabinet is air-cooled, and operates directly from the 60 Hz AC power mains. The power consumption and therefore the heat generated by the interface cabinet varies with the complexity of the interface. The cabinet contains two or more logic modules and appropriate cabling connector panels. Internal power supplies provide the required logic and communication voltages. Cabinet grounding is flexible and can be configured to specific site requirements.

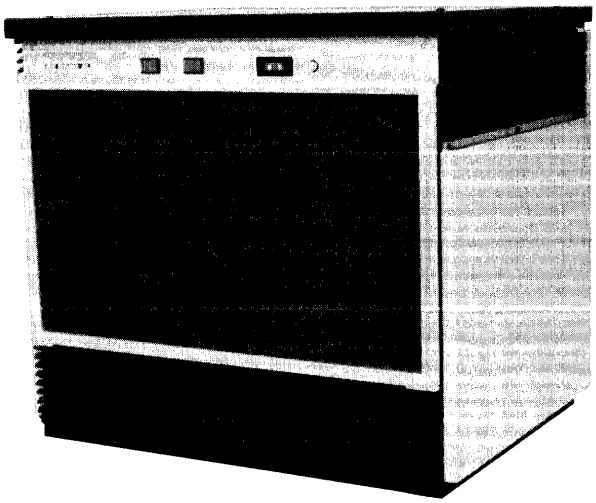


Figure 7-1. Typical interface cabinet

The MCU interface is built on a circuit board that is installed in the MCU computer chassis.

CABLING LIMITATIONS

An interface can be located as far as 320 cable feet from the CRAY-1 CPU. This generally allows enough distance to locate the interface cabinet near the front-end computer, easily meeting the cable length requirements of the front-end machine.

Connectors and cabling are supplied with the interface, so that it is ready to connect to host and CRAY-1 cabling.

The MCU interface has special cabling considerations.

OPERATION

The interface uses hardware logic to perform all command translation and protocol conversion needed to transfer data. Thus, its operation is invisible to the front-end user and the CRAY-1 user. The CRAY-1 is often treated simply as another peripheral device of the front-end system.

PART 3

I/O SUBSYSTEM

GENERAL INFORMATION 1

INTRODUCTION

The I/O Subsystem provides high-capacity data communications between the Central Processing Unit and peripheral devices, storage devices, and front-end computers. The I/O Subsystem is composed of two to four I/O Processors, a group of interfaces, and a Buffer Memory. This part of the manual describes these components.

The I/O Processor is a fast, multipurpose computer capable of extremely high data transfer rates. A 16-bit processor and a fast bipolar memory combine to support high-speed I/O operations. The input and output capabilities make the I/O Processor useful for network control, mass storage access, and computer interfacing.

Figure 1-1 shows the basic organization of an I/O Processor. The I/O Memory stores 65K parcels of 16 bits each. The control section has instruction stack, program exit stack, and control logic. The computation section consists of registers and functional units having interconnected data paths. The I/O section is based in six direct memory access (DMA) ports, and each port can handle data at approximately 850 Mbits per second. These DMA ports are expandable into multiple I/O channels for specialized interfacing needs.

MEMORY SECTION

The memory of an I/O Processor consists of four sections of four banks each of bipolar LSI (large scale integration) memory. All memory sections are independent of each other. The memory cycle time is 4 CPs meaning a section is ready for use again 4 CPs after the preceding memory reference. An exception to this is the I/O write operation which requires 6 CPs. The access time is the time required to bring an operand from memory to the accumulator, which is 7 CPs. Memory capacity is fixed at 65,536 parcels of 16 bits of data plus 2 odd parity bits for each parcel.

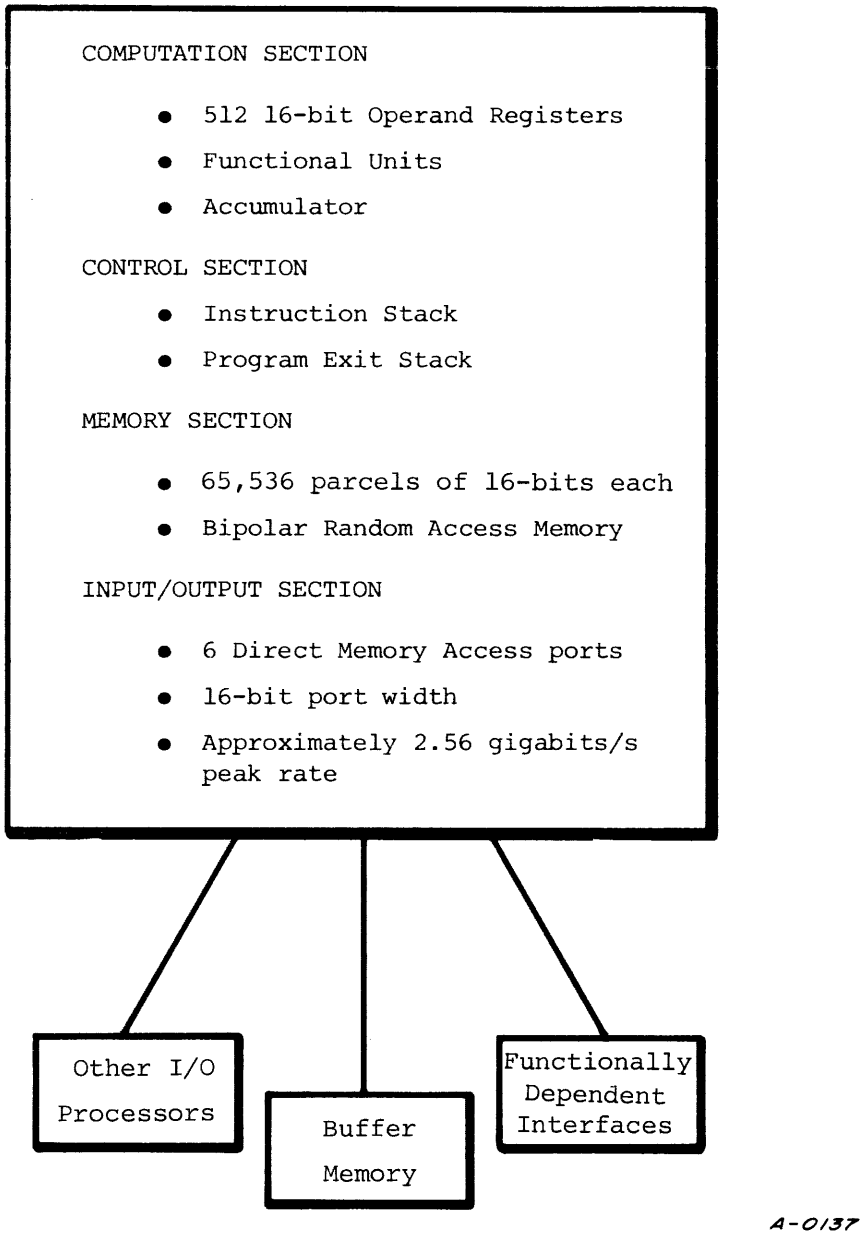


Figure 1-1. Basic organization of an I/O Processor

CONTROL SECTION

The I/O Processor executes 128 instruction codes as either 16-bit (1-parcel) or 32-bit (2-parcel) instructions. A wide assortment of branching and I/O instructions are included in the set. Instructions are stored in memory, and transferred into the instruction stack, under the control of a program address counter. Instructions issue from the instruction stack and are decoded into the control signals that enable the functions of the instruction.

The instruction stack provides fast access to a moving window of program instructions. The instruction stack holds 32 instruction parcels (16 bits each). The program is free to branch quickly about inside the stack, only slowing when more instructions must be read from memory. The instruction stack is actually a 32-parcel circular buffer: the parcels held for the longest time are overwritten by newly transferred instructions. Details are given in the control section discussion in this part of the manual.

The program exit stack is a last-in-first-out set of 16 registers that stores return addresses for program subroutine calls. The 16 registers provide for 14 nested levels of subroutines in the program; a pointer keeps track of the levels involved. An interrupt is generated when the stack is emptied or filled. The stack may be loaded or unloaded by the program. Therefore, the processor is not limited to the 14 nested levels, and it is possible to nest an unlimited number of subroutines by means of the software.

COMPUTATION SECTION

The computation section contains operating registers, functional units and an accumulator which operate together to execute a program of instructions stored in memory. All arithmetic (addition and subtraction) is in twos complement mode in a single adder. Floating-point arithmetic is not incorporated. A shift unit provides left or right shifts of up to 31 bit positions, either circular or end-off shifting. A logical product operation is provided.

Any of the 512 operand registers are used for temporary data storage or for indirect addressing to the memory. All operand registers are 16 bits wide.

The accumulator is a 16-bit signed register for temporary storage of operands or results. All movement of data within this single-address machine uses the accumulator either as the source of data or as the destination for results. The instruction code specifies the register or memory address from which data is brought to the accumulator or the register or memory address to which data is sent from the accumulator.

All transfers between memory and operand registers take place via the accumulator, and accumulator data can also be sent to or received from specialized I/O channels. A seventeenth accumulator bit occupies the 2^{16} bit position as the carry flag. When operations in the adder or shifter yield a carry, the accumulator 2^{16} bit is toggled. The carry flag is available for conditional testing.

INPUT/OUTPUT SECTION

Communication with the I/O Processor is through six direct memory access (DMA) ports. Each port is bidirectional and can transfer four 16-bit data words every 6 CPs. Input and output channels may be active at the same time, as long as they use different ports and are referencing different memory sections. The ports are assigned to channels with the possibility of several channels sharing one port. The slower the required data rate on the channels, the more channels may be multiplexed into one DMA port. The I/O Processor has a total of 40 channels.

Channels use Busy and Done flags for signalling the I/O Processor, and can communicate directly with the I/O Processor accumulator for control information. The use of the flags and the accumulator varies with the specific design of the channel interface logic. All channels communicate status and functions through the accumulator. Some low-speed devices may transfer data directly to and from the accumulator, using one of the channel registers.

I/O SUBSYSTEM CLOCK

The clock controlling the entire I/O Subsystem is a crystal-controlled oscillator running at 80 MHz. This gives a clock period of 12.5 nanoseconds. The speed can be adjusted slightly higher or lower, for maintenance purposes. When doing operations that require exact timing information, such as interval timing with the real-time clock, contact maintenance personnel to verify the clock period.

I/O MEMORY SECTION 2

INTRODUCTION

The I/O Processor memory, called the I/O Memory, consists of 16 banks of bipolar LSI (large scale integration) storage circuits. Each 4-bank section is independent of the other sections. The following paragraphs describe the memory speeds, memory organization, memory access, memory addressing, and memory parity protection.

Table 2-1. I/O Processor memory characteristics

- 65,536 parcels of 16 bits
- 16 banks of 4,096 parcels each
- 4 CP bank cycle time
- Read to accumulator in 7 CPs
- 1 instruction fetched/CP
- 1 data parcel/CP on sequential addressing
- Dual odd parity protection
- 6 full-duplex direct-memory-access ports

MEMORY SPEEDS

The memory cycle time is 4 CPs. The access time, that is, the time required to fetch an operand from memory to the accumulator, is 7 CPs. Instructions are fetched from memory at a rate of one parcel per clock period, in 4-parcel bursts. I/O operations transfer data in bursts of four parcels, one parcel each clock period. Due to the memory organization, explained later in this section, it is possible to reference sequential addresses every clock period, regardless of whether the operation is a read or a write. However, the same section can only be referenced once every 6 CPs. The data transferred in the operation may be one parcel of operand data or four parcels of I/O data.

MEMORY ORGANIZATION

The 16 banks of 4096 parcels each are divided into four sections of four banks each. Each section has separate access paths and sequence controls, as well as write data registers and read data registers. The four banks within a section share a common sequence control. A reference to any one bank of the section also references the other three banks in the section, but only the addressed bank transfers data.

A read reference to a memory section causes all four banks in that section to read out parcels to their read registers. The addressed parcel or parcels are then gated to the destination. In the case of an operand reference, a single parcel goes to the accumulator. In an I/O reference each of the four parcels is gated to the output channel in sequence. The read reference and transmission sequence are fixed with reference to timing of the initiation and moving of data. Once the reference has been started, it continues automatically. After 4 CPs, the section is then free to begin another overlapping reference.

A write sequence from the I/O section can be assembling data for a memory section in the bank write data registers at the same time that an I/O read reference is reading data from that same memory secton. The actual data moved per reference may vary from one operand parcel to a four parcel burst for I/O operation.

MEMORY ACCESS

Each memory section has three 16-bit data paths for reading and two 16-bit data paths for writing. One read path and one write path go to the accumulator. One read path and one write path go to the I/O section as Direct Memory Access (DMA) ports. The last read path goes to the instruction stack and carries instruction parcels. The DMA ports are explained in greater detail in the description of the I/O section later in this manual.

MEMORY ADDRESSING

A parcel in the 16-bank I/O Memory is addressed in 16 bits as shown in figure 2-1.

The low-order 4 bits specify one of the 16 banks. The next higher field specifies an address within the storage chip. The high-order bits select the storage chip.

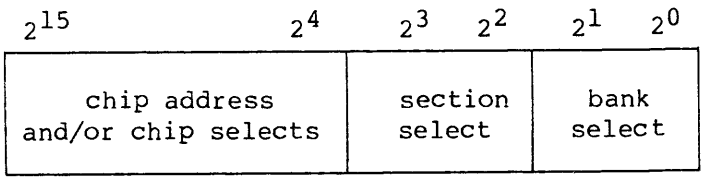


Figure 2-1. I/O Memory address format

Three address paths go to each of the four memory sections: one from the I/O, one from the accumulator, and one for instruction fetch references.

MEMORY PARITY PROTECTION

Data stored in memory is protected by an odd parity scheme. When a 16-bit word is stored in memory, 2 additional parity bits are generated and stored at the same address. One parity bit is assigned to the high-order 8 bits of the data word, and the other is assigned to the low-order 8 bits. Each parity bit is set or cleared to make the sum of 1 bit in the byte and parity bit an odd number. When the data is read from memory, it is checked for odd parity. If parity is even, an error has occurred in storage and an interrupt is generated.

The error handling routine may use the I/O Processor I/O Memory error channel to determine the location of the failing hardware and issue this information for maintenance people. Normally, no further I/O Processor operation would be attempted until the memory fault is corrected. The I/O Memory error channel is described with the Input/Output section in section 5.

IOP CONTROL SECTION 3

INTRODUCTION

The control section of an I/O Processor consists of an instruction control network, an instruction stack, and a program exit stack. Characteristics are summarized in table 3-1.

Table 3-1. Characteristics of IOP control section

- Single addressing mode
- 128 operation codes
- Instruction stack
- Program exit stack, 16 primary levels

Instructions can move data from a source to the accumulator or from the accumulator to a destination. Operand registers serve as temporary storage for operands and results. The functional units receive operand pairs and produce single results. In this single-address machine, one operand address is designated by the instruction and the other operand is contained in the accumulator. A typical data flow is: from memory to accumulator; from accumulator with an operand to functional unit and result back to accumulator; and from accumulator to memory. Figure 3-1 shows the organization of an I/O Processor and following paragraphs describe the elements of the control section.

INSTRUCTION CONTROL NETWORK

Figure 3-1 shows the general organization of the instruction control network for an I/O Processor. Important features are the instruction stack, II register, B register, RP and DP registers, exit stack, and P register. The decoding of operation codes into data path enablings, register reservations, and functional unit requirements are some of the activities of the instruction control network that are beyond the scope of this manual.

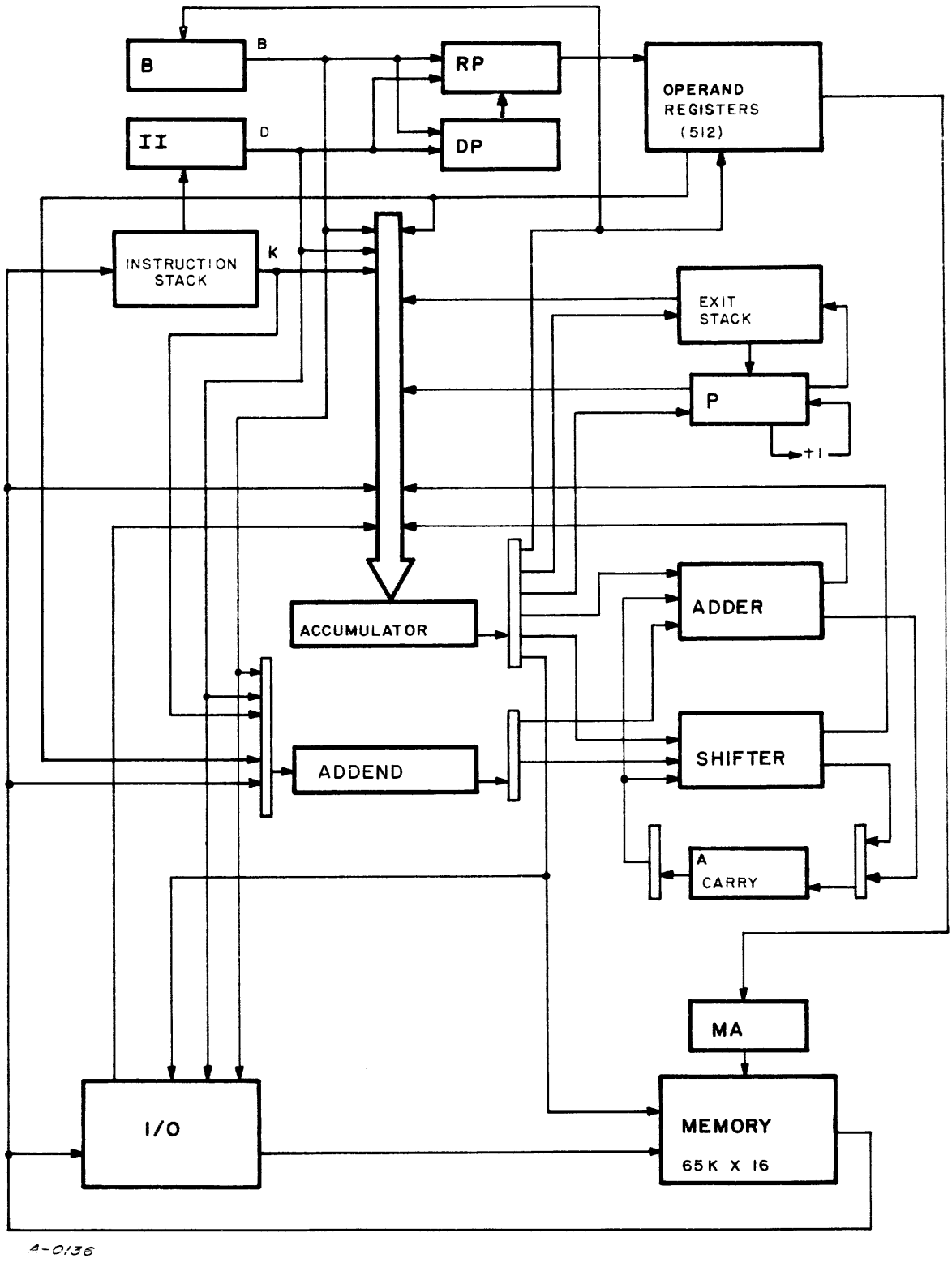


Figure 3-1. I/O Processor block diagram

INSTRUCTION STACK

Instructions occupy one or two parcels and consist of two designators and a constant field as shown in figure 3-2.

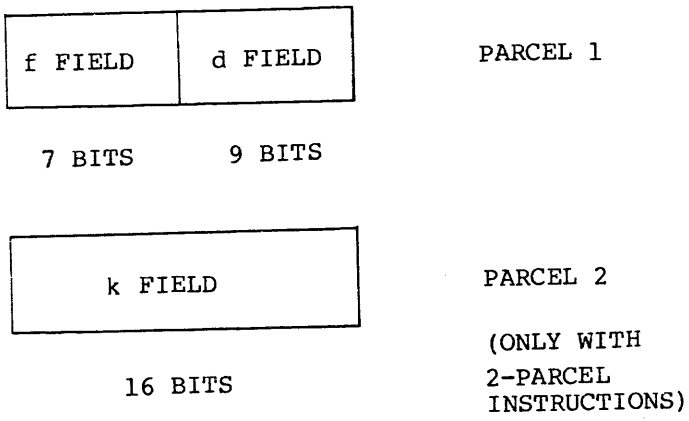


Figure 3-2. Instruction format

The 7-bit f designator is the operation code that specifies which instruction is to be executed, and also designates where and how the execution occurs. The 9-bit d field can contain data, address, or shift count. The 16-bit k field is a constant field that occupies the program parcel immediately following the d and f field parcel. A more detailed explanation of the instruction formats is given at the beginning of the Instructions section.

Program instructions are fetched from memory and stored in a 32-position, 16-bit instruction stack. Instructions are one parcel (16 bits) or two parcels (32 bits). The stack capacity for 32 instruction parcels allows short program loops to execute in the stack without reference to memory.

The instruction stack consists of two banks of registers with 16 parcels of program code in each bank (figure 3-3). The addresses alternate between the two banks so that the loading of data from storage can interleave with the readout of instructions for execution. Instructions are fetched from the memory in bursts of four parcels. Each burst references a storage address that is a multiple of four. The issue control selects which parcel of the four is received first. Loading is then continued sequentially. Memory supplies one parcel each CP if there are no memory conflicts.

The instruction passes directly into execution if the instruction sequence can issue in that clock period. The instruction is stored in the instruction stack regardless of whether it issues immediately or waits for issue. When an issue delay occurs, arriving parcels are stored in the instruction stack.

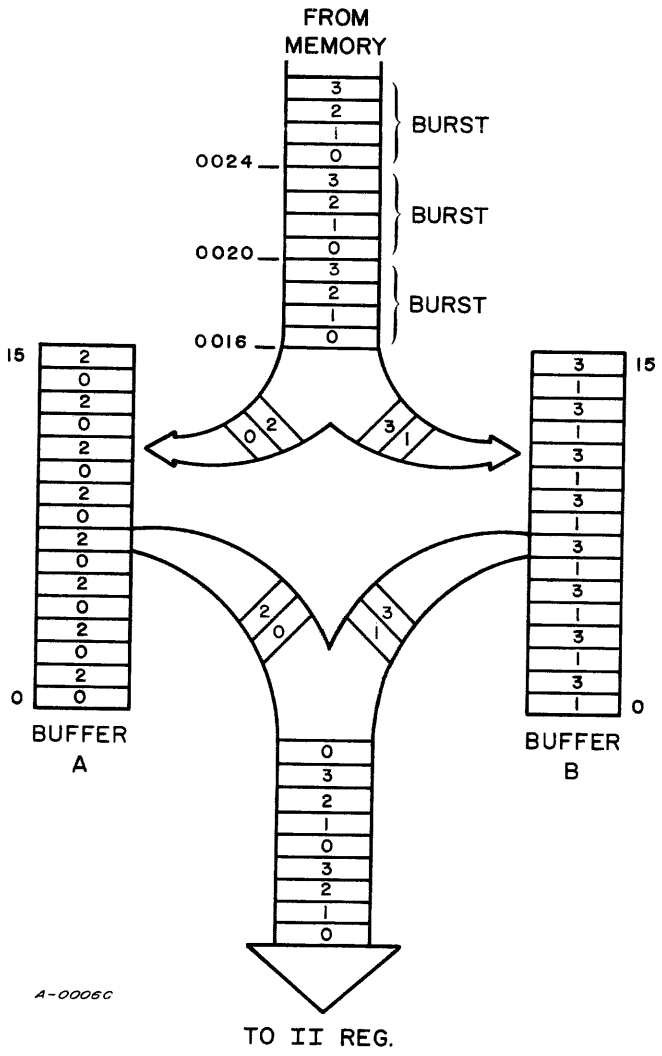


Figure 3-3. Instruction stack operation

The instruction stack is implemented as a circular buffer so that when the stack has been filled and new program code is required, the new code is stored in the beginning address of the stack. Loading continues circularly with the issue control circuits keeping a record of the current section of memory that is represented in the instruction stack. In an attempt to keep the next instructions to be executed in the stack, internal (background) fetches are performed as instructions execute sequentially.

In the following discussion, all numbers are octal.

Normally, two internal fetches are performed to keep the load point of the stack minimally 10_8 locations ahead of the next instruction to issue (stack point). Since fetches are performed on a 4-parcel boundary, the load pointer (L) can actually be 17_8 parcels ahead of the stack pointer (S) if S is at parcel 0 of a 4-parcel group. Once the condition $L-S \geq 10_8$ is satisfied, internal fetches are stopped. When S increments to the point that $L-S < 10_8$, then internal fetches resume. For sequential instructions, when the stack fills up, loading will begin again at location 0. Out-of-stack branch conditions will also cause a fetch beginning at stack location 0. Once this fetch is accomplished, barring other branches, the internal fetch mechanism will take over. All absolute branches (074-077, 120-137) are considered to be out-of-stack. Relative jumps (070-073, 100-117) may be in- or out-of-stack depending on the offset, d, and the location of the stack pointer.

Forward relative branch

Forward relative branches into the area of the stack which has been or will be satisfied by predetermined internal fetches will be in-stack, that is, a forward branch of 10_8 to 17_8 parcels. Normally, 13_8 is the maximum; but under certain conditions, a forward branch of up to 17_8 parcels is considered in-stack. In this case, an extra internal fetch is generated to retrieve the desired instructions. It is, however, impossible to predict when this will happen. Thus 10_8 parcels should be considered the upper limit for in-stack forward branches.

Backward relative branch

When the stack is filling for the first time after an out-of-stack condition, backward relative branches to stack location 0 are valid. After the stack is filling for succeeding times, backward branches into the area being loaded by internal fetches necessitates an out-of-stack condition. However, since the load pointer can only get a maximum of 17_8 parcels ahead of the stack pointer, a reverse jump of 23_8 parcels is guaranteed. If the stack pointer is at parcel 3 of a 4-parcel group, a reverse jump of 27_8 may be achieved. However, the maximum loop size guaranteed to be in-stack is 23_8 parcels.

II REGISTER

The II (Instruction Issue) register is a 16-bit register that receives the instruction parcel from the instruction stack. The instruction parcel may stay in the II register for more than 1 CP and leaves the II register when a new instruction is needed. The f field of the instruction parcel (2^9-2^{15}) is the operation code and is translated by logic associated with the II register to determine the particular sequence of operations required. Bits 2^0 - 2^8 of the instruction parcel are the d field, sent to the RP, DP or addend register, or to the accumulator. If the f field translation shows this parcel is the first of a 2-parcel instruction, the second parcel is sent from the instruction stack to the addend register or accumulator and is not interpreted as an instruction.

B REGISTER

The B register is a 9-bit address register used to designate one of the 512 operand registers. The B register is loaded from the accumulator, taking the low-order (2^0 - 2^8) bits. Accumulator bit 2^0 goes into B register 2^0 location. The contents of B may also address the I/O channel for an I/O instruction or may be used as an operand. In I/O instructions, the B register is an alternate for the instruction d field low-order bits 2^0 - 2^8 and can be altered by the program, in contrast to the d field which is part of the program.

RP REGISTER

The RP (Register Pointer) register is a 9-bit register that directly addresses one of the 512 operand registers for reading or writing. The RP register receives the d field from the II register on issue of each instruction using operand registers.

The operand registers are built to automatically read out data as addressed by the RP register, unless an instruction specifically demands a write into an operand register. The automatic read occurs each clock period, and if the read data is not needed, it is ignored.

DP REGISTER

The DP (Destination Pointer) register contains a 9-bit pointer that selects one of the 512 operand registers to receive the contents of the accumulator. The DP register receives the pointer from the D or B register bits 2^0-2^8. The pointer is sent when the instruction

issues. In a write operation from the accumulator to an operand register, there is usually a delay between the time the instruction issues and the time the register pointer is required. The DP register stores the pointer during the delay period.

A disadvantage exists if the next instruction calls for reading an operand register. Because the transfer of a pointer from the DP register to the RP register uses the same path into RP as is used by the pointer coming from II, instruction issue may be blocked until the path into RP is again free.

P REGISTER

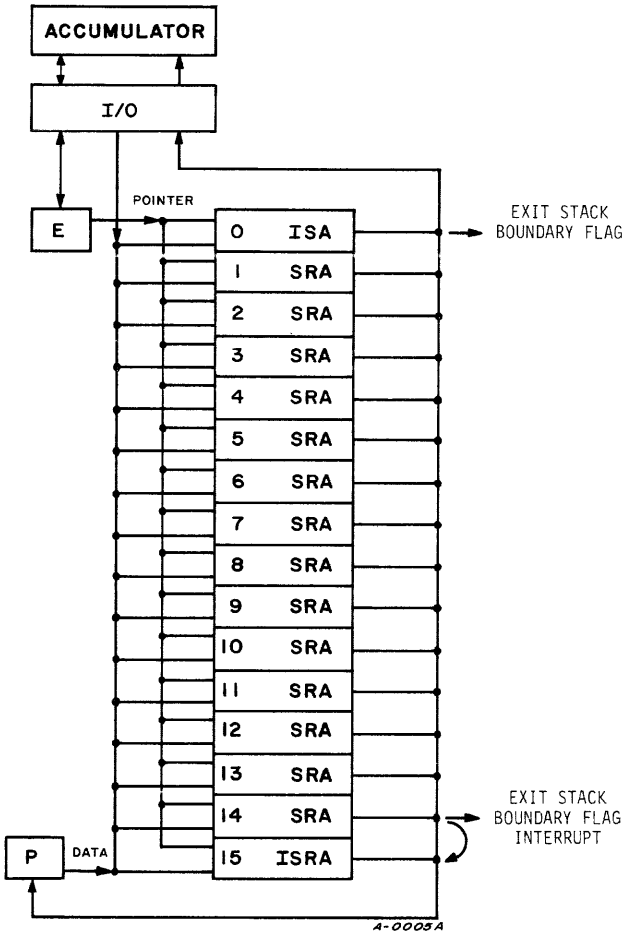
The 16-bit P (Program address) register holds the memory address of the instruction currently awaiting issue. The P register contents are automatically incremented as each instruction is executed in program sequence. A delay between reading from the instruction stack and issue keeps the address in P two program steps behind the instruction stack readout address. This delay is invisible to the programmer.

Branch instructions alter the P register content by either adding a positive or negative displacement value, or by entering an entirely new value.

PROGRAM EXIT STACK

The I/O Processor has a special hardware mechanism for storing the program return addresses when subroutines are called, and for storing the return addresses when the program is suspended to handle interrupts. This mechanism is the program exit stack, which consists of sixteen 16-bit registers. The program exit stack is addressed by a 4-bit pointer, the E register. The program can access and modify the contents of the program exit stack and the E register through I/O channel functions. The program exit stack is shown in figure 3-4.

The zero position in the stack is reserved for the interrupt handler starting address. This address is entered in the stack by the deadstart program and generally remains unaltered for the remainder of system execution. Interrupts cause the hardware to reference the stack zero position without reference to the E register.



ISA = Interrupt Start Address
SRA = Subroutine Return Address
ISRA = Interrupted Subroutine Return Jump Destination Address

Figure 3-4. Program exit stack

Positions 1 through 14 in the program exit stack are used for subroutine return addresses. A subroutine call from a routine advances the E pointer by 1 and then stores in the stack the return address at which the routine will continue when the called subroutine finishes. An exit from the subroutine to the routine reads the currently-pointed location in the stack for the return address. Then the E pointer decrements by one count to point to the higher level stack location. As a program goes into deeper levels of subroutines, the E count increases and more subroutine return addresses are stored. As the program exits out of subroutines, the E count drops back toward 0.

When the E pointer approaches the full limit of the stack, an exit stack interrupt is generated to allow the software to reconfigure the stack. The sequence is as follows:

1. E reaches 13
2. Subroutine call
3. E goes to 14
4. New return address loads to stack position 14
5. Interrupt sets
6. Return jump to new subroutine enters new value to P, but does not jump
7. Interrupt blocks further instruction issue (if system and channel interrupts are enabled)
8. Stack loads the address that was interrupted to position 15
9. Interrupt handler begins executing

When the E pointer reaches 0 and an exit instruction issues, an interrupt sets and the program jumps to the interrupt handler routine.

The sequence is as follows:

1. E reaches 0
2. Exit instruction occurs (normal or error)
3. Interrupt sets
4. Interrupt blocks further instruction issue (if system and channel interrupts are enabled)
5. Interrupt handler begins executing

If return jumps are used in an interrupt handler, it should be verified that there are enough levels left available in the stack. An interrupt with the exit stack pointer at 13_{10} will cause the pointer to go to 14_{10} and leave only one location open. A worse case exists if a return jump which causes a program fetch request (PFR) interrupt is issued with the stack pointer at 13_{10}. The return address will go in 14_{10} and the interrupt address with go into 15_{10}. This will now leave two interrupts present--both the exit stack boundary and PFR, with the PFR being the highest priority and no stack locations available. If the stack pointer is allowed to increment from 15_{10}, it will clear to 0, and incorrect return addresses will be used.

A suggested way of reconfiguring the program exit stack is to keep the stack half full. This gives wide freedom to call deeper levels or exit to higher levels. The deepest level interrupt is characterized by E = 15. When handling it, save the higher half (1-7) of the stack in memory,

move the lower half (8-15) to the higher part of the stack (1-8). This is the old 15 location that holds the interrupted subroutine return address. Exit back to the interrupted subroutine, and operation continues.

When a reconfigured stack exits to the E = 0 level, it exits directly to the interrupt handler routine. This highest level interrupt is characterized by E = 0 with the exit stack interrupt present. The handling routine can then rebuild the original higher half of the stack, reading it from memory. Set E to 7, and exit to that subroutine return address.

Program Exit Stack and I/O Interrupts

An I/O interrupt is treated much like a subroutine call. The interrupted program address is stored in the program exit stack at the next stack position and the entrance address for the interrupt routine is read from the zero position of the stack. On servicing the interrupt, the hardware clears the system interrupt enable flag to prevent other interrupts from interrupting the handling routine.

If return jumps are used in the interrupt handling routine, enough locations must be left in the exit stack to handle the maximum number of levels encountered in the interrupt handling routine. Then, when the interrupt handling routine is finished, it sets the system interrupt enable flag. The exit at the end of the handling routine reads from the exit stack the return address for the interrupted program.

Program Exit Stack Timing Note

After any modification to the stack locations or to the E pointer, at least 4 CPs must elapse before a program exit, return jump, or enabling system interrupts. A simple way to achieve the required delay is to read the modified contents back to the accumulator before exiting the routine. Modifying stack locations or the E pointer should only be done when system interrupts are disabled.

PROGRAM FETCH REQUEST FLAG

The program fetch request (PFR) flag is set during execution of jump instructions 074 - 077 and 120 - 137 when the instruction sequence finds a zero value in operand register d. This condition requires the monitor program to locate and fetch a segment of program code and to find a location in the I/O Memory for execution.

The setting of the program fetch request flag suspends execution of the current program sequence with an interrupt request at the completion of the interrupted instruction. The monitor program then reads the channel number through the channel 1 interface input register and interprets that number as the operand register requiring service.

MA REGISTER

The MA (Memory Address) register holds the address for an I/O Memory reference. The MA register contains 16 bits. It receives address information from an operating register, and holds it for I/O Memory use. This register is used for both read and write memory references.

IOP COMPUTATION SECTION 4

INTRODUCTION

The I/O Processor adds, subtracts, left shifts, and right shifts, using the adder, shifter and logical functional units. Temporary data storage is provided by a block of operand registers. All transfers to operand registers and all results from the functional units pass through the accumulator. These parts of the computation section are described in this section. Characteristics are summarized in table 4-1. Refer to the I/O Processor block diagram in the preceding section, which shows the organization the computation section in the I/O Processor.

Table 4-1. Characteristics of the I/O computation section

- 16-bit architecture
- Twos complement arithmetic
- Integer addition/subtraction unit
- Shift unit
- Logical operations
- 512 operand registers, 16 bits wide

OPERAND REGISTERS

Computation in the I/O Processor is supported by 512 operand registers. Each operand register contains 16 bits of data and has a 1 CP access time. The registers are addressed by the Register Pointer (RP) register. The only data path into the operand registers is from the accumulator. Operand register output data can go to either the accumulator or the addend register as operand data, or it can go to the Memory Address (MA) register as memory address data. The operand registers act as temporary locations for data, as index registers, and as indirect address registers for memory.

FUNCTIONAL UNITS

The computation section consists of an adder functional unit and a shifter functional unit. The computation section performs all the arithmetic required by the instruction set.

ADDER

The arithmetic logic of the I/O Processor is a twos complement adder that can also be used for subtraction. Adder operands come from the accumulator, instruction fields, B register, P register, operand registers, and memory. Except for the accumulator contents, operands come to the adder through the addend register. Adder results go to the accumulator for distribution, as needed and to the P register. The 17-bit operands are received from the accumulator and returned to the accumulator; the seventeenth bit corresponds to the carry bit of the accumulator. Operands from the addend register have 16 bits. The adder is also used by the branch instructions for P address calculations.

This adder can also be used for subtraction. In twos complement arithmetic, subtraction takes place by adding the ones complement of the subtrahend (the number subtracted from the minuend) to the minuend and then adding 1. When subtracting, the contents of the addend register are inverted and passed to the adder. The subtraction control signal is a 1, which is added to the intermediate result to give the final difference. Either an add or subtract occupies 1 CP; another clock period is required to put the results into the accumulator and carry bit register.

SHIFTER

The shifter implements the shifting instructions of the I/O Processor. It shifts up to 31 places left or right, either circularly or end-off with zero fill. The shifter receives the 17-bit accumulator data (including carry bit) to be shifted and the 5-bit addend register shift count. The shifted results, 17 bits, are returned to the accumulator and carry bit register. One CP is required for the shift; this is independent of the shift count and type of shift. Any shift instruction occupies a total of 2 CPs.

The maximum number of places an operand can be shifted is 31. If the shift count is 0 for a left or right shift, no shift occurs. With an end-off shift, if the count is greater than 16, the zero filling clears the result. In all shifts, the carry bit is treated as the highest order bit (2^{16}) of the operand and the result.

ACCUMULATOR

The accumulator is a 16-bit register that temporarily stores operands and results. Data from a wide variety of sources can be routed to the accumulator; many destinations for accumulator contents are available. Sources and destinations are listed in the table 4-2.

The accumulator is used to perform the logical product function. Logic at the input of the accumulator is enabled by the logical product instructions and creates the logical product of two operands. The result goes directly to the accumulator, with no extra time taken for the logical product function

Table 4-2. Accumulator sources and destinations

Sources	Destinations
B register II register d field II register k field Operand registers Adder/Shifter Memory I/O channels	Operand registers Adder/Shifter B register Memory I/O channels

Program branch instructions require arithmetic to form the destination address from two operands. These 070-137 instructions do not alter the content of the accumulator. Execution of these sequences is performed with a separate background accumulator not visible to the programmer.

CARRY-BIT REGISTER

The carry bit register is a 1-bit register that holds the carry generated in the adder or shifter. The carry bit is treated as if it were bit 2^{16} of the accumulator operand and is included in all adds, subtracts, and shifts.

The carry bit can be set by several conditional instructions that test I/O channel flags. The carry bit is also used as a criterion for many conditional jump and return jump instructions.

ADDEND REGISTERS

The addend register supplies operands to the adder and shifter. Whenever two operands are required, the accumulator supplies one operand and the addend register supplies the other. The 16-bit addend register receives data from either the B register, the instruction stack, the operating registers, or memory. Its only destinations are the adder and the shifter.

INPUT/OUTPUT SECTION 5

INTRODUCTION

The I/O Processor supports up to 40 channels for input or output use. There are six Direct Memory Access (DMA) ports to I/O memory, which can be used by these 40 channels. Some of the channels and one of the ports are assigned standard purposes for the system, but the remainder are free for peripheral device or Central Processing Unit support. This section describes I/O configuration, speeds, and channel characteristics as well as the interrupt scheme used. The standard channels are also covered in this section. Table 5-1 summarizes characteristics of the IOP input/output section.

Table 5-1. Characteristics of the IOP input/output section

- Supported by 6 full-duplex direct-memory-access ports
- Approximately 850 megabits/s per DMA port (maximum speed)
- 16 data bits, 2 status bits (Busy and Done)
- Channel number selected by instruction or register contents
- Simultaneous input and output via separate ports

I/O CONFIGURATION

The I/O channels are numbered octally; in general, an input channel of a channel pair is given an even number. The 12 standard channels (numbers $0\text{-}13_8$) are the same among all of the I/O Processors of the computer system. Channels 014_8 through 017_8 may be channel options or may be an optional group of channels. The 020_8 through 047_8 channels are implemented as optional groups of four channels. Channels 014_8 through 047_8 are variable among the I/O Processors, depending on the overall system configuration. Five DMA ports to the I/O memory are available to the interfaces associated with channels 014_8 through 047_8.

(The sixth DMA port connects to the Buffer Memory.) Faster devices connect to an I/O Processor via the DMA ports, which allow block transfers. The slower devices can be supported via the accumulator channels. Several devices may be interfaced to share a single DMA port among the devices while each device has a unique channel number. This method is used in supporting groups of four disk storage units and Block Mux Channels.

I/O SPEEDS

The DMA ports are each capable of transferring a block of data at the approximate rate of 850 Mbits per second. Only 3 DMA ports can be active at once. DMA ports may be transferring data into I/O Memory, while other DMA ports simultaneously are transferring data from I/O Memory. This gives a maximum memory data rate of approximately 2.56 billion bits a second.

The maximum speed of accumulator channels depends on the speed of the interrupt service routine.

CHANNEL CHARACTERISTICS

The operating characteristics for the accumulator channels and the channels using DMA ports are similar in many respects. The following descriptions outline the control and data signals that are used and give the requirements for each signal. The channels are described independently of the interfaces that may be connected to them.

ACCUMULATOR CHANNELS

Each accumulator channel uses the following signals:

Function designators
Function strobe
Accumulator data
Busy/done flag to carry bit
Read done
Read busy
Master clear
Clock
Interrupt

Within each interface are two 1-bit registers comprising the done and busy flags for the channel. The interface is able to set or clear these flags, and the I/O Processor can sample them by means of the read done and read busy control signals.

Function Designators

Bits 2^0 - 2^3 of the f field of I/O instructions are used as a function code ($0 - 17_8$) to an interface (see the Instructions section in this part of the manual). This function is interpreted by the interface in a manner unique to the interface. The same function code can mean entirely different operations to different interfaces. The function code is 4 bits sent on lines from the I/O Processor instruction logic to the interface, prior to any other channel action. The function code may be stable on the lines for only 1 CP.

Function Strobe

The function strobe signal accompanies the function code bits. This signal alerts the interface to the presence of the function code.

Accumulator Data

The use of the accumulator data depends on the function code and the particular interface. For example, it may be treated as a parameter for a Buffer Memory transfer, or as a character for a display. The outgoing data from the accumulator is reliable only for the clock period containing the function strobe signal. Data coming from the interface to the accumulator also depends on the function code and interface.

Read Done

When the read done signal is sent to the interface, the interface done flag is sent back to the accumulator carry bit. The done flag is carried on the busy/done signal line described below.

Read Busy

When the read busy signal is sent to the interface, the interface busy flag is sent back to the accumulator carry bit. The busy flag is carried on the busy/done signal line described below.

Busy/Done

The busy signal is a response to the read busy or read done signals to the interface. If the read busy signal has been received, the interface busy flag is copied to the busy/done line. If the done flag has been requested, the interface done flag is sent on the busy/done line. The busy/done flag simply carries the set or cleared state of the requested flag. The signal arrives back at the I/O Processor to enter the carry bit position. It becomes the new carry bit in the same clock period that the read busy/done instruction issues.

Master Clear

A master clear signal is sent to each interface when the I/O Processor is deadstarted. After that, the only master clear function would have to be an interface interpretation of one of the function codes as a master clear command.

Clock

The I/O Processor clock signal is sent to each interface to synchronize the logical operations. The clock signal is a pulse approximately 12.5 nanoseconds wide. The clock speed can be varied for maintenance, in which case the pulse width stays constant. This clock is similar to the clock used in the Central Processor; however, the two clock generators are independent.

Interrupt

The interrupt signal from an interface to the I/O Processor causes an interrupt request in the I/O Processor for the channel. An interrupt occurs if all of the following are true: (1) the interrupt condition remains present, (2) the interrupt enable flag is set for the interface channel, and (3) the system interrupt enable is set.

CHANNELS USING A DMA PORT

A channel using a DMA port is an accumulator channel with connections to the I/O Memory. In addition to the accumulator channel signals already described, the DMA channel also uses the following signals which are described next:

I/O Memory data
I/O Memory address
Request read
Request write
Acknowledge write

I/O Memory Data

A group of 16 lines carries data from I/O Memory to the interface. A second group of 16 lines carries interface data to the I/O Memory. Data is transferred in groups of four 16-bit parcels, one parcel per clock period in sequential clock periods. When writing data into I/O Memory, the first parcel must be sent in the clock period after the acknowledge write is received from the I/O Processor. When reading data from I/O Memory, the data is received 7 CPs after the acknowledge read signal. In either case, the data is reliable for only 1 CP.

I/O Memory Address

The I/O Memory address comes from the interface to the I/O Processor on 14 lines. The interface receives the address from the accumulator under a specific function code, or the interface may be intelligent enough to create its own address. Any transfer of four parcels must have an address, which is the I/O Memory address for the first parcel of the 4-parcel group. The I/O Processor logic then takes care of incrementing the address for the later three parcels of the group. The I/O Memory address must be stable during the clock period of the request read or the request write signals.

Request Read

The request read signal is sent from the interface to the I/O Processor to take data from I/O Memory. This signal accompanies the I/O Memory address bits to the I/O Processor.

Request Write

The request write signal is sent from the interface to the I/O Processor when data is to be sent to I/O Memory. This signal accompanies the I/O Memory address bits to the I/O Processor.

Acknowledge Read

The acknowledge read signal is sent from the I/O Processor when the read requested by the interface can be performed. It has a minimum delay after the request read of 1 CP, but the delay may be stretched by memory conflicts. It occurs 7 CPs before the first parcel of data of the 4-parcel group and lasts 1 CP.

Acknowledge Write

The acknowledge write signal is sent from the I/O Processor when the requested write can be performed. It has a minimum delay after the request write signal of 1 CP, but memory conflicts may increase the delay. The interface must place the first parcel of data on the lines to the I/O Processor in the clock period after the acknowledge write signal is received at the interface.

READ SEQUENCE

The interface begins the read from I/O Memory after being issued an I/O instruction that commands the interface to start a transfer from I/O Memory to a peripheral. An address may be in the accumulator when the instruction issues and the interface may take that address as the starting address. Then, the interface sends a request read signal and the address bits to the I/O Processor. After a minimum delay of 1 CP, the acknowledge read signal is received at the interface. After 7 CPs, the interface takes the first parcel. It then takes the three following parcels in 3 CPs. To read another four parcels from I/O Memory to the interface, another request read and address must be sent from the interface.

WRITE SEQUENCE

The interface begins the write into I/O Memory after being issued an I/O instruction that commands the interface to start a transfer from a peripheral to I/O Memory. The address for the transfer may be contained in the accumulator at the time of instruction issue, or other means may be used to generate the address. Then, the interface sends a request write signal and the address bits to the I/O Processor. After a minimum delay of 1 CP, the acknowledge write signal is received from the I/O Processor. The interface must send the first parcel of data to the I/O Processor in the clock period after the acknowledge write signal was received. Each of the next 3 CPs transfers another parcel into the I/O Memory.

If another group of four parcels is to be written into I/O Memory, another request write and address must be sent from the interface.

STANDARD CHANNELS

Standard functions are assigned to 12 of the 40 available channels, which are the same for all I/O Processors. The interface logic has been built into each I/O Processor to handle each standard channel. The functions, channel numbers, and A Programming Machine Language (APML) mnemonics are listed in table 5-2. Another six standard channels interconnect the system I/O Processors as shown in the table. The standard channels are all accumulator channels except for the one DMA port used by the Buffer Memory.

Table 5-2. I/O Processor standard channel assignments

	Channel Number	Mnemonic		Function
All IOPs	000	IOR		Interrupt request
	001	PFR		Program fetch request
	002	PXS		Program exit stack
	003	LME		I/O Memory error
	004	RTC		Real-time clock
	005	MOS		Buffer Memory interface
MIOP 0	006	AIA	BIOP	I/O Processor input
	007	AOA		I/O Processor output
	010	AIB	DIOP	I/O Processor input
	011	AOB		I/O Processor output
	012	AIC	XIOP or 2nd DIOP	I/O Processor input
	013	AOC		I/O Processor output

Table 5-3 lists the functions for each of the standard channels. These functions are explained in subsequent paragraphs.

Table 5-3. Standard channel functions

Device	Mnemonic	Function
I/O REQUEST CH. 0	IOR : 10	Read interrupt channel number
PROGRAM FETCH REQUEST CH. 1	PFR : 0	Clear the program fetch request flag
	PFR : 6	Clear the channel interrupt enable flag
	PFR : 7	Set the channel interrupt enable flag
	PFR : 10	Read the operand register number
PROGRAM EXIT STACK CH. 2	PXS : 0	Clear the exit stack boundary flag
	PXS : 6	Clear the channel interrupt enable flag
	PXS : 7	Set the channel interrupt enable flag
	PXS : 10	Read exit stack pointer, E
	PXS : 11	Read exit stack address, (E)
	PXS : 14	Enter exit stack pointer, E
	PXS : 15	Enter exit stack address, (E)
I/O MEMORY ERROR CH. 3	LME : 0	Clear the I/O Memory parity error flag
	LME : 6	Clear the channel interrupt enable flag
	LME : 7	Set the channel interrupt enable flag
	LME : 10	Read error information
REAL-TIME CLOCK CH. 4	RTC : 0	Clear the channel done flag
	RTC : 6	Clear the channel interrupt enable flag
	RTC : 7	Set the channel interrupt enable flag
	RTC : 10	Read real-time clock
BUFFER MEMORY CH. 5	MOS : 0	Clear the channel busy and done flags
	MOS : 1	Enter the I/O Memory address for next transfer
	MOS : 2	Enter upper portion of Buffer Memory address
	MOS : 3	Enter lower portion of Buffer Memory address
	MOS : 4	Read Buffer Memory to I/O Memory
	MOS : 5	Write Buffer Memory to I/O Memory
	MOS : 6	Clear the channel interrupt enable flag
	MOS : 7	Set the channel enable interrupt flag
	MOS : 14	Set the control flags
I/O PROCESSOR INPUT (AIA-AIC) CH. 6, 10, 12	AIA : 0	Clear the channel done flag
	AIA : 6	Clear the channel interrupt enable flag
	AIA : 7	Set the channel interrupt enable flag
	AIA : 10	Read input to accumulator and resume channel
I/O PROCESSOR OUTPUT (AOA-AOC) CH 7, 11, 13	AOA : 0	Clear the channel busy and done flags
	AOA : 1	Enter control bits from accumulator
	AOA : 6	Clear the channel interrupt enable flag
	AOA : 7	Set the channel interrupt enable flag
	AOA : 14	Set the channel busy flag and output accumulator data

CHANNEL FOR I/O REQUESTS (CH. 0)

The I/O Processor has the 0 channel number reserved for reading the interrupt requests. The only function implemented is the following.

IOR: 10 Read interrupt channel number

This function replaces the accumulator content with the highest priority channel number currently requesting an interrupt. The channel number is loaded into the low-order 9 bits of the accumulator content. The high-order bits are forced to 0. Only the four least significant bits are used for the channel number. A zero value means there are no unhandled channel interrupts.

The interface register used by this channel contains the number of the highest priority channel on which an interrupt is present. The value is changed either by clearing the interrupt enable flag for the appropriate channel or by clearing the done flag for that channel.

For this channel the done flag is always set and the busy flag is always cleared. This channel can be used with the 040-043 instructions to set or clear the carry flag.

CHANNEL FOR PROGRAM FETCH REQUEST (CH. 1)

The I/O Processor has an I/O channel that responds to the program fetch request flag when that flag is set. (The program fetch request flag is explained in the IOP control section of this manual.) This channel provides a mechanism for calling the I/O Processor monitor program when a new section of program code is required.

A 9-bit interface register holds the operand register number associated with the interrupt request. This register is cleared and a new number entered at the time the program fetch request flag is set.

PFR : 0§ Clear the program fetch request flag. This flag is treated as the channel done flag. There is no busy flag for this channel.

PFR : 6§§ Clear the channel interrupt enable flag. The program fetch request flag is not altered in this process.

PFR : 7§§ Set the channel interrupt enable flag. The program fetch request flag is not altered in this process.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR:10).

PFR : 10 Replace the content of the accumulator with the content of the interface register. The interface register content is loaded into the low-order 9-bit positions in the accumulator. The high-order bits are forced to 0. The content of the interface register remains unchanged until a new PFR occurs. The done flag is cleared.

CHANNEL TO PROGRAM EXIT STACK (CH. 2)

The I/O Processor has an I/O channel connected to the program exit stack hardware. This channel provides the monitor program with the information needed to reorganize the content of the program exit stack when the stack overflows.

PXS : 0§ Clear the exit stack boundary flag. This flag is treated as the channel done flag. There is no busy flag for this channel.

PXS : 6§§ Clear the channel interrupt enable flag. The exit stack boundary flag is not altered in this process.

PXS : 7§§ Set the channel interrupt enable flag. The exit stack boundary flag is not altered in this process.

PXS : 10 Load the E designator into the low-order 4 bits of the accumulator. The high-order bits of the accumulator are forced to 0.

PXS : 11 Replace the accumulator content with the content of the program exit stack address currently pointed by the E designator.

PXS : 14 Replace the E designator with the low-order 4 bits of the accumulator content.

PXS : 15 Enter the program exit stack with the accumulator contents at the address currently pointed by the E designator.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR:10).

CAUTION

The exit stack is both an I/O device and an integral part of the processor. Return and exit instructions and interrupts use the exit stack values immediately. The I/O channel access to the exit stack takes 4 CPs to complete. Time must be allowed for the I/O channel to complete the transfer before using the exit stack value by a return, exit or interrupt.

Any attempt to use a value changed in the stack within 5 CPs after it was changed has potentially disastrous effects on the program execution sequence.

A simple way to provide delay is to change the exit stack, then read the value back from the exit stack to the accumulator. Then do the exit or return.

For example:

NOT RECOMMENDED	RECOMMENDED
(E) = AAAB	(E) = AAAB
EXIT	A = (E) (delay)
	EXIT
or: A = AAAB	A = AAAB
PXS : 15	PXS : 15
EXIT	PXS : 11
	EXIT

Deadstart Sequence

On master clear, stack location 0 is cleared to a zero value. On the deadstart interrupt, P is therefore set to 0 and program execution begins at memory location 0. The exit stack pointer, E, is set to the current value plus 1 (wrapping around to 0 if the current value is 15.

CHANNEL FOR I/O MEMORY ERROR (CH. 3)

The I/O Processor has an I/O channel connected to the error detection circuits in the I/O Memory. This channel provides the error indication in the event of memory malfunction and provides maintenance information. The intent of this channel is to provide information helpful to the maintenance function as quickly as possible. No attempt at continued operation is expected beyond system shutdown.

LME : 0 Clear the I/O Memory parity error flag.

LME : 6§ Clear the channel interrupt enable flag.

LME : 7§ Clear the channel interrupt enable flag.

LME : 10 Read error information into the five lowest-order positions of the accumulator as follows.

Bit	Meaning
2^0	Bank number 2^0
2^1	Bank number 2^1
2^2	Section number 2^0
2^3	Section number 2^1
2^4	Error bit: 0 = Error in 2^0-2^7 byte 1 = Error in 2^8-2^{15} byte

CHANNEL TO REAL-TIME CLOCK (CH. 4)

The I/O Processor real-time clock (RTC) is a 17-bit counter/timer which interrupts the I/O Processor at 1-millisecond intervals. The real-time clock increments every clock period. Upon reaching a count of 234177_8, it sets the RTC channel done flag, clears to 0, and continues incrementing. There is no busy flag for this channel. Since the accumulator only holds 16 bits, the low-order bit of the counter is ignored, giving a timing accuracy of 2 CPs, and a count of 116077_8.

§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

There is no capability to set the real-time clock. To time an interval, the RTC must be read at the beginning and end of an interval. Thus to time an interval, the program must:

- Read the clock at the beginning of the interval,
- Store the value in a register and
- Read the clock at the end of the interval.

This instruction timing adds 8 CPs (4 clock counts) to the measured sequence. The algorithm for determining an interval without RTC interrupts is:

$$\text{Time (ns)}_8 = (\text{RTC ending}_8 - \text{RTC beginning}_8 - 4) \times 2 \text{ (CP ns)}$$

If RTC interrupts do occur during the interval, the algorithm becomes:

$$\text{Time (ns)}_8 = ((\text{RTC ending}_8 - \text{RTC beginning}_8 - 4) + (116077_8 \times \text{number of interrupts})) \times 2 \text{ (CP ns)}$$

For time intervals expected to be less than 1 millisecond, synchronize their beginnings with the occurrence of an RTC interrupt, thus eliminating the possibility of an RTC interrupt occurring sometime during the timing sequence.

The commands for the real-time clock channel are as follows:

RTC : $0^{\S}$	Clear the channel done flag.
RTC : $6^{\S\S}$	Clear the channel interrupt enable flag.
RTC : $7^{\S\S}$	Set the channel interrupt enable flag.
RTC : 10	Read the real-time clock count into the accumulator.

CHANNEL TO BUFFER MEMORY (CH 5.)

The I/O Processor has an I/O channel connected to the Buffer Memory. This channel allows the I/O processor program to transfer blocks of data in either direction between its I/O Memory and the Buffer Memory. The function requests for this channel are summarized below.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

This channel has three interface registers used to control the block copy operations. The Buffer Memory address is held in a 24-bit register. The high-order 15 bits of this address are entered with a function 2 request. The low-order 9 bits are entered with a function 3 request. The I/O Memory address is held in a 14-bit register which is entered with a function 1 request. The block length in Buffer Memory words is held in a 14-bit register. This register is entered with a function 4 or 5 request.

The I/O Memory address in the channel register is forced to a value that is a multiple of four. This is done by forcing the 2 low-order bits of the register content to zero values. This provision allows the maximum data rate to the I/O Processor I/O Memory.

All three register values are cleared to 0 at the end of a block copy. A zero value will be used if a register entry is omitted in the next sequence.

At the end of a read transfer (done flag set), the busy flag is left set if a multiple bit error occurred in the transfer.

MOS : 0§ Clear the channel busy and channel done flags.

MOS : 1 Enter the accumulator content in the channel interface register for the I/O Memory address. The 2 low-order bits are forced to 0.

MOS : 2 Enter the 15 low-order bits of the accumulator content as the 15 high-order bits of the Buffer Memory address. See figure 5-1.

MOS : 3 Enter the 9 low-order bits of the accumulator content as the 9 low-order bits of the Buffer Memory address. See figure 5-1.

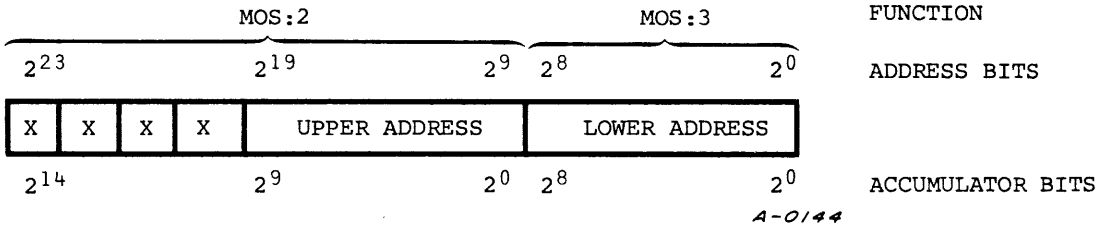


Figure 5-1. Buffer Memory address formation

§ Allow 1 CP before checking busy or done.

MOS : 4[§] Initiate the transfer of the block of data from the Buffer Memory to the I/O Processor I/O Memory. The channel busy flag is set and the channel done flag is cleared by the function request. The 14 low-order bits of the accumulator content at the time of the function request is entered in the channel interface block length register. The channel done flag is set and the channel busy flag is cleared when the block transfer has been completed. If the block length register contains a zero value, the block length is set to 65,536 parcels (full I/O Memory). The busy flag is left set if a multiple-bit error occurred in the transfer. Single-bit errors are automatically corrected.

MOS : 5[§] Initiate the transfer of a block of data from the I/O Processor I/O Memory to the Buffer Memory. The channel busy flag is set and channel done flag is cleared by the function request. The 14 low-order bits of the accumulator content at the time of the function request is entered in the channel interface block length register. The channel done flag is set and the channel busy flag is cleared when the block transfer has been completed.

MOS : 6[§§] Clear the channel interrupt enable flag.

MOS : 7[§§] Set the channel interrupt enable flag.

MOS : 14 Enter the 3 low-order bits of the accumulator content in the control register. This is used for diagnostic purposes only.

2^0 = Disable error correction
2^1 = Disable write check bits
2^2 = Disable refresh

Setting a control bit to 0 enables error correction, write check bits, or refresh.

---

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

Error Handling

When an error occurs, the busy and done flags are both set. A MOS : 0 command must be issued to the channel to clear the busy and done flags before another read or write is initiated.

Buffer Memory Interface Deadstart

The Buffer Memory interface has provisions for deadstarting the I/O Processor. If the I/O processor is master cleared with the deadstart signal set, the I/O Memory address register, the Buffer Memory address register, and the block length register are cleared. This sets up a 65K parcel transfer. A MOS : 4 read Buffer Memory function is initiated when the I/O Processor master clear signal goes to 0. The interrupt enable flag is set (MOS : 7), and when the transfer is complete, the I/O Processor interrupts.

Buffer Memory Interface Dead Dump

The Buffer Memory interface also provides for a dead dump of the I/O Processor. If the I/O Processor is master cleared with the dead dump signal set, the I/O Memory address, Buffer Memory address and the block length registers are cleared. A MOS : 5 Buffer Memory function write is initiated when master clear goes to 0. The interrupt enable flag is cleared (MOS : 6) and no interrupt occurs when the transfer completes.

CHANNEL FOR I/O PROCESSOR INPUT (CH. 6, 10, 12)

The I/O Processor has three input channels for connection to other I/O Processors. These provide a communication link between I/O Processors. These channels have no busy flags. Each channel uses a single 16-bit register to hold the data parcel being transferred. The register is cleared as soon as the data enters the receiving accumulator.

AIA : 0[§] Clear the channel done flag. There is no busy flag for this channel.

AIA : 6[§§] Clear the channel interrupt enable flag.

AIA : 7[§§] Set the channel interrupt enable flag.

AIA : 10 Read the accumulator data of the other I/O Processor into the accumulator of this I/O Processor, and resumes the channel. The data does <u>not</u> remain in the interface register after it has been read.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

CHANNEL FOR I/O PROCESSOR OUTPUT (CH. 7, 11, 13)

The I/O Processor has three output channels that connect to other I/O Processors. These provide a communications link between I/O Processors, and provide the ability to master clear, deadstart, or dead dump another I/O Processor.

A 3-bit control register receives the 3 low-order bits of the sending I/O Processor accumulator and causes the required action. A 16-bit register holds transmitted data until it is loaded into the receiving accumulator, at which time the data register is cleared.

A0A : 0§§ Clear the channel busy and done flags.

A0A : 1 Enter the 3 low-order accumulator bits into the control register. A set bit in the register positions causes the following actions:

Bit	Meaning
2^0	Master clear
2^1	Deadstart
2^2	Dead dump

To perform a deadstart from Buffer Memory, set the master clear and deadstart control bits simultaneously and keep them set for at least 200 nanoseconds. Then clear both bits. The deadstart control bit has no effect without the master clear control bit. The deadstart transfer is initiated with a block length of 65K parcels starting at I/O Memory address 0. In approximately 2 milliseconds the transfer completes and then interrupts the I/O Processor.

To perform a dead dump from I/O Memory to Buffer Memory, set the master clear and dead dump control bits simultaneously and keep them set for at least 200 nanoseconds. Then clear both bits. The dead dump control bit has no effect without the master clear control bit. The dead dump transfer initiates with a block length of 65K parcels starting with I/O Memory address 0 contents going into Buffer Memory address 0. The dead dump completes in approximately 2 milliseconds, but does not interrupt the I/O Processor at completion.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

A0A : 6[§§]	Clear channel interrupt enable flag.
A0A : 7[§§]	Set channel interrupt enable flag.
A0A : 14[§]	Set the channel busy flag and output accumulator data. When data is accepted by receiving I/O Processor the busy flag clears and the done flag sets.

INTERRUPT SEQUENCE

The I/O Processor program is interrupted for service by a monitor program when an interrupt request is present and the system interrupt enable flag is set. The system interrupt enable flag applies to the entire I/O Processor, as contrasted to the channel interrupt enable flags that only apply to the particular channel. The 003 I = 1 instruction sets the system interrupt enable flag. The interruption occurs upon completion of an instruction in the currently executing program.

A jump or exit instruction must be completed before interrupts are actually set. Therefore, interrupts are not actually enabled until the first non-branching instruction is executed.

The interrupt sequence begins by the hardware clearing of the system interrupt enable flag to prevent further interrupts. The address for the next instruction in the interrupted program is stored in the exit stack. The entry in the exit stack is made by first advancing the E register by one count, and then entering the exit stack at the newly pointed position. This is the same sequence which occurs on a return jump execution. The execution of the interrupted program is then suspended and a new program sequence initiated. The address for beginning the monitor program sequence is obtained from the 0 position in the exit stack, without using the E register.

The monitor program, when finished, restores the system interrupt enable flag and exits to the interrupted program as if it were a subprogram call. The last two statements in the monitor program should be the equivalent of the following:

I = 1		I = 1 (Set system interrupt enable flag)
EXIT	or	P = dd. (Exit or jump to interrupted program)

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

When issuing an I=1, the system interrupt enable is delayed until the next non-branch or non-I/O instruction is issued. The instructions that do not enable interrupts are the 40-43 and 70-137. This allows the executive/monitor to get back to the interruptible activity before an interrupt is accepted.

An I=0 instruction should be used at the interrupt handler entrance. If a redundant I=1 is executed and an interrupt occurs before a non-branch or non-I/O instruction is encountered, the interrupt handler will be entered (with interrupts disabled). But interrupts will be re-enabled when the first non-branch or non-I/O instruction is issued within the interrupt handler.

After issuing a command 6 or 7 to any I/O channel, allow 3 clock periods before seeing its effect on system interrupt. (Assuming system interrupts are, or will be, enabled.)

IOP INSTRUCTIONS 6

INSTRUCTION FORMAT

Each I/O Processor instruction occupies one or two 16-bit parcels in the I/O Memory. The instruction consists of two designators and a constant field as illustrated in figure 6-1.

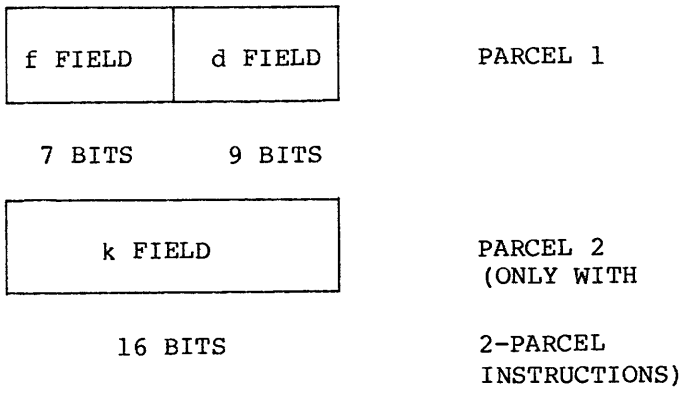


Figure 6-1. Instruction format

The f designator is the instruction function code and specifies which of the instructions in the machine repertoire is intended for execution. The d designator has several uses, depending on the instruction function, but in general specifies where in the machine resources the function of the f designator is to be performed. The d designator may be thought of as a displacement specification and is used in several ways:

- To point to a specific operand register where one is required,
- To specify the amount of the displacement of data in a shift instruction,
- To specify the amount of displacement forward or backward in program code for a branch instruction,
- As an operand value.

The d designator is always treated as a 9-bit positive integer. Small integers may be entered directly into the computation from instructions using the d designator as a constant. Separate instructions are provided to add or subtract this 9-bit constant rather than to consider it as a sign extended quantity. A branch instruction may designate a forward or a backward displacement from the current program location. Separate instructions are provided for the forward and backward jumps using the d designator as a 9-bit displacement magnitude.

Certain instructions use the program parcel immediately following the instruction as a constant field, designated k. These instructions may be considered to be 2-parcel instructions.

INSTRUCTION DESCRIPTIONS

The I/O Processor instruction repertoire is described on the following pages. The mnemonics for each instruction are pseudo machine language statements representing the individual operations performed. Special symbols used in the descriptions are:

>	Shift right
<	Shift left
>>	Shift right circular
<<	Shift left circular
&	Logical product
#	Not equal to
dd	Contents of operand register specified by d field
(dd)	Contents of memory location specified by dd
iod	3-character channel mnemonic, i.e., IOR, PXS,...
B	Contents of register B
(B)	Contents of operand register specified by B register

The following descriptions explain the results of each instruction and show the steps and times involved in executing the instruction.

000 Instruction

PASS

This instruction performs no operation. It is used to fill program fields with null operations where desired.

CP 0 Issue.

001 Instruction

EXIT

The EXIT instruction terminates execution of the current program sequence and returns to the sequence that was suspended in calling this subprogram. The current P register value is discarded. The beginning address for the reinitiated sequence is obtained from the program exit stack at the location currently pointed by E. The value of E is then decremented by 1. The decrementing is blocked and the exit stack boundary flag is set if the value of E was previously 0. The exit stack boundary flag will cause an interrupt of the program sequence for restructuring the content of the program exit stack.

If the EXIT instruction follows a modification of the program exit stack or of the E pointer, at least 4 CPs must elapse between the last modification and the EXIT instruction. Reading the modified value back to the accumulator may be used as the necessary delay.

CP 0 Issue. Transmit exit stack data to P.
Decrement E.

002 Instruction

I = 0

This instruction clears the system interrupt enable flag.

CP 0 Issue. Clear system interrupt enable flag.

003 Instruction

I = 1

This instruction sets the system interrupt enable flag.

When issuing an I = 1, the system interrupts enable is delayed until the next non-branch or non-I/O instruction is issued. Instructions that do not enable interrupts after I = 1 issue are the 040-043 and 070-137 instructions. The delay in setting flag for this instruction allows the interrupt program to re-enable the interrupt mode and then exit to the interrupted program.

If the instruction following the I = 1 is an I = 0, the I = 0 takes precedence and system interrupts are disabled.

CP 0 Issue. Set delay interrupt enable flag.

004 Instruction

A = A > d

This instruction shifts the content of the accumulator and the associated carry flag to the right by d bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. Zero values are entered in the carry flag and propagated to the right as the shift progresses. No shift is performed if the shift count is 0. The accumulator and carry flag are cleared if the shift count is greater than 16 decimal.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit accumulator data and inverted d to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

005 Instruction

A = A < d

This instruction shifts the content of the accumulator and the associated carry flag to the left by d bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. Zero values are entered in the low-order bit positions of the accumulator and are propagated to the left as the shift progresses. Bits shifted from the carry flag are discarded. No shift is performed if the shift count is 0. The accumulator and carry flag are cleared if the shift count is greater than 17_{10}.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit accumulator data and d to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

006 Instruction

A = A >> d

This instruction shifts the content of the accumulator and the associated carry flag to the right in a circular mode by d bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. No bits are discarded in this shift mode. Bits shifted from the right end of the accumulator are returned to the carry flag. No shift is performed if the shift count is 0.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit accumulator data and inverted d to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

007 Instruction

A = A << d

This instruction shifts the content of the accumulator and the associated carry flag to the left in a circular mode by d bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. No bits are discarded in this shift mode. Bits shifted from the carry flag are returned to the low-order bit position in the accumulator. No shift is performed if the shift count is 0.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit accumulator data and d to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

010 Instruction

A = d

This instruction enters the d designator in the accumulator as a 9-bit positive integer and clears the carry flag. The high-order bits are 0.

CP 0 Issue. Transmit d to accumulator.

011 Instruction

A = A & d

This instruction forms the bit-by-bit logical product of the previous accumulator content and the d designator and places the result in the accumulator. The d designator is treated as a 9-bit positive integer. It then clears the carry flag.

The logical product of the previous accumulator content and the d operand from the instruction is formed at the input to the accumulator.

CP 0 Issue. Transmit d and accumulator to accumulator.

012 Instruction

A = A + d

This instruction adds the d designator to the previous accumulator content in the 16-bit twos complement mode. The d designator is treated as a 9-bit positive integer in this addition. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit data to adder. Add mode.

CP 2 Transmit adder result to accumulator.

013 Instruction

A = A - d

This instruction subtracts the d designator from the previous accumulator content in a 16-bit twos complement mode. The d designator is treated as a 9-bit positive integer in this operation. The subtraction is performed by complementing the content of the addend register and adding the result to the previous accumulator content. One is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator during either addition process.

CP 0 Issue. Transmit d to addend register.

CP 1 Transmit data to adder. Subtract mode.

CP 2 Transmit adder result to accumulator.

014 Instruction

A = k

This instruction enters a 16-bit constant in the accumulator and clears the carry flag. The constant is obtained from the next sequential parcel in the program field. The next instruction is obtained from the following parcel.

CP 0 Issue. Read next parcel out of instruction stack.

CP 1 Transmit k data to accumulator. The function of CP 1 is delayed if the next parcel of instruction buffer data is not available in the instruction stack.

015 Instruction

A = A & k

This instruction forms the bit-by-bit logical product of the previous accumulator content and a 16-bit constant and places the result in the accumulator. It then clears the carry flag. The constant is obtained from the next sequential parcel in the program field. The next instruction is obtained from the following parcel.

The logical product of the previous accumulator content and the operand constant is formed at the input to the accumulator.

CP 0 Issue. Read the next parcel out of the instruction stack.

CP 1 Transmit k data to accumulator. The function of CP 1 is delayed if the next parcel of instruction buffer data is not available in the instruction stack.

016 Instruction

A = A + k

This instruction adds a 16-bit constant to the previous accumulator content in a twos complement mode. The constant is obtained from the next sequential parcel in the program field. The next instruction is obtained from the following parcel. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Read next parcel out of the instruction stack.

CP 1 Transmit k data to addend register. The function of CP 1 is delayed if the next parcel of instruction buffer data is not available in the instruction stack.

CP 2 Transmit data to adder. Add mode.

CP 3 Transmit adder result to accumulator.

017 Instruction

A = A - k

This instruction subtracts a 16-bit constant from the previous accumulator content in a twos complement mode. The constant is obtained from the next sequential parcel in the program field. The next instruction is obtained from the following parcel. The subtraction is performed by complementing the constant and adding the result to the accumulator content. One is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator in either addition process.

This instruction is redundant in the sense that an equivalent function can be performed with the 016 instruction using a different constant. This instruction is included in the list for completeness in the translational pattern.

CP 0 Issue. Read next parcel out of the instruction stack.

CP 1 Transmit k data to addend register. The function of CP 1 is delayed if the next parcel of instruction buffer data is not available in the instruction stack.

CP 2 Transmit data to adder. Subtract mode.

CP 3 Transmit adder result to accumulator.

020 Instruction

A = dd

This instruction enters the content of operand register d in the accumulator and clears the carry flag.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator.

021 Instruction

A = A & dd

This instruction forms the bit-by-bit logical product of the previous accumulator content and the content of operand register d and places the result in the accumulator. It then clears the carry flag.

The logical product of the previous accumulator content and the operand register content is formed at the input to the accumulator. No additional time is required for this function.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator.

022 Instruction

A = A + dd

This instruction adds the content of operand register d to the previous accumulator content. The addition is performed in the twos complement mode. It complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to addend register.

CP 2 Transmit data to adder. Add mode. The data in the accumulator and addend register is transmitted to the adder in this clock period. The addend register is free in this clock period to accept another operand.

CP 3 Transmit adder result to accumulator.

023 Instruction

A = A - dd

This instruction subtracts the content of operand register d from the previous accumulator content. The subtraction is performed by complementing the subtrahend and adding the result to the accumulator content in ones complement mode. One is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to addend register.

CP 2 Transmit data to adder; subtract mode. The data in the accumulator and addend register is transmitted to the adder in this clock period. The addend register is free in this clock period to accept another operand.

CP 3 Transmit adder result to accumulator.

024 Instruction

dd = A

This instruction stores the accumulator content in operand register d.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit d data to RP and DP.

CP 1 Transmit accumulator data to operand register. The function of CP 1 is delayed if the data is not available in the accumulator during the indicated clock period. In this case, the pointer in RP and DP is held until the accumulator data is available.

025 Instruction

dd = A + dd

This instruction adds the content of operand register d to the previous accumulator content. The addition is performed in a twos complement mode. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process. It then replaces the content of operand register d with the new accumulator content.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit d data to RP and DP.

CP 1 Transmit operand register data to addend register. The RP pointer is discarded in CP 1 and reentered with the same pointer from DP in CP 3. The function at CP 1 is delayed if the data is not available in the accumulator.

CP 2 Transmit data to adder. Add mode.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit accumulator data to operand register.

026 Instruction

dd = dd + 1

This instruction replaces the content of operand register d with the previous content increased by 1. The result is left in the accumulator as well as in the operand register. The carry flag is cleared at the beginning of this operation. One is entered in the accumulator. The content of the operand register enters the addend register and is then added to the accumulator content in a twos complement mode. The carry flag is set if a carry is propagated from the accumulator in the addition process. The result is then returned to the operand register.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit d data to RP and DP.

CP 1 Transmit operand register data to addend register. Enter a +1 in the accumulator.

The RP pointer is discarded in CP 1 and reentered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit copy of accumulator data to operand register.

027 Instruction

dd = dd - 1

This instruction replaces the content of operand register d with the previous content decreased by 1. The result is left in the accumulator as well as in the operand register. The carry flag is cleared at the beginning of this operation. The accumulator bits are forced set. The content of the operand register is entered in the addend register and then added to the accumulator content. The carry flag is set if a carry is propagated from the accumulator in the addition process. The result is then returned to the operand register.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit d data to RP and DP.

CP 1 Transmit operand register data to addend register.
Enter a -1 in the accumulator.

The RP pointer is discarded in CP 1 and reentered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit copy of accumulator data to operand register.

030 Instruction

A = (dd)

This instruction enters the content of an I/O Memory location in the accumulator. The I/O Memory address is obtained from operand register d. It then clears the carry flag.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send Memory Read Reference. The function of CP 2 is repeated until the acceptance signal is received.

CP 3 Acceptance signal from I/O Memory. The function of CP 2 is repeated until the acceptance signal is received.

CP 4 -

CP 5 -

CP 6 Transmit memory data to accumulator.

031 Instruction

A = A & (dd)

This instruction forms the bit-by-bit logical product of the previous accumulator content and the content of an I/O Memory location and places the result if the accumulator. The I/O Memory address is obtained from operand register d. It then clears the carry flag.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send memory read reference. The function of CP 2 is repeated until the acceptance signal is received.

CP 3 Acceptance signal from I/O Memory.

CP 4 -

CP 5 -

CP 6 Transmit memory data to accumulator. The logical product of memory data and the accumulator content is done at the input to the accumulator. No additional time is required for this function.

032 Instruction

A = A + (dd)

This instruction adds the content of an I/O Memory location to the content of the accumulator. The I/O Memory address is obtained from operand register d. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read request to memory.

CP 3 Acceptance signal from I/O Memory.

CP 4 The function of CP 2 is repeated until the acceptance signal is received.

CP 5 -

CP 6 Transmit memory data to addend register.

CP 7 Transmit data to adder. Add mode.

The data from the accumulator and addend register is transmitted to the adder in CP 7. The addend register is free in this clock period to accept another operand.

CP 8 Transmit adder result to accumulator.

033 Instruction

A = A - (dd)

This instruction subtracts the content of an I/O Memory location from the content of the accumulator. The I/O Memory address is obtained from operand register d. The subtraction is performed by complementing the subtrahend and adding the result to the accumulator content in a ones complement mode. A 1 is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator during either addition process.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read request to memory.

CP 3 Acceptance signal from I/O Memory.

CP 4 The function of CP 2 will be repeated until the acceptance signal is received.

CP 5 -

CP 6 Transmit memory data to addend register.

CP 7 Transmit data to adder. Subtract mode.

The data from the accumulator and addend register is transmitted to the adder in CP 7. The addend register is free in this clock period to accept another operand.

The complementing of the addend data is accomplished in the transmission of the data from the addend register to the adder. The addition of +1 to the result is accomplished in this same clock period in the adder.

CP 8 Transmit adder result to accumulator.

034 Instruction

(dd) = A

This instruction replaces the content of an I/O Memory location with the current content of the accumulator. The I/O Memory address is obtained from operand register d.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted or the accumulator data is not available.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Transmit a copy of the accumulator data to bank operand registers. Send write request to memory.

The function of CP 2 is repeated until an acceptance signal is received.

CP 3 Acceptance signal from I/O Memory.

035 Instruction

(dd) = A + (dd)

This instruction replaces the content of an I/O Memory location with its previous content plus the current accumulator content. The I/O Memory address is obtained from operand register d. The addition is performed using the accumulator in a 16-bit twos complement mode. The carry flag is complemented if a carry is propagated from the accumulator in the addition process. The result is left in the accumulator as well as transmitted to the I/O Memory location.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted or the accumulator data is not available.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read requet to memory. The functions of CP 2 is repeated until an acceptance signal is received.

CP 3 Acceptance signal from I/O Memory. The MA data is held from CP 3.

CP 4 -

CP 5 -

CP 6 Transmit memory data to addend register.

CP 7 Transmit data to adder. Add mode.

CP 8 Transmit adder result to accumulator.

CP 9 Transmit MA data to bank address registers. Transmit a copy of the accumulator data to bank operand registers. Send write request to memory.

The function of CP 9 is repeated until an acceptance signal is received.

CP 10 Acceptance signal from I/O Memory.

036 Instruction

(dd) = (dd) + 1

This instruction increments the content of an I/O Memory location by 1. The I/O Memory address is obtained from operand register d. This operation is performed using the accumulator. The accumulator and carry flag are cleared. A +1 is entered in the accumulator. The content of the memory location is entered in the addend register and then added to the accumulator contents. The result is returned to I/O Memory. The result remains in the accumulator. The carry flag is complemented if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read request to memory. The function of CP 2 is repeated until an acceptance signal is received.

CP 3 Acceptance signal from I/O Memory. The MA data is held from CP 3.

CP 4 -

CP 5 -

CP 6 Transmit memory data to addend register. Clear carry flag and enter plus one in accumulator.

CP 7 Transmit data to adder. Add mode.

CP 8 Transmit adder result to accumulator.

CP 9 Transmit MA data to bank address registers. Transmit copy of accumulator data to bank operand registers. Send write request to memory. The function of CP 9 is repeated until the acceptance signal is received.

CP 10 Acceptance signal from I/O Memory.

037 Instruction

(dd) = (dd) - 1

This instruction decrements the content of an I/O Memory location by 1. The I/O Memory address is obtained from operand register d. This operation is performed using the accumulator. The carry flag is cleared. The accumulator bits are forced set. The content of the I/O memory location is entered in the addend register and then added to the accumulator content. The result is returned to the I/O memory location. The result also remains in the accumulator. The carry flag is complemented if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit d data to RP. The function of CP 0 is delayed if the MA data from a previous instruction has not been accepted.

CP 1 Transmit operand register data to MA.

CP 2 Transmit MA data to bank address registers. Send read request to memory. The functions of CP 2 is repeated until an acceptance signal is received.

CP 3 Acceptance signal from I/O Memory. The MA data is held from CP 3.

CP 4 -

CP 5 -

CP 6 Transmit memory data to addend register. Clear carry flag and enter all ones in accumulator.

CP 7 Transmit data to adder. Add mode.

CP 8 Transmit adder result to accumulator.

CP 9 Transmit MA data to bank address registers. Send write request to memory. Transmit copy of accumulator data to bank operand registers.

The function of CP 9 is repeated until the acceptance signal is received.

CP 10 Acceptance signal from I/O Memory.

040 Instruction

C = 1, iod = DN

This instruction forces the carry flag to the same state as the channel d done flag.

Another instruction that would alter the state of the carry flag before CP 4 cannot issue in CP 1 or CP 2. A delay of 1 CP must be inserted if an I/O instruction is issued that alters the Busy or Done flags before the 40 instruction.

Channel 000 is always done. The carry flag can be set by setting d = 000 in this instruction.

CP 0 Issue. Transmit d designator to I/O channels.

CP 1 -

CP 2 -

CP 3 Force state of carry flag.

041 Instruction

C = 1, iod = BZ

This instruction forces the carry flag to the same state as the channel d busy flag.

Another instruction that would alter the state of the carry flag before CP 4 cannot issue in CP 1 or CP 2. A delay of 1 CP must be inserted if an I/O instruction is issued that alters the Busy or Done flags before the 41 instruction.

Channel 000 is never busy. The carry flag can be forced clear by setting d to 000 in this instruction.

CP 0 Issue. Transmit d designator to I/O channels.

CP 1 -

CP 2 -

CP 3 Force state of carry flag.

042 Instruction

C = 1, IOB = DN

This instruction forces the carry flag to the same state as the done flag of the channel specified by the contents of the B register.

Another instruction that would alter the state of the carry flag before CP 4 cannot issue in CP 1 or CP 2. A delay of 1 CP must be inserted if an I/O instruction is issued that alters the Busy or Done flags before the 42 instruction.

This instruction cannot issue if the B register is reserved.

Channel 000 is always done. The carry flag can be forced clear by setting B to 000 in this instruction.

CP 0 Issue. Transmit B designator to I/O channels.

CP 1 -

CP 2 -

CP 3 Force state of carry flag.

043 Instruction

C = 1, IOB = BZ

This instruction forces the carry flag to the same state as the busy flag of the channel specified by the contents of the B register.

Another instruction that would alter the state of the carry flag before CP 4 cannot issue in CP 1 or CP 2. A delay of 1 CP must be inserted if an I/O instruction is issued that alters the Busy or Done flags before the 43 instruction.

This instruction cannot issue if the B register is reserved.

Channel 000 is never busy. The carry flag can be forced clear by setting B to 000 for use by this instruction.

CP 0 Issue. Transmit B designator to I/O channels.

CP 1 -

CP 2 -

CP 3 Force state of carry flag.

044 Instruction

A = A > B

This instruction shifts the content of the accumulator and the associated carry flag to the right by B bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. Zero values are entered in the carry flag and propagated to the right as the shift progresses. No shift is performed if the shift count is 0. The accumulator and carry flag are cleared if the shift count is greater than 17_{10}.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data and inverted B to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

045 Instruction

A = A < B

This instruction shifts the content of the accumulator and the associated carry flag to the left by B bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. Zero values are entered in the low-order bit positions of the accumulator and are propagated to the left as the shift progresses. Bits shifted from the carry flag are discarded. No shift is performed if the shift count is 0. The accumulator and carry flag are cleared if the shift count is greater than 17_{10}.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data and B to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

046 Instruction

A = A >> B

This instruction shifts the content of the accumulator and the associated carry flag to the right in a circular mode by B bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. No bits are discarded in this shift mode. Bits shifted from the right end of the accumulator are returned to the carry flag. No shift is performed if the shift count is 0.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data and inverted B to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

047 Instruction

A = A << B

This instruction shifts the content of the accumulator and the associated carry flag to the right in a circular mode by B bit positions. The carry flag may be regarded as a seventeenth bit to the left of the accumulator content for this operation. No bits are discarded in this shift mode. Bits shifted from the right end of the accumulator are returned to the carry flag. No shift is performed if the shift count is 0.

The low-order 5 bits of the addend register content are interpreted in determining the shift count. High-order bits are ignored.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data and B to shifter. Shift mode. The addend register is free in this CP.

CP 2 Transmit shifter result to accumulator.

050 Instruction

A = B

This instruction enters the B register content in the accumulator as a 9-bit positive integer and clears the carry flag. The high-order bits are 0.

CP 0 Issue. Transmit B to accumulator.

051 Instrucion

A = A & B

This instruction forms the bit-by-bit logical product of the previous accumulator content and the B register content, and places it in the accumulator. The B register content is treated as a 9-bit positive integer. The instruction clears the carry flag.

The logical product of the previous accumulator content and the operand from the instruction is formed at the input to the accumulator.

CP 0 Issue. Transmit B and accumulator to accumulator.

052 Instruction

A = A + B

This instruction adds the B register content to the previous accumulator content in the 16-bit twos complement mode. The B register content is treated as a 9-bit positive integer in this addition. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data to adder. Add mode.

CP 2 Transmit adder result to accumulator.

053 Instruction

A = A - B

This instruction subtracts the B register content from the previous accumulator content in a 16-bit twos complement mode. The B register content is treated as a 9-bit positive integer in this operation. The subtraction is performed by complementing the content of the addend register and adding the result to the previous accumulator content. One is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator during either addition.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data to adder. Subtract mode.

CP 2 Transmit adder result to accumulator.

054 Instruction

B = A

This instruction replaces the B register content with the low-order 9 bits of the accumulator content.

This instruction cannot issue until the accumulator sequence has been completed for any previous instructions.

CP 0 Issue. B register is free.

CP 1 Issue + 1 (is accumulator ready).

CP 2 Issue + 2 (transmit accumulator data to B register).

055 Instruction

B = A + B

This instruction adds the content of the B register to the previous accumulator content. The B register content is treated as a 9-bit positive integer in this operation. The addition is performed in a 16-bit twos complement mode. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process and then replaces the B register content with low-order 9 bits of the accumulator content.

CP 0 Issue. Transmit B to addend register.

CP 1 Transmit data to adder. Add mode.

CP 2 Transmit adder result to accumulator. The addend register is free in this CP.

CP 3 Transmit a copy of the accumulator data to B register.

056 Instrucion

B = B + 1

This instruction replaces the content of the B register with its previous content increased by 1. The result is left in the accumulator as well as in the B register. The carry flag is cleared at the beginning of this operation. One is entered in the accumulator. The content of the B register is then added to the accumulator content in a 16-bit twos complement mode. The B register content is treated as a 9-bit positive integer in this process. The low-order 9 bits of the accumulator content are then returned to the B register.

CP 0 Issue. Transmit B to addend register. Enter a +1 in the accumulator.

CP 1 Transmit data to adder. Add mode. The addend register is free in this CP.

CP 2 Transmit adder result to accumulator.

CP 3 Transmit a copy of the accumulator data to B register.

057 Instruction

B = B - 1

This instruction replaces the content of the B register with its previous content decreased by 1. The result is left in the accumulator as well as in the B register. The carry flag is cleared at the beginning of this operation. The accumulator bits are forced set. The content of the operand register is entered in the XB register and then added to the accumulator content. The B register content is treated as a 9-bit positive integer in this process. The carry flag is complemented if a carry is propagated from the accumulator in the addition process. The low-order 9 bits of the accumulator content are then returned to the B register.

CP 0 Issue. Transmit B to addend register. Enter a -1 in the accumulator.

CP 1 Transmit accumulator data to adder. Add mode. The addend register is free in this CP.

CP 2 Transmit adder result to accumulator.

CP 3 Transmit a copy of the accumulator data to B register.

060 Instruction

A = (B)

This instruction enters the content of operand register B in the accumulator. It then clears the carry flag.

CP 0 Issue. Transmit B data to RP.

CP 1 Transmit operand register data to accumulator.

061 Instruction

A = A & (B)

This instruction forms the bit-by-bit logical product of the previous accumulator content and the content of operand register B and places the result in the accumulator. It then clears the carry flag.

The logical product of the previous accumulator content and the operand register content is formed at the input to the accumulator. No additional time is required for this function.

CP 0 Issue. Transmit B data to RP.

CP 1 Transmit operand register data to accumulator.

062 Instruction

A = A + (B)

This instruction adds the content of operand register B to the previous accumulator content. The addition is performed in a twos complement mode. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process.

CP 0 Issue. Transmit B data to RP.

CP 1 Transmit operand register data to addend register.

CP 2 Transmit data to adder; add mode. The data in the accumulator and addend register is transmitted to the adder in CP 2. The addend register is free in this clock period to accept another operand.

CP 3 Transmit adder result to accumulator.

063 Instruction

A = A - (B)

This instruction subtracts the content of operand register B from the previous accumulator content. The subtraction is performed by complementing the subtrahend and adding the result to the accumulator content in a twos complement mode. A one is then added to the result. The instruction complements the carry flag if a carry is propagated from the accumulator during either addition process.

CP 0 Issue. Transmit B data to RP.

CP 1 Transmit operand register data to addend register.

CP 2 Transmit data to adder; subtract mode. The data in the accumulator and addend register is transmitted to the adder in CP 2. The addend register is free in this clock period to accept another operand.

CP 3 Transmit adder result to accumulator.

064 Instruction

(B) = A

This instruction stores the accumulator content in operand register B.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit B data to RP and DP.

CP 1 Transmit accumulator data to operand register. The function of CP 1 is delayed if the data is not available in the accumulator during the indicated clock period. In this case, the pointer in RP and DP is held until the accumulator data is available.

065 Instruction

(B) = A + (B)

This instruction adds the content of operand register B to the previous accumulator content. The addition is performed in a twos complement mode. The instruction complements the carry flag if a carry is propagated from the accumulator in the addition process. It then replaces the content of operand register B with the new accumulator content.

This instruction cannot issue if the DP register contains a pointer from a previous instruction.

CP 0 Issue. Transmit B data to RP and DP.

CP 1 Transmit operand register data to addend register. The RP pointer is discarded in CP 1 and reentered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode the function at CP 2 is delayed if the data is not available in the accumulator. The addend register is free in this CP.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit a copy of the accumulator data to operand register.

066 Instruction

(B) = (B) + 1

This instruction replaces the content of operand register B with its previous content increased by 1. The result is left in the accumulator as well as in the operand register. The carry flag is cleared at the beginning of this operation. One is entered in the accumulator. The content of the operand register entered in the addend register and is then added to the accumulator content in a twos complement mode. The carry flag is complemented if a carry is propagated from the accumulator in the addition process. The result is then returned to the operand register.

This instruction cannot issue if the DP register contains a pointer from a previous intruction.

CP 0 Issue. Transmit B data to RP and DP.

CP 1 Transmit operand register data to addend register. Enter a plus one in the accumulator.

The RP pointer is discarded in CP 1 and re-entered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode. The addend register is free in this CP.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit copy of accumulator data to operand register.

067 Instruction

(B) = (B) - 1

This instruction replaces the content of operand register B with its previous content decreased by 1. The result is left in the accumulator as well as in the operand register. The carry flag is cleared at the beginning of this operation. The accumulator bits are forced set. The content of the operand register is entered in the addend register and then added to the accumulator content. The carry flag is complemented if a carry is propagated from the accumulator in the addition process. The result is then returned to the operand register.

This instruction cannot issue if the DP register contains a pointer from a previous intruction.

CP 0 Issue. Transmit B data to RP and DP.

CP 1 Transmit operand register data to addend register. Enter an all ones value in the accumulator.

The RP pointer is discarded in CP 1 and re-entered with the same pointer from DP in CP 3.

CP 2 Transmit data to adder. Add mode. The addend register is free in this CP.

CP 3 Transmit adder result to accumulator. Transmit DP data to RP.

CP 4 Transmit copy of accumulator data to operand register.

070 Instruction

P = P + d

This instruction terminates the current program sequence and begins a new sequence. The initial address for the new sequence is obtained by adding the d designator to the address of the current instruction. The d designator is treated as a 9-bit positive integer and is added in a 16-bit twos complement mode. The accumulator content and carry flag are not altered in this process.

The issue of further instruction is blocked until the branch mode is resolved.

CP 0 Issue. Transmit P data to accumulator. Transmit d data to addend register.

CP 1 Transmit accumulator data to adder. Add mode.

CP 2 Transmit adder result to P register. The first instructon in the new program sequence is transmitted from the instruction stack to the II register in CP 2 if the branch is within the range of the stack. That intruction may issue at CP 4.

CP 3 If the branch is out of stack, a fetch request at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

071 Instruction

P = P - d

This instruction terminates the current program sequence and begins a new sequence. The initial address for the new sequence is obtained by subtracting the d designator to the address of the current instruction. The d designator is treated as a 9-bit positive integer and is subtracted in a 16-bit twos complement mode. The accumulator content and carry flag are not altered in this process.

The issue of further instruction is blocked until the branch mode is resolved.

CP 0 Issue. Transmit P data to accumulator. Transmit d to addend register.

CP 1 Transmit accumulator data to adder. Subtract mode.

CP 2 Transmit adder result to P register. The first instruction in the new program sequence is transmitted from the instruction stack to the II register in CP 2 if the branch is within the range of the stack. This intruction may issue at CP 4.

CP 3 If the branch is out of stack, a fetch request at the new P address will be generated. This action may be delayed m CPs due to previous internal fetch requests.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

072 Instruction

R = P + d

This instruction suspends execution of the current program sequence and calls a subprogram for execution by the following actions. It advances the value of E by 1 and stores the address of the next sequential instruction of this program sequence in the program exit stack. Then, it begins executing a new program sequence. The initial address for the new sequence is obtained by adding the d designator to the address of the current instruction. The d designator is treated as a 9-bit positive integer and is added in a 16-bit twos complement mode. The accumulator content and carry flag are not altered in ths process.

CP 0 Issue. Transmit P data to accumulator. Transmit d to addend register.

CP 1 Transmit accumulator data to adder. Add mode. Advance E by 1.

CP 2 Transmit P+1 data to program exit stack. Transmit adder result to P register.

CP 3 If the branch is out of stack, a fetch reqest at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

073 Instruction

R = P - d

This instruction suspends execution of the current program sequence and calls a subprogram for execution by the following actions. It advances the value of E by 1 and stores the address of the next sequential instruction of this program sequence in the program exit stack. Then, it begins executing a new program sequence. The initial address for the new sequence is obtained by subtracting the d designator from the address of the current instruction. The d designator is treated as a 9-bit positive integer and is subtracted in a 16-bit twos complement mode. The accumulator content and carry flag are not altered in ths process.

CP 0 Issue. Transmit P data to accumulator. Transmit d to addend register.

CP 1 Transmit accumulator data to adder. Subtract mode. Advance E by 1.

CP 2 Transmit P+1 data to program exit stack. Transmit adder result to P register.

CP 3 If the branch is out of stack, a fetch request at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

074 Instruction

P = dd

This instruction terminates the current program sequence and begins a new sequence. The initial address for the new sequence is obtained from operand register d.

The program fetch request flag is set if the content of operand register d is 0. The issue of further instructions is blocked until the first instruction of the new sequence is in the II register.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator. Enter 0 in the addend register.

CP 2 Transmit accumulator data to adder. Add mode.

CP 3 Transmit adder result to P register. A fetch request at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

075 Instruction

P = dd + k

This instruction terminates the current program sequence and begins a new sequence. The initial address for the new sequence is obtained by adding the content of operand register d to the next parcel of the current program sequence. The addition is performed in a 16-bit twos complement mode. The content of the accumulator and carry flag are not altered in this process.

The program fetch request flag is set if the content of operand register d is 0. The issue of further instructions is blocked until the first instruction of the new program sequence is not in the II register in CP 1.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator. Transmit k data to addend register.

CP 2 Transmit accumulator data to adder. Add mode.

CP 3 Transmit adder result to P register. A fetch request at the new P address will be generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

076 Instruction

R = dd

This instruction suspends execution of the current program sequence and calls a subprogram for execution by the following actions. It advances the value of E by 1 and stores the address of the next sequential instruction of this sequence in the program exit stack. It then begins executing a new program sequence. The initial address for the new sequence is obtained from operand register d.

The exit stack boundary flag is set if the advanced E value is 14. The program fetch request flag is set if the content of operand register d is 0. The issue of further instructions is blocked until the first instruction of the new program sequence is in the II register.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator. Enter a 0 value in addend register.

CP 2 Transmit accumulator data to adder. Add mode. Advance E by 1.

CP 3 Transmit P+1 data to exit stack. Transmit adder result to P register. A fetch request at the new P address is generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new intruction.

077 Instruction

R = dd + k

This instruction suspends execution of the current program sequence and calls a subprogram for execution by the following actions. It advances the value of E by 1 and stores the address two greater than the address of the current instruction in the program exit stack. Then it begins a new program sequence. The initial address for the new sequence is obtained from operand register d to the next parcel of the current program sequence. The addition is performed in a 16-bit twos complement mode. The content of the accumulator and carry flag are not altered in this process.

The exit stack boundary flag is set if the advanced E value is 14. The program fetch request flag is set if the content of operand register d is 0. The issue of further instructions is blocked until the first instruction of the new program sequence is in the II register.

CP 0 Issue. Transmit d data to RP.

CP 1 Transmit operand register data to accumulator. Transmit k data to addend register.

The entry of the k data in the addend register is delayed if the next parcel of the program sequence is not in the II register in CP 1.

CP 2 Transmit accumulator data to adder. Add mode. Advance E by 1.

CP 3 Transmit P+1 data to exit stack. Transmit adder result to P register. A fetch requst at the new P address is generated. This action may be delayed m CPs due to a previous internal fetch request.

CP m+1 Memory request generated by the fetch. A delay of n CPs is possible due to memory conflicts.

CP n+1 Acceptance signal from memory.

CP n+2 Transmit memory data to II register.

CP n+3 II data available for decode.

CP n+4 Issue new instruction.

100-137 Conditional Branch Instructions

Instructions 100 through 137 are branch instructions that jump to a new program sequence only if a branch condition is met. There are eight branch modes, which are represented in the unconditional form by instructions 070 through 077. All possibilities of these eight modes are combined with four branch criteria to form the set of instructions 100 through 137. The branch criteria are as follows:

------, C = 0
------, C = 1
------, A = 0
------, A # 0

The first of these branch conditions, C = 0, causes the branch to be taken if the carry flag is 0. If the carry flag is set, the current program sequence is continued.

The second branch condition, C = 1, is the complement of the first. The branch is taken if the carry flag is set. The current sequence is continued if the carry flag is 0.

The third branch condition, A = 0, causes the branch to be taken if the accumulator content is 0. If the accumulator content is nonzero, the current sequence is continued.

The final branch condition, A # 0, is the complement of the third. The branch is taken if the accumulator content is nonzero. The current sequence is continued if the accumulator content is 0.

The timing of the above sequences is the same as the timing of the corresponding instruction in the unconditional mode, if the branch is taken. Formation of the branch condition requires 1 CP after the accumulator has received the desired data. The issue of the next instruction is delayed until the branch criterion is available. If the branch criterios is available in CP 0 and the branch is not taken based on that criterion, the next instruction in the current program sequence may issue in the next clock period.

140-177 I/O Channel Instructions

Instructions 140 through 177 allow I/O processor control of the I/O channel activity. The d designator in instructions 140 through 157 specifies which I/O channel is addressed. In the 160 through 177 instructions, the content of the B register specifies the channel. The low-order 4 bits of the function code for the instruction are sent to the channel interface control along with a go function signal. This 4-bit code is then interpreted by the channel interface control circuits in a manner unique to that channel.

This series of instructions may provide the accumulator data to the interface and the data coming back from the interface may replace the present accumulator contents. Instructions 150 through 153 and 170 through 173 read a 16-bit quantity into the accumulator. This quantity is whatever value the channel interface provides as a result of its interpretation of the channel function. This data transfer is defined in the description for each individual channel. Instructions 140 through 177 may transfer data from the accumulator to the channel interface.

None of the I/O channel instructions involves any significant delay in the execution of the program sequence. There is no mechanism for the I/O channel control to delay execution of further instructions as a result of interpreting the 4-bit code. Delays in executing program functions must be programmed through sampling of the channel busy and done flags or through the equivalent use of the interrupt mechanism.

After issuing any comment to the I/O channels, allow 1 CP (by issuing a pass instruction, or other instruction) before checking the channel busy or done flags. One CP should also be allowed after any : 6 or : 7 I/O instruction (modifying channel interrupt flag) before checking for the channel interrupt number (IOR : 10).

INTERFACES 7

INTRODUCTION

Interfaces are required to adapt the I/O Processor to peripheral devices to take advantage of its capabilities. The main purposes of an interface are buffering data, generating control signals for the peripheral device, and possibly multiplexing several devices into the same I/O Processor channel. This section describes the characteristics of the interfaces, gives a table of currently used functions for the interfaces, and describes operational characteristics of the interfaces.

INTERFACE CHARACTERISTICS

The I/O Processor provides for 40 I/O channels. These channels are addressed by the d designator in the program instruction or by the B register contents. Data may be transferred from the I/O Processor accumulator to an interface register or from an interface register to the accumulator. I/O Memory ports may be used for block transfers of data into or out of I/O Memory. Data transfers and channel interface actions are a function of each interface logic control.

Each interface may interpret up to 16 function signals from the I/O Processor program. These functions are generated by instructions 140 through 177. Interpretation of each function is specifically designated by each interface. However, three of the function codes are fairly common among interfaces and are described below.

iod : 0 or IOB : 0.

This function clears the channel busy and done flags and places the channel in an idle status.

iod : 6 or IOB : 6.

This function clears the channel interrupt flag for the associated channel, which blocks any further interrupt requests from that channel.

iod : 7 or IOB : 7.

This function sets the channel interrupt enable flag for the associated channel and enables the interrupt requests from that channel.

Each channel interface provides for a busy flag. This flag is normally set during the active period of the channel and cleared during an idle period. The setting and clearing of this flag depends on the channel interface interpretation of the 16 function codes. The channel busy flag may be sensed by the I/O Processor program through execution of the 041 and 043 instructions.

Each channel interface provides for a done flag. This flag is normally used to signal the I/O Processor program when some step of the channel activity has reached a point where program action is required. Setting and clearing of the flag is normally a function of the interface hardware, but the program may set or clear the flags for special purposes. The program may sense the state of this flag through the 040 and 042 instructions. An interrupt is normally generated by the interface hardware when the channel done flag is set and the channel interrupt enable flag is also set. The system must have interrupts enabled to process the interrupt.

INTERFACE FUNCTION CODES

Table 7-1 lists all the currently supported peripheral devices and briefly explains each function code interpretation that has been implemented. The mnemonic shown is for A Programming Machine Language, APML.

Table 7-1. Interface functions

Device	Mnemonic	Function
DISK STORAGE UNIT (DKA-DKP)	DKA : 0	Clear the channel control
	DKA : 1	Select mode or request status
	DKA : 2	Read data into I/O Memory
	DKA : 3	Write data from I/O Memory
	DKA : 4	Select a new head group
	DKA : 5	Select a new cylinder
	DKA : 6	Clear the channel interrupt enable flag
	DKA : 7	Set the channel interrupt enable flag
	DKA : 10	Read I/O Memory current address
	DKA : 11	Read status response
	DKA : 14	Enter I/O Memory beginning address
	DKA : 15	Status response register diagnostic
CONSOLE KEYBOARD (TIA,TIB,TIC...)	TIA : 0	Clear the channel done flag
	TIA : 6	Clear the channel interrupt enable flag
	TIA : 7	Set the channel interrupt enable flag
	TIA : 10	Read data into accumulator and clear done flag
CONSOLE DISPLAY (TOA,TOB,TOC...)	TOA : 0	Clear the channel busy and done flags
	TOA : 6	Clear the channel interrupt enable flag
	TOA : 7	Set the channel interrupt enable flag
	TOA : 14	Send accumulator data to display
EXPANDER CHASSIS	EXB : 0	Idle the channel
	EXB : 1	Data input from A register (DIA)
	EXB : 2	Data input from B register (DIB)
	EXB : 3	Data input from C register (DIC)
	EXB : 4	Read busy/done flag, interrupt number
	EXB : 5	Load device address
	EXB : 6	Send interface mask (MSKO)
	EXB : 7	Set interrupt mode
	EXB : 10	Read data bus status
	EXB : 11	Read status 1
	EXB : 13	Read status 2
	EXB : 14	Data output to A register (DOA)
	EXB : 15	Data output to B register (DOB)
	EXB : 16	Data output to C register (DOC)
	EXB : 17	Send control

Table 7-1. Interface functions (continued)

Device	Mnemonic	Function
INPUT FROM CPU I/O CHANNEL (CIA,CIB,CIC...)	CIA : 0	Clear channel
	CIA : 1	Enter I/O Memory address, start input
	CIA : 2	Enter parcel count
	CIA : 3	Clear channel parity error flags
	CIA : 4	Clear ready waiting flag
	CIA : 6	Clear interrupt enable flag
	CIA : 7	Set interrupt enable flag
	CIA : 10	Read I/O Memory address
	CIA : 11	Read status (ready waiting, parity errors)
OUTPUT TO CPU I/O CHANNEL (COA,COB,COC...)	COA : 0	Clear channel
	COA : 1	Enter I/O Memory address
	COA : 2	Enter parcel count
	COA : 3	Clear error flag
	COA : 4	Set/clear external control signals
	COA : 6	Clear interrupt enable flag
	COA : 7	Set interrupt enable flag
	COA : 10	Read I/O Memory address
	COA : 11	Read status (4-bit channel data, error)
INPUT FROM CPU MEMORY CHANNEL (HIA,HIB,HIC...)	HIA : 0	Clear channel busy, done flags
	HIA : 1	Enter I/O Memory address
	HIA : 2	Enter upper Central Memory address
	HIA : 3	Enter lower Central Memory address
	HIA : 4	Read Central Memory, enter block length
	HIA : 6	Clear interrupt enable flag
	HIA : 7	Set interrupt enable flag
	HIA : 14	Enter diagnostic mode
OUTPUT TO CPU MEMORY CHANNEL (HOA,HOB,HOC...)	HOA : 0	Clear channel busy, done flags
	HOA : 1	Enter I/O Memory address
	HOA : 2	Enter upper Central Memory address
	HOA : 3	Enter lower Central Memory address
	HOA : 5	Write Central Memory, enter block length
	HOA : 6	Clear interrupt enable flag
	HOA : 7	Set interrupt enable flag
	HOA : 14	Enter diagnostic mode

Table 7-1. Interface functions (continued)

Device	Mnemonic	Function
ERROR LOGGING CHANNEL (ERA,ERB,ERC...)	ERA : 0	Idle channel
	ERA : 6	Clear interrupt enable flag
	ERA : 7	Set interrupt enable flag
	ERA : 10	Read error status
	ERA : 11	Read error information (first parameter)
	ERA : 12	Read error information (second parameter)
	ERA : 13	Read error information (third parameter)
BLOCK MULTIPLEXER CHANNEL (BMA,BMB,BMC...)	BMA : 0	Clear channel control
	BMA : 1	Send reset functions
	BMA : 2	Channel command
	BMA : 3	Read request-in address
	BMA : 4	Asynchronous I/O
	BMA : 5	Delay counter diagnostic
	BMA : 6	Clear channel interrupt enable flag
	BMA : 7	Set channel interrupt enable flag
	BMA : 10	Read I/O Memory address
	BMA : 11	Read byte count
	BMA : 12	Read status
	BMA : 13	Read input tags
	BMA : 14	Enter I/O Memory address
	BMA : 15	Enter byte count
	BMA : 16	Enter device address
	BMA : 17	Enter output tags

DISK STORAGE UNIT CHANNEL

Each Buffer I/O Processor and Disk I/O Processor may have up to 16 channels connected to DD-29 Disk Storage Units. Each unit may operate independently of the other units and all may be transferring data at the same time. APML mnemonics DKA through DKP indicate these channels. The function requests for the first channel are summarized below.

DKA : 0§	Clear the channel control
DKA : 1§	Select mode or request status
DKA : 2§	Read data into I/O Memory
DKA : 3§	Write data from I/O Memory
DKA : 4	Select a new head group
DKA : 5	Select a new cylinder§
DKA : 6§§	Clear the channel interrupt enable flag
DKA : 7§§	Set the channel interrupt enable flag
DKA : 10	Read I/O Memory current address
DKA : 11	Read status response
DKA : 14	Enter I/O Memory beginning address
DKA : 15	Status response register diagnostic

DISK STORAGE UNIT CHARACTERISTICS

The DD-29 Disk Storage Unit consists of 40 rotating disk surfaces with a read/record head on each surface. The period of disk rotation is 16.6 milliseconds. The heads are moved simultaneously to one of 823 disk cylinders by means of a servomechanism. Positioning time from one cylinder to another varies from 15 milliseconds to 80 milliseconds depending on the distance the head assembly must travel.

---

§ Valid channel busy and channel done flags cannot be read until 1 CP after this function is issued.

§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

Within each disk cylinder the 40 read/record heads are divided into ten groups. Each head group then reads or records 4 bits of data in parallel. The selection of a new head group requires 6 microseconds. The recording surface available to each head group is called a disk track. This is the basic storage unit reserved by the operating system. A flaw on the disk surface requires that a track be removed from the available resources in the track reservation table for the system.

Within each disk track are 18 sectors in which data may be recorded and read back. The data in one sector is called a data block and consists of 2048 parcels of I/O Processor data plus verification and error correction data. Data may be transferred between the I/O Memory and the disk surface only in blocks of this fixed size.

DSU DATA SEQUENCE PATTERN

The data recorded in a sector of a disk track consists of a number of parts as shown in figure 7-1. The numbers below each segment in the figure are the total bits of all four heads, for the segment.

	GAP	PREAMBLE	SYNC	ID	DELAY	PREAMBLE	SYNC	512 WORDS	CRC	POSTAMBLE	
INDEX	720	912	8	24	304	912	8	32,768	128	24	BITS

Figure 7-1. DSU data sequence pattern

The total number of bits in the above figure is 35,808. This is the portion of a disk track assigned to a sector. An additional gap after the last sector has 576 bits. The total number of bits in a disk track is 645,120.

The bit positions assigned to the angular locations on the disk surface are determined by an index mark and a servo clock. The index mark is a unique mechanical mark on the rotating mechanism which provides a pulse once per disk revolution. This pulse clears a counter which then counts servo clock pulses to define the remainder of the disk timing. Servo pulses are also derived mechanically from the rotating mechanism. These pulses occur every 12 bit positions. The clock used for recording data on the disk surface is obtained by a frequency multiplier. The index mark begins the data sequence pattern listed above for sector 0. The beginning location for the other sectors is determined by the servo counter. These begin every 746 servo pulses or 8952 bit positions.

The data sequence pattern for a sector, as listed above, is recorded in two separate processes. The sector identification (ID) word, which appears as 6 bits under each recording head, is recorded on each new set of disk surfaces and is not modified in normal use of the disk storage unit. The data block and cyclic checkword are recorded with each disk write function in a normal operation. Associated with each of these recordings is a preamble, sync, and postamble which are a necessary part of the recording and reading process.

The write heads are turned on for a normal disk block write function during the intersector gap. The writing begins with the second group of preamble and sync bits which are sequenced by the disk control circuits. The data from the I/O Memory is then recorded in a block of 32,768 bits. This is followed by a 128-bit cyclic redundancy checkword which was generated from the data by a Fire code generator. The write heads are turned off after the 24-bit postamble.

SECTOR IDENTIFICATION WORD

The sector identification (ID) word for each sector is 24 bits. It is composed of a cylinder number, a head-group number, a sector number and four parity bits. The ID format is shown in figure 7-2.

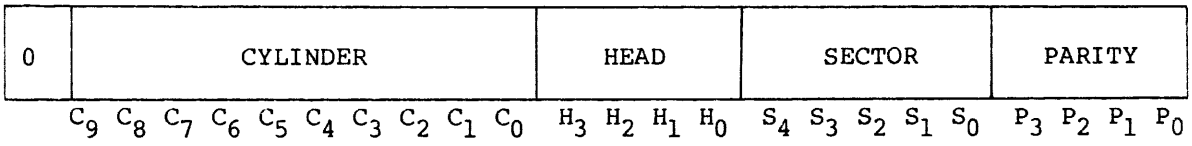


Figure 7-2. Sector ID format

The parity bits in the sector identification word each protect a cross-section of the other 20 bits of the ID. The assignment of parity bits to groups of ID bits is shown in table 7-2.

I/O MEMORY ADDRESS REGISTER

The I/O Memory address register is both an interface input register and an interface output register for the channel. The beginning address for a block of disk data is entered in the register by the processor before issuing a read or write function. This address is restricted to values which are a multiple of four because of the burst mode used in moving data into and out of the I/O Memory. The low-order 2 bits of the address are forced to 0 in the register. If the processor enters bit values other than 0 in these positions, these values are discarded.

Table 7-2. Sector ID parity bit assignments

Parity Bits	$p^0/2^0$	$p^1/2^1$	$p^2/2^2$	$p^3/2^3$
Data Bits	$C^7/2^{20}$ $C^3/2^{16}$ $H^3/2^{12}$ $S^4/2^8$ $S^0/2^4$	$C^8/2^{21}$ $C^4/2^{17}$ $C^0/2^{13}$ $H^0/2^9$ $S^1/2^5$	$C^2/2^{22}$ $C^5/2^{18}$ $C^1/2^{14}$ $H^1/2^{10}$ $S^2/2^{26}$	$0/2^{23}$ $C^6/2^{19}$ $C^2/2^{15}$ $H^2/2^{11}$ $S^3/2^{27}$

C = Cylinder, H = Head, S = Sector, P = Parity

The address in the I/O Memory address register is increased by a count of four as each burst of four words is transferred to or from the I/O Memory. This address may be monitored by the I/O Processor using the DKA : 10 function. The address may be used without a new entry from the processor after completion of a data transfer. In this case the beginning address for the next block of disk data will be the next sequential storage address. For a disk write the I/O Memory address will be left pointing four parcels beyond the last address of the buffer.

STATUS RESPONSE REGISTER

The status response register is used for the specific response requested by a function 1 request and also for the implied response of function 5. Details of these functions are listed under the appropriate headings.

DKA : 0 - CLEAR CHANNEL

Clear the channel busy and channel done flags. No parameters are required for this function. This function is not interlocked with any disk sequence which may be in process.

DKA : 1 - SELECT MODE

This function request allows the processor to select a mode for the disk storage unit or to request status information from the interface. The content of the accumulator at the time of the function request is used as a selecting parameter. The categories of parameter values are summarized in table 7-3.

Table 7-3. DKA : 1 parameters

Value	Meaning
000xxx	Release Unit
001xxx	Reserve Unit
002xxx	Clear fault flags
003xxx	Return to zero cylinder
004xxx	Select margin conditions
005xxx	Read sector number
006xxx	Read error flags
007xxx	Read disk status

Parameter 000xxx - Release Unit

This function request sets the channel busy flag and clears the channel done flag. No other flag request can be made on a particular DSU until a DKA : 1 request has been issued for that unit. After a few microseconds, the interface clears the channel busy flag and sets the channel done flag. The disk storage unit is released from the reservation on that port and is free for reservation on the other port.

Parameter 001xxx - Reserve Unit

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later, the interface clears the channel busy flag and sets the channel done flag. The DSU is reserved for the requesting I/O Processor if it is not currently reserved by another device using the DSU access port. The function does not automatically return a status response. The I/O Processor must issue a separate function 1 status request to determine whether the reservation was accepted. The reserve unit function automatically selects head group 0. The unit must be reserved before it will recognize a read function (DKA : 2), a write function (DKA : 3), or a select cylinder function (DKA : 5). All other functions may be issued to an unreserved unit, unless the DSU is reserved on another access port.

Parameter 002xxx - Clear Fault Flags

This function 1 request with an 002xxx parameter sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. The fault conditions stored in fault registers in the interface and in the DSU are all cleared.

Parameter 003xxx - Return to Zero Cylinder

This function request sets the channel busy flag and clears the channel done flag. The read/write heads are positioned to cylinder 000 as if the DSU were just powered up. The time for positioning the heads depends upon the distance to be traveled, and may extend to 500 milliseconds for an 822-cylinder move. When the positioning is completed, the interface clears the channel busy flag and sets the channel done flag.

Parameter 004xxx - Select Cylinder Margin

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the disk interface clears the channel busy flag and sets the channel done flag. The read/write heads are moved slightly away from the normal cylinder center to attempt reading data which cannot be recovered using normal head positioning. The amount to be offset is determined by the low-order 5 bits in the function parameter. Each unit of the 5-bit value offsets the heads 25 microinches (0.64 micrometers). Bit 2^5 of the parameter defines the direction of the offset: a 1 indicates an offset toward the center of the disk and a 0 specifies a move away from the center of the disk. The nominal cylinder width is 2200 microinches (5600 micrometers), and a nominal center-to-center cylinder spacing spacing is 2600 microinches (6600 micrometers). The offset position is maintained until the next positioning function is received, either another margin select, or a new cylinder select. A following cylinder select automatically cancels the margin select and centers the heads over the new cylinder. The direction and amount of offset active is contained in the DSU offset register, and can be inspected by a DKA : 1, parameter 007002 command.

Parameter 005xxx - Read Sector Number

This function request sets the channel busy flag and clears the channel done flag. About 10 I/O Processor CPs later the interface clears the channel busy flag and sets the channel done flag. At this time the sector number of the sector currently under the read/write heads is loaded into the status response register in the interface. A following function 11 to the interface will read the status response register to the I/O Processor accumulator.

The sector number is not read from the disk. Instead, an interface counter operates from the DSU servo clock, tracking the sector number. The counter is updated at the start of the 180-frame intersector gap, and it is cleared by the index mark.

Parameter 006xxx - Read Error Flags

Issuing this function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the status response register on the interface receives data from the fault registers in the DSU and in the interface. This data remains in the status response register until it is replaced by data from another function request. Figure 7-3 shows the bit assignments for the error flags, and table 7-4 explains each error condition.

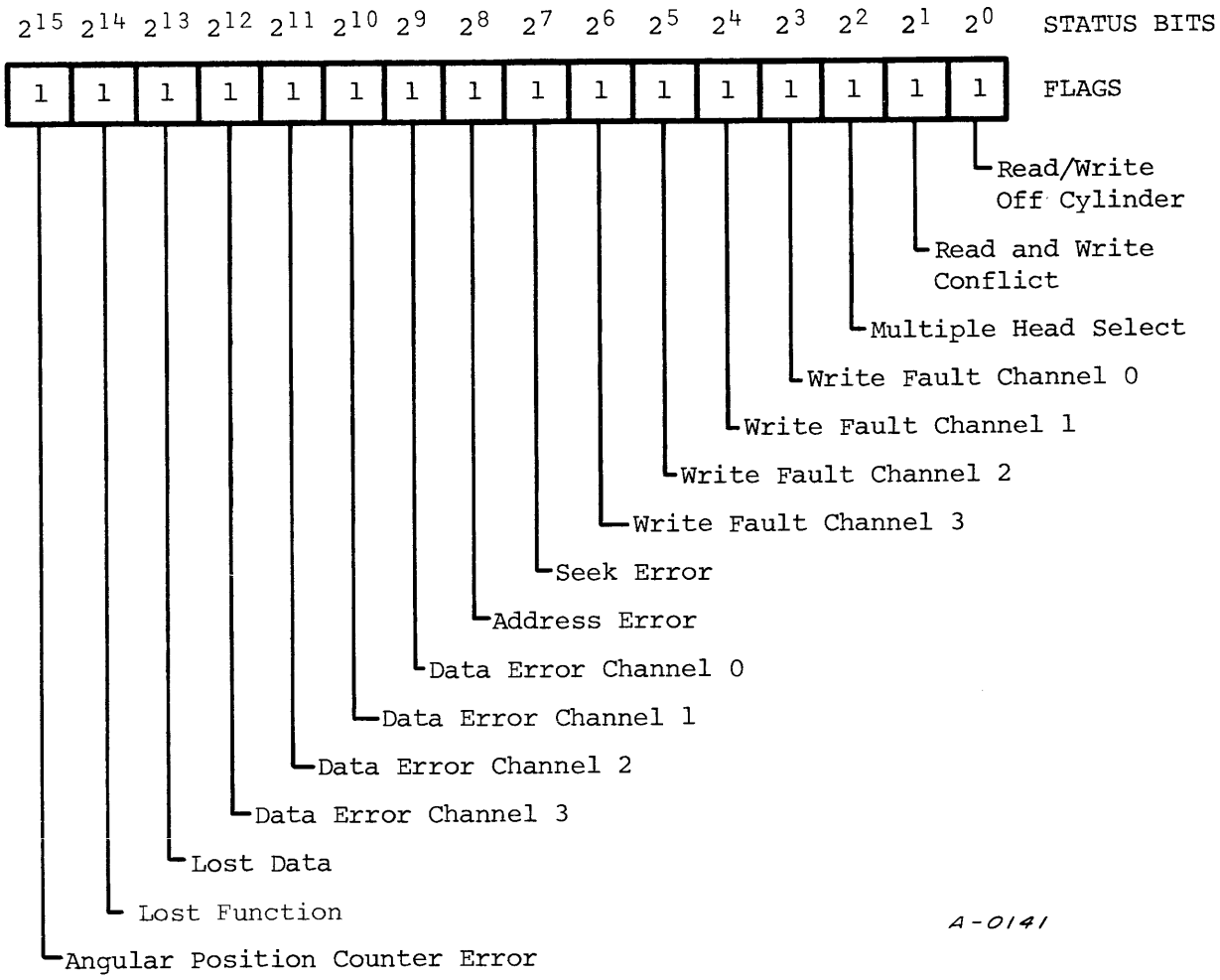


Figure 7-3. Status response error flags

Table 7-4. Parameter 006 error flags

Bit	Name	Meaning
2^0	Read/write off cylinder	An attempt was made to read or write data when the read/write heads were still in motion on a change of cylinder.
2^1	Read and write conflict	An attempt was made to strobe data simultaneously early and late, or an attempt was made to write data with the cylinder margin offset, or the unit attempted to read and write at the same time.
2^2	Multiple head select	More than 4 heads were selected simultaneously.
2^3	Write fault channel 0	A failure occurred associated with the recording head for channel 0.
2^4	Write fault channel 1	A failure occurred associated with the recording head for channel 1.
2^5	Write fault channel 2	A failure occurred associated with the recording head for channel 2.
2^6	Write fault channel 3	A failure occurred associated with the recording head for channel 3.
2^7	Seek error	A failure occurred in moving the read/write heads to a new cylinder number.
2^8	Address error	The disk storage unit received a head group select for a group number greater than 11_8, or a cylinder select for a cylinder number greater than 1466 octal, or a margin selection when the disk was not on cylinder, or a cylinder select when the disk was not on cylinder.
2^9	Data error channel 0	An error occurred in the channel 0 data during the last read operation.
2^{10}	Data error channel 1	An error occurred in the channel 1 data during the last read operation.
2^{11}	Data error channel 2	An error occurred in the channel 2 data during the last read operaton.

Table 7-4. Parameter 006 error flags (continued)

Bit	Name	Meaning
2^{12}	Data error channel 3	An error occurred in the channel 3 data during the last read operation.
2^{13}	Lost data	The data transfer between the I/O Memory and the deskewing buffers did not keep up with the disk read/write transfer in a read or write operation.
2^{14}	Lost function	A function was received before a previous function completed.
2^{15}	Angular position counter error	The index mark from the disk storage unit was received in the middle of a sector.

Parameter 007000 - Read Cylinder Register

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the interface status register receives from the DSU the currently selected cylinder number. The cylinder number occupies the 10 low-order bits of the status register. This data remains in the status register until replaced by another function request.

Parameter 007001 - Read Head Register

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the interface status register receives from the DSU the currently selected head group number, if the DSU is reserved to the processor. The head number goes in the 4 low-order register bits, and bit 2^5 is a 1 to show reservation to the requesting processor. Bit 2^6 indicates the DSU capacity is 600 Mbytes. The register value has a range of 40_8-51_8 for the 10 head groups. A zero word returned indicates the DSU is not reserved to the requesting processor. Status data remains in the register until another function request.

Parameter 007002 - Read Margin/Difference Register

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the interface status response register receives from the DSU either the currently selected offset margin or the difference between the present position of the heads and the final cylinder position during a seek. If the last function request was an offset margin selection, the status is the ones complement of the offset number selected by that function. The 5 low-order bits on the status register hold the offset number, and bit 2^5 shows the offset direction. Bit 2^5 is set to 1 if the offset is toward the center of the disk. See figure 7-4.

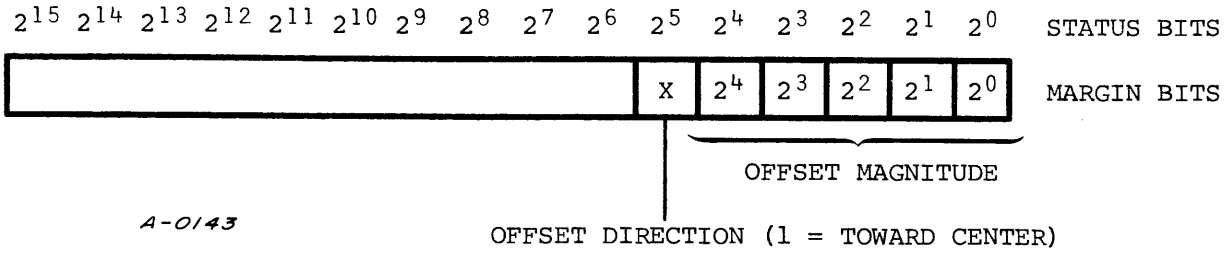
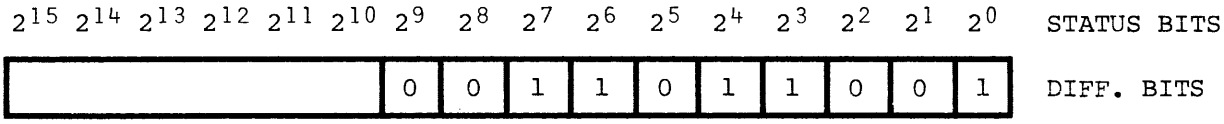


Figure 7-4. Offset margin status word

If the previous function request was a select cylinder, the interface status register receives the ones complement of the number of cylinder positions yet to be crossed before reaching the desired cylinder. The register contains 1777_8 when the heads are positioned at the desired cylinder. The difference number goes into the least significant 10 bits of the status register. See figure 7-5.



Example: 294 = 0100100110 tracks to go
1011011001 one's complement to register

Figure 7-5. Difference register example

Parameter 007003 - Read Interlock Register

This function request sets the channel busy flag and clears the channel done flag. A few microseconds later the interface clears the channel busy flag and sets the channel done flag. At this time the interface status response register receives from the DSU the contents of the interlock register. Eight interlock flags are placed in the low-order bit positions of the status response register. The interlock flags are shown in figure 7-6. Ones indicate fault conditions. Table 7-5 explains each flag.

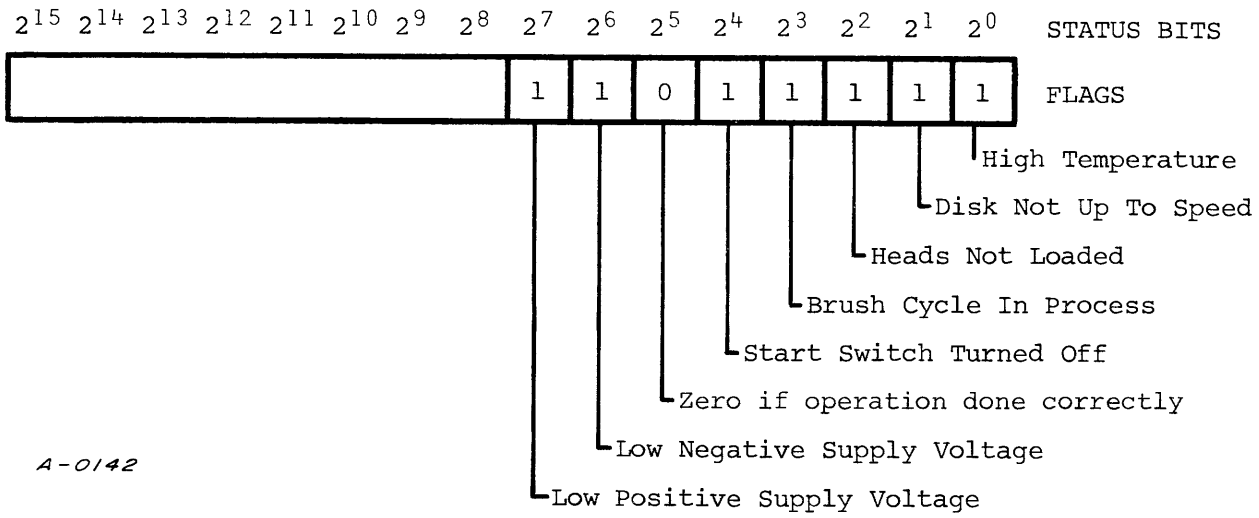


Figure 7-6. Interlock register status bits

Timing Notes

A function 4 select head group request can be followed immediately by another function to be done as soon as the new head group is active. For example, a read disk data function may be stacked behind the select head group function. If a function 1 status request is sent to the interface before the read disk data function begins, the function 1 command takes the place of the read disk data function, and the read disk data function is lost. In this case, the Lost Function error flag sets in the interface and will be available for the Read Error flags status request.

It is possible to issue a function 1 status request after a select cylinder function and before the select cylinder function has finished. But the status request should be issued immediately after the select cylinder function, to give plenty of time separation between the done flag of the status request and the done flag of the cylinder select. If the two done flags occur close together, the program may not be able to distinguish between them and handle the conditions incorrectly. It is best to avoid status request functions near the end of a head positioning sequence.

Table 7-5. Interlock status bits

Bit	Meaning
2^0	This bit is set to indicate that the disk drive cabinet is over the normal temperature range.
2^1	This bit is set to indicate that the disk surfaces are not up to speed.
2^2	This bit is set to indicate that the disk heads are not loaded on the disk surface.
2^3	The bit is set to indicate that the disk drive brush cycle is in process.
2^4	This bit is set to indicate that the disk drive start switch is turned off.
2^5	This bit is always zero.
2^6	This bit is set to indicate that the negative voltage supply for the disk drive is below normal voltage.
2^7	This bit is set to indicate that the positive voltage supply for the disk drive is below normal voltage.

DKA : 2 - READ DISK DATA

A function 2 request to the disk channel interface begins the process of reading a block of disk data into the I/O Memory at the beginning address specified by the interface A register. The disk sector number is specified by the accumulator content at the time of the function request. The head group number and cylinder number are the values which were last selected by the appropriate functions for that purpose.

The reading process begins by a hardware test for proper angular position of the disk surface. The sector number requested by the processor is compared with the sector number currently under the reading heads. If the disk is not in the proper position, the execution of the read function is delayed until the disk surface is properly positioned.

The interface anticipates the processor request for reading data when other function requests have been satisfied. This anticipation takes the form of reading each sector as the data appears under the reading heads with the expectation that the read request for that data will be forthcoming.

Data read from the disk surface is routed to the I/O Memory through two buffers, labeled A and B. Each buffer holds 512 words of I/O Processor data. Buffer A receives the first 512 words of disk data. Buffer B receives the second 512 words. The use of the two buffers alternates until the entire block of data has been processed.

The function 2 request clears the channel done flag and sets the channel busy flag. The sector number is captured from the accumulator and compared with the sector count. The currently anticipated reading process will be accepted if buffer A has not yet filled with the first sector of data from the disk surface. If the read process has proceeded beyond this point, or if the sector number is wrong, the current read process is aborted and the function request waits up to one disk revolution for sector coincidence.

The data in buffer A begins moving to the processor I/O Memory as the disk read circuits begin filling buffer B. The data moves to the I/O Memory in bursts of four words and normally empties buffer A before the disk read circuits have filled buffer B. The roles of the two buffers then reverse and data continues moving from the disk surface to the I/O Memory until the entire block has been processed.

The eight words of error correction data which follow the data block are read into a buffer as the last section of the data block is moving from the other buffer to the I/O Memory. Reading stops at this point for a check of the Fire code generators which have been summing the data as it was read from the disk surface. If all four generators are clear, the data read from the disk is correct. The channel done flag is set and the channel busy flag is cleared as the last word of data is entered in the I/O Memory.

DKA : 2 - Abnormal Conditions

An abnormal condition in the disk storage unit or in the reading or processing of the data is indicated to the I/O Processor by a terminating sequence which sets the channel done flag and leaves the channel busy flag set as well. The processor program can then analyze the error by appropriate function 1 requests for status information. Three types of error conditions cause this termination, as described below.

Recorded data error - While the recorded data is transferring from the DSU to the interface, the Fire code generators operate on the data as explained in "Fire Code Generator" later in this section. The 32-bit error correction code from each read head also passes through the Fire code generator for that head. The four new error correction codes generated should each be 32 zeros, if the data was stored and read correctly.

If any error correction code is non-zero, the recorded data error flag for that head is set in the fault register. There are four such flags for the four read channels. The I/O processor may use a special read mode to obtain the error correction code necessary to proceed with the correction software algorithm. The special read mode is described in the following "DKA : 2 - Special Modes" section.

Lost data error - The lost data flag is set in the fault register if the transfer of data from the buffers did not keep up with the reading of data from the disk surface. The processor must reread the sector in this case.

Lost Function - The lost function flag in the error status response is set if a function is received at the interface before a previous function has finished. The new function is lost, and should be requested again.

DKA : 2 - Special Modes

Special modes for the reading of disk data are requested by the processor through sector numbers larger than 40 octal. The low-order 5 bits of the requested sector number are translated for sector coincidence. The high-order bits are interpreted for special mode.

Format mode - Sectors 40_8 through 61_8 request that the reading process begin with the verification word and continue for a total of 1000 octal words. The following eight words are then interpreted as the error correction code. This read mode is used for maintenance only. In this case the error correction code was not generated by the Fire code generators because the data length during the write sequence was much longer than the requested length for this mode. This combination of long write and short read allows the maintenance routine to test for various failure modes in the Fire code generators.

Read correction code - Sectors 100_8 through 121_8 request that the reading process begin with the error correction code and continue for the eight words of that code. This mode is used when the data in the associated sector has been read incorrectly and the processor program wishes to do error correction.

Read early - Sectors 200_8 through 221_8 cause the reading heads to sample the data from the disk surface somewhat earlier than normal. This mode is used to recover data that cannot be read in a normal mode.

Read late - Sectors 400_8 through 421_8 cause the reading heads to sample the data from the disk surface somewhat later than normal. This mode is used to recover data that cannot be read in a normal code.

Mixed modes - The read early and read late selections described above may be used together with the other special modes by combining the high-order bit values in a logical sum.

Buffer Echo Mode

A special diagnostic feature allows writing a test data block into the A and B buffers and then reading out the test data. This can only be done after a master clear and before a DKA : 1 function. The master clear could be from a power on sequence or from a deadstart sequence. A DKA : 14 function to enter the starting address for the read to I/O Memory must precede the DMA : 2 read function. Refer to the buffer echo mode explanation given with the following DKA : 3 write function discussion.

DKA : 3 - WRITE DISK DATA

A function 3 request to the disk channel interface begins the process of writing a block of data from the I/O Memory onto the disk surface. The I/O Memory address for beginning the block of data is specified by the content of the interface A register. The disk sector number is specified by the accumulator content at the time of the function request. The head group number and cylinder number are the values which were last selected by the appropriate functions for that purpose. The function request sets the channel busy flag and clears the channel done flag.

The writing process begins by filling buffer A with data from the I/O Memory. When this buffer is full, the interface monitors the angular position of the disk surface for the proper position to begin writing data. This position is slightly past the prerecorded verification word for the requested sector. The write circuits are turned on when the disk is in the proper position and the data in buffer A is transmitted to the disk surface. At this time the interface begins filling buffer B with data from the I/O Memory. Buffer B should be filled before the disk writing circuits have emptied buffer A. The lost data flag in the fault register is set if this should not be the case. The disk writing circuits begin transmitting data from buffer B to the disk surface as soon as buffer A has been emptied. This process continues with the roles of buffers A and B alternating until the last of the data in the block has been loaded into buffer B. No further data is read from the I/O Memory and the interface waits for completion of the data transfer from buffer A to the disk surface.

The interface sets the channel done flag as soon as buffer A is emptied for the last portion of the disk data. The channel busy flag is cleared if no error has occurred in the writing process. The interface then continues transmitting data from buffer B to the disk surface. The I/O Processor is free at this point to issue another write function request and begin loading buffer A with data for another sector on this track.

The disk write circuits follow the last data from buffer B with eight words of error correction code. This data comes directly from the Fire code generators. The writing circuits are turned off at this time and the interface prepares to write data in the following sector if the I/O Processor has requested this function.

The busy flag is left set at the time the done flag is cleared if an error has occurred during the writing process. There are two possible error conditions, which are described in the following paragraphs.

Fire Code Generation

As data is transferred from the I/O Processor into the interface buffer, the interface generates the Fire code, or cyclic redundancy checkword (CRC). The Fire code generator uses the polynomial $x^{32} + x^{23} + x^{21} + x^{11} + x^{2} + 1$ to create one 32-bit checkword for each of the four DSU write heads. This polynomial allows for the correcting of data in a single error burst of 11 bits or less; and detects errors if there are two bursts, or bursts of more than 11 bits in error.

The checkword is generated by a 32-bit shifting register; the register is cleared before a data transfer is begun. The data bit going to the buffer is compared to bit 2^{31} of the shifting register. If the two bits are alike, bit 2^{0} of the register is cleared and the register is shifted left one position. If the bits are not alike, register bits 2^{1}, 2^{10}, 2^{20}, and 2^{22} are complemented, the register contents are then shifted left one position, and bit 2^{0} of the register is set to 1. Then, the next data bit is compared with register bit 2^{31} and the register contents are altered as previously.

Data bit 2^{63} is the first data bit to be used in generating the checkword for DSU head 3; 2^{62} is the first data bit used for DSU head 2, 2^{61} for head 1, and 2^{60} for head 0. Bits 2^{47}, 2^{31}, and 2^{15} are the other bits used first of their 16-bit channel words destined for DSU head 3.

The checkword forms continuously while the interface transfers data to the DSU. The 32-bit error correction code is appended to the data block.

Lost Data Error

The lost data flag is set if the data transfer from the I/O Memory has not kept up with the data transfer to the disk surface.

Lost Function Error

The status response lost function error flag is set if a function is received at the interface before the previous function is finished. The new function is lost. To recover, issue the new function again.

Format Mode

A special mode of operation is provided in the disk channel interface to prerecord the cylinder verification data on the disk surface. This mode is selected by a function 3 request with a sector number value of 40 through 61 octal. The disk control circuits translate the low-order 4 bits of the sector number in the normal way to select the sector for recording. The high-order bit causes the writing process to begin process to begin earlier than normal. This selection records a 4000_8 word block of data on the disk surface in an otherwise normal manner in such a way that the first word of this data block is properly positioned for the cylinder verification word. The remainder of the recording in this format mode is used for maintenance functions and is then erased by the recording of the normal sector data. Figure 7-7 shows the pattern written in each sector under format mode.

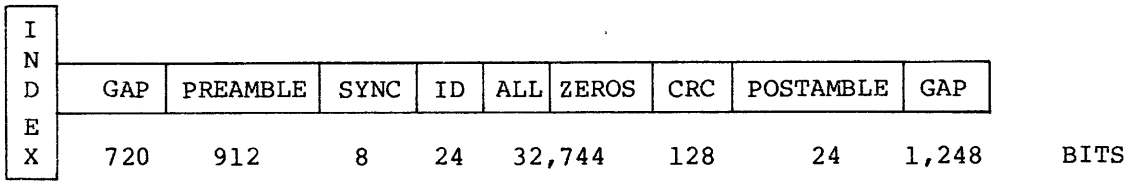


Figure 7-7. Format mode sector pattern

Buffer Echo Mode

A special diagnostic feature allows writing a test data block into buffers A and B and then reading out the test data. This can only be done after a master clear (from a power on sequence or a deadstart) and before a DKA : 1 function. The sequence is as follows:

1. Master clear
2. DKA : 14 enter I/O Memory starting address
3. DKA : 3 write data to buffers A and B
4. DKA : 14 enter I/O Memory starting address
5. DKA : 2 read data from buffers
6. Verify data
7. DKA : 1 function places controller in normal operation mode

DKA : 4 - SELECT HEAD GROUP

A function 4 request to the disk channel interface reserves the DSU and causes the head group selection circuits to select a head group. The new head group number is specified by the low-order 4 bits of the accumulator content at the time of the function request. This function may be requested at any time with respect to the execution of other disk channel functions. The 4-bit code for selection of the head group is captured in a special register in the disk channel interface. The action of switching the head group circuits requires about 5 microseconds and is delayed until the completion of any other function currently in process. It is possible, therefore, to select a new head group during a read or write sequence and continue the reading or writing from the last sector of one track to the first sector of a different track in the same disk cylinder, without missing a disk revolution.

If this function is used to reserve the DSU, any one of the DKA : 1 functions must still be issued before this DKA : 4 function.

This function request does not alter the condition of the channel busy or done flags. There is no channel interface response to this function.

DKA : 5 - SELECT CYLINDER

A function 5 request to the disk channel interface causes the disk read/record head assembly to move to a cylinder position. The cylinder number is specified by the 10 low-order bits of the accumulator content at the time of the function request.

The channel busy flag is set and the channel done flag is cleared by the function request. The servomechanism for positioning the head assembly then begins moving to the new cylinder position. This process takes from 15 milliseconds for adjacent cylinders to 80 milliseconds for maximum travel. A cylinder selection for the current cylinder requires a few microseconds for the process.

The interface monitors the cylinder positioning and when the read/record heads are on the newly requested cylinder, the data recorded on the selected track is read for verification of the cylinder. The data in the first verification word to pass under the read heads is captured and entered in the status response register. The channel busy flag is cleared to indicate completion of the requested function and the channel done flag is set. If the function cannot be completed because of an abnormal condition in the disk storage unit the channel done flag is set and the channel busy flag is left set.

It is possible to program the disk channel interface in such a way that the progress of the cylinder positioning can be monitored. This is done by aborting the normal sequence described above with a function 1 status request. This resets the busy flag and clears the done flag and begins the status response sequence. The cylinder positioning will continue but the verification process will not occur. The progress of the cylinder positioning can then be monitored by reading the difference register content from the disk storage unit. A verification can be programmed by repeating the cylinder selection.

DKA : 6 - CLEAR INTERRUPT ENABLE

A function 6 request to the disk channel interface clears the channel interrupt enable flag. This prevents interruption of the I/O Processor program and requires program monitoring of the channel done flag for proper sequencing of disk control functions.

DKA : 7 - SET INTERRUPT ENABLE

A function 7 request to the disk channel interface sets the channel interrupt enable flag. This causes an I/O interrupt request for this channel whenever the channel done flag is set. The interrupt enable flag function for the I/O Processor system.

DKA : 10 - READ LOCAL MEMORY ADDRESS

This function request reads the current value in the channel interface I/O Memory address register and enters this value in the accumulator. This function may be performed at any time with respect to a disk storage unit sequence.

DKA : 11 - READ STATUS RESPONSE

This function request reads the current content of the disk storage unit status response register and enters this value in the accumulator. This function request may be performed at any time with respect to the disk storage unit sequence. The value read will be the response from the last function which entered the status response register.

DKA : 14 - ENTER LOCAL MEMORY ADDRESS

This function enters the current accumulator content in the channel interface I/O Memory address register. The channel busy and channel done flags are not altered in this process.

DKA : 15 - STATUS RESPONSE REGISTER DIAGNOSTIC

This function can be used to verify the operation of the status response register. The function transfers a test value from the accumulator to the status response register, overwriting the current status. A DKA : 11 read status response function immediately following this diagnostic function will return the test value to the accumulator for verification. The channel busy and done flags are not affected by the DKA : 15 function.

CONSOLE DISPLAY CHANNEL

The I/O Processor has provision for a number of I/O channels connecting to operator display consoles. Each display is assigned to a separate channel and all may operate independently. Data is transmitted serially from a channel interface register to the display device.

This channel has a 7-bit interface register which receives data from the I/O Processor accumulator and transmits this data serially to the display device. Function requests for this channel 1 are described below.

TOA : 0§ Clear the channel busy and channel done flags.

TOA : 6§§ Clear the channel interrupt enable flag. The channel busy and channel done flags are not altered in this process.

TOA : 7§§ Set the channel interrupt enable flag. The channel busy and channel done flags are not altered in this process.

TOA : 14§ Enter the low-order 7 bits of the accumulator content in the channel interface register. It sets the channel busy flag and clear the channel done flag. It begins the transmission of the data from the interface register to the display device and clears the channel busy flag and set the channel done flag when the transmission has been completed.

§ Valid channel busy and channel done flags cannot be read until 1 CP after this function is issued.

§§ Allow 1 CP before checking the interrupt channel number (IOR : 10)

CONSOLE KEYBOARD CHANNEL

The I/O Processor has provision for a number of I/O channels connecting to operator console keyboards. Each keyboard is assigned to a separate channel and all may operate independently. Data is transmitted serially from the keyboard to a channel interface register. The channel busy flag is set by the channel hardware at the beginning of the transmission. The channel busy flag is cleared and the channel done flag is set when the data has been assembled in the interface register.

This channel has a 7-bit interface register which assembles the data for a character associated with a key depression. This data may then be read into the I/O Processor accumulator.

TIA : 0§ Clear the channel done flag.

TIA : 6§§ Clear the channel interrupt enable flag. The channel busy and channel done flags are not altered in this process.

TIA : 7§§ Set the channel interrupt enable flag. The channel busy and channel done flags are not altered in this process.

TIA : 10§ Read the contents of the channel interface register into the low-order 7-bit positions in the accumulator and clear the high-order bits. It sets the channel busy flag and clears the channel done flag. It transfers the data to the accumulator, then clears the channel busy flag and sets the channel done flag.

---

§ Valid channel busy and channel done flags cannot be read until 1 CP after this function is issued.

§§ Allow 1 CP before checking the interrupt channel number (IOR : 10)

PERIPHERAL EXPANDER CHANNEL

Depending on its function, an I/O Processor may have a channel connected to a peripheral expander. The peripheral expander can contain 16 controllers for peripheral devices. Only one controller can be active at one time, but that controller may be servicing more than one peripheral device. Each peripheral unit is assigned a device address and is individually selectable. The functions for the peripheral expander are listed below.

EXB : 0§	Idle channel
EXB : 1§	DIA data input from A register
EXB : 2§	DIB data input from B register
EXB : 3§	DIC data input from C register
EXB : 4§	Read busy/done, interrupt number
EXB : 5	Load device address
EXB : 6§§	MSKO mask out
EXB : 7§§	Set interrupt mode
EXB : 10	Read data bus status
EXB : 11	Read status 1
EXB : 13	Read status 2
EXB : 14§	DOA data output to A register
EXB : 15§	DOB data output to B register
EXB : 16§	DOC data output to C register
EXB : 17§	Send control

INTERFACE REGISTERS

Each peripheral controller has three interface registers: A, B, and C. These registers are used for control and data communication between the peripheral device and the I/O Processor. The specific uses of the registers are defined by the peripheral device controller.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

CHANNEL ASSIGNMENTS

All peripheral devices handled by the peripheral expander share the same channel number. Device addresses select among the peripheral units.

EXB : 0 - IDLE CHANNEL

This instruction clears the peripheral expander channel busy and done flags, clears the peripheral expander interrupt enable flag, and clears the interface DMA (direct memory access) enable flag. The peripheral expander channel is then inactive and DMA references via the data channel will not be allowed.

EXB : 1 - DIA

This instruction requests the A input register contents from the selected peripheral controller in the peripheral expander. Peripheral expander channel busy sets and done clears, both in the clock period following issue. Since this command requires the use of the data bus, it is delayed by the function delay time (minimum 1 microsecond). When it completes, the peripheral expander channel done flag sets and the busy flag clears. At this time, the I/O Processor may follow this instruction with an EXB : 10 to load the A input register information into the I/O Processor accumulator.

EXB : 2 - DIB

This instruction performs exactly as the EXB : 1 instruction, except that the peripheral controller B input register is sampled.

EXB : 3 - DIC

This instruction performs exactly the same as the EXB : 1 instruction, except that the peripheral controller C input register is sampled.

EXB : 4 - READ BUSY/DONE, INTERRUPT NUMBER

This instruction serves two purposes. The first purpose is to return the specified peripheral controller busy and done flags to the interface. The second is to determine which peripheral controller has the highest priority interrupt, its done flag set, and its interrupts enabled.

This is a delayed function because it uses the bus data lines from the peripheral expander. The peripheral expander channel busy flag sets and done flag clears 1 CP after instruction issue. Upon completion, the peripheral expander channel busy flag is cleared and the done flag is set. At this time, an EXB : 11 instruction may be issued to load the peripheral controller busy flags, done flags, and the present interrupt device address.

EXB : 5 - LOAD DEVICE ADDRESS

This instruction loads the bits 2^0 - 2^5 of the I/O Processor accumulator into the interface device address register. The interface device address register is used to hold the device address of a peripheral controller to which a delayed function is being sent. A delayed function is any function which requires the use of the bus data lines in the expander, thus requiring time to complete. Functions that are considered delayed functions are the EXB : 1, 2, 3, 4, 6, 14, 15, 16, and 17 functions. The address in the interface device address register must not be changed at any time during the execution of a delayed function. This instruction does not change the peripheral expander channel done and busy flags. The device address is loaded into the interface register in the I/O Processor clock period following instruction issue.

EXB : 6 MSKO MASK OUT

This instruction sends the present contents of the I/O Processor accumulator to the peripheral expander where the 16-bit word acts as a mask to disable interrupts from specific peripheral controllers. Every device controller in the peripheral expander has a mask bit. The specific mask bits for some probable peripheral devices are given in table 7-6. The mask bits are subject to change, so the latest documentation for the peripheral interface should be consulted. Three Cray Research assignments are shown on the table. If the mask bit is set, interrupts from the corresponding peripheral controller are disabled. The peripheral expander channel busy flag is set and the done flag is cleared upon initiation of this instructon. Because the mask must be sent throughout the peripheral expander via the bus data lines, this instruction is a delayed instruction. When it does complete, the peripheral expander channel busy flag clears and the done flag sets.

Issuing this instruction clears the interrupt flag. If an interrupt condition is still present 400 nanoseconds after completion, the interrupt flag will set again.

Table 7-6. Peripheral device mask bits for interrupt disabling

Octal Device Code	Mask bit	Device
11/51§	2^0	Teletype output
11/50§	2^1	Teletype input
30/70§		Asynchronous hardware multiplexer
70		Synchronous Line Adapter
14/54§	2^2	Real-time clock option
31,32/71,72§		IBM 360/370 interface
06/46§	2^3	Multiprocessor adapter transmitter
07/47§		Multiprocessor adapter receiver
15/55§		Incremental plotter
17/57§		Card Reader CR0 (CRI modification)
44		Modem control for multiline asynchronous controller
34,35/74,75§	2^4	Multiline asynchronous controller
16/56§	2^5	Magnetic tape MT0 (CRI)
22/62§		Cassette tape
34/74§		
20/60§	2^6	Fixed head disk
21/61§	2^7	Analog/Digital converter
40		Interprocessor bus full-duplex unit
41		Interprocessor bus full-duplex unit
40§§		Synchronous communication receiver
41§§§		Synchronous communication transmitter
33/73	2^8	Moving head disk
42		Digital I/O
43	2^9	Line Printer PR0 (CRI modification)
64,65,66/74	2^{10}	Floating-point (NOVA only)
76,76§		
	2^{11}	Unassigned
	2^{12}	Unassigned
	2^{13}	Unassigned
	2^{14}	Unassigned
	2^{15}	Data communications multiplexer

§ First device code/second device code
§§ May be set up with any unused even device code greater than 40_8
§§§ May be set up with any unused odd device code greater than 41_8

EXB : 7 - SET INTERRUPT MODE

This instruction is used to enable or disable interrupts for the peripheral expander I/O channel and for the peripheral controllers. The accumulator value present when this instruction issues determines what types of interrupts are honored. If a bit is set, the interrupts will be enabled as follows:

Accumulator bit 2^0 - Enables interrupts from the I/O channel to the I/O Processor

Accumulator bit 2^1 - Enables interrupts from the I/O controllers to the I/O Processor

The I/O channel busy and done flags are not affected by this instruction.

EXB : 10 - READ DATA BUS STATUS

This instruction reads the data on the data bus into the I/O Processor accumulator. This is normally used to bring back the data received from the bus by the EXB : 1, 2, or 3 input instructions. The peripheral expander I/O channel busy and done flags are not effected by this instruction. The data will stay valid until another EXB : 1, 2, or 3 function is issued.

EXB : 11 - READ STATUS 1

This instruction returns to the I/O Processor the status of the peripheral expander I/O channel and of the peripheral controller. Channel busy and done flags are not affected by this instruction. The status bits are assigned as shown in table 7-7.

Table 7-7. Read status 1 bit assignments

Bit	Meaning
2^0	Interrupting device code bit 2^0
2^1	Interrupting device code bit 2^1
2^2	Interrupting device code bit 2^2
2^3	Interrupting device code bit 2^3
2^4	Interrupting device code bit 2^4
2^5	Interrupting device code bit 2^5
2^6	Unassigned
2^7	Direct memory access enabled
2^8	Expander I/O channel interrupts enabled
2^9	Controller interrupts enabled
2^{10}	Function active - delayed function executing
2^{11}	Expander I/O channel busy flag
2^{12}	Expander I/O channel done flag
2^{13}	INTR - interrupt request from peripheral interface
2^{14}	SELB - select busy flag of addressed device
2^{15}	SELD - select done flag of addressed device

EXB : 13 - READ STATUS 2

This instruction returns to the I/O Processor the status of the peripheral expander I/O channel and of the peripheral controller. Channel busy and done flags are not affected by this instruction. The instruction is similar to EXB : 11, except that the contents of the device address register in the interface is returned instead of the interrupt device code number. Status bit assignments are shown in table 7-8.

Table 7-8. Read status 2 bit assignments

Bit	Meaning
2^0	Device address bit 2^0
2^1	Device address bit 2^1
2^2	Device address bit 2^2
2^3	Device address bit 2^3
2^4	Device address bit 2^4
2^5	Device address bit 2^5
2^6	Unassigned
2^7	Direct memory access enabled
2^8	Expander I/O channel interrupts enabled
2^9	Controller interrupts enabled
2^{10}	Function active - delayed function executing
2^{11}	Expander I/O channel busy flag
2^{12}	Expander I/O channel done flag
2^{13}	INTR - interrupt request from peripheral interface
2^{14}	SELB - select busy flag of addressed device
2^{15}	SELD - select done flag of addressed device

EXB : 14 - DOA (DATA OUT A)

This instruction sends the contents of the I/O Processor accumulator to the selected peripheral controller in the peripheral expander. The data goes into the A register of the peripheral controller whose device address is currently in the device address register. This command causes the peripheral expander I/O channel busy flag to set and the done flag to clear. This command which requires the use of the data bus in the peripheral expander and is a delayed function. When it does complete, the peripheral expander I/O channel done flag sets and busy clears.

EXB : 15 - DOB (DATA OUT B)

This instruction sends the contents of the I/O Processor accumulator to the selected peripheral controller in the peripheral expander. The data goes into the B register of the addressed device controller. The instruction operation is otherwise the same as the EXB : 14 DIA instruction.

EXB : 16 - DOC (DATA OUT C)

This instruction sends the I/O Processor accumulator contents to the C register of the selected peripheral controller in the peripheral expander. Otherwise, the operation is the same as the EXB : 14 DOA instruction operation.

EXB : 17 - SEND CONTROL

The peripheral controllers use four control signals in the peripheral expander: I/O Reset, Pulse, Clear, and Start. Each controller may have different uses for these signals. This instruction sends the control signal selected by the bits in the accumulator at instruction issue. The control signals are delayed instructions. The peripheral expander I/O channel busy flag sets at issue time and the done flag clears. Then when the command has completed, the channel done flag sets and the busy flag clears. The accumulator bits set control signals as shown in table 7-9.

Table 7-9. Accumulator bit control signals

Bit	Function
2^0	Start (S)
2^1	Clear (C)
2^2	Pulse (P)
2^3	I/O reset (IORST)

DELAYED FUNCTIONS

Several functions require the data bus in the peripheral expander. These need extra time to complete, and may have to wait for the data bus to be free. The completion of a delayed function is shown by setting the peripheral expander I/O channel done flag. The done flag for the preceding delayed function must be received at the I/O Processor before sending another delayed function.

Those functions that are considered delayed functions are:

EXB : 1	DIA
EXB : 2	DIB
EXB : 3	DIC
EXB : 4	Read busy/done, interrupt number
EXB : 6	MSKO
EXB : 14	DOA
EXB : 15	DOB
EXB : 16	DOC
EXB : 17	Send control

TRANSFER SPEEDS

The peripheral expander interface is capable of sustaining a transfer rate of 16 megabits per second from the I/O Processor to the expander chassis. In the reverse direction, it can sustain a speed of approximately 14.5 megabits per second. This is achieved by using the data channel mode which transfers a 16-bit parcel every microsecond. The programmed I/O mode can also reach these speeds.

CHANNEL FOR INPUT FROM CRAY-1 CHANNEL

The I/O Processor may have one or more channels dedicated to receiving data from a CRAY-1 I/O channel. Data is transferred in block mode directly into I/O Processor I/O Memory. The functions are listed below.

CIA : 0§	Clear channel
CIA : 1§	Enter I/O Memory address, start transfer
CIA : 2	Enter parcel count
CIA : 3	Clear channel parity error flags
CIA : 4	Clear ready waiting flag
CIA : 6§§	Clear interrupt enable flag
CIA : 7§§	Set interrupt enable flag
CIA : 10	Read I/O Memory address
CIA : 11	Read ready waiting/error flags

I/O MEMORY ADDRESS REGISTER

The I/O Memory address register contains the I/O Memory address for the next I/O Memory reference. It is loaded with the I/O Memory starting address at the initiation of the input transfer. The low-order 2 bits of starting address are forced to 0 when loaded into the register. This is done because the memory references are done in bursts of four parcels, and upon completion of a reference, the register address is increased by 4. If, for some reason, the input transfer stops on some boundary other than four, the final memory reference stores undefined parcels. The number of defined parcels can be determined by reading the final memory address, which is equal to the address of the last defined parcel, plus 1.

CIA : 0 - CLEAR CHANNEL

This function clears the interface channel busy and done flags and aborts any transfer in progress. The new states of busy and done are not valid until the second clock period following this function. Wait one clock period after issuing this function before sampling these flags.

---

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

CIA : 1 - ENTER I/O MEMORY ADDRESS

This function enters the accumulator content into the 16-bit I/O Memory address register (forcing the 2 low-order bits to 0) and starts an input transfer from the Central Processing Unit. The busy flag is set and the done flag is cleared. Upon receipt of four parcels, a memory reference is made. Then another four parcels are received and stored. This continues until the parcel count is 0 or until a disconnect is received from the CRAY-1 I/O channel. At that time, the done flag is set and the busy flag is cleared.

CIA : 2 - ENTER PARCEL COUNT

This function stores the accumulator content into the interface parcel count register. This value is a positive count of the number of parcels to be transferred. The status of the busy and done flags are not affected.

CIA : 3 - CLEAR CHANNEL PARITY ERROR FLAGS

Four parity bits protect the 16-bit parcels on the CPU channel. A parity error in a 4-bit group causes one parity error flag to set. This funtion clears all four parity error flags. A parity error does not affect the states of the channel busy and done flags. Similarly, issuing this function does not affect the states of the channel busy and done flags.

CIA : 4 - CLEAR READY WAITING

If the I/O Processor input channel is inactive and a ready pulse is received from the CPU, the channel sets a ready waiting flag. If, in order to resynchronize the transfer, this ready must be discarded, this function will clear the ready waiting flag. If the ready signal is not discarded and the input channel is started, the data on the lines will be sampled as the first parcel of the transfer. The channel busy and done flags are not affected.

CIA : 6 - CLEAR INTERRUPT ENABLE FLAG

This function disables the interface from interrupting the I/O Processor. The channel may still be monitored via the busy and done flag status. The channel busy and done flags are not affected by this function.

CIA : 7 - SET INTERRUPT ENABLE FLAG

This enables interrupts on the interface. The channel will then interrupt whenever the done flag sets. The channel busy and done flags are unaffected by this function.

CIA : 10 - READ MEMORY ADDRESS

This function transfers the content of the interface I/O Memory address register into the accumulator. The address is one greater than the address of the last parcel stored. Since a memory reference is four parcels of data, there will be undefined parcels of data written into memory if the transfer length is not a multiple of four. The channel busy and done flags are unaffected by this function.

CIA : 11 - READ READY WAITING/ERROR FLAGS

This function reads into the accumulator the content of the interface status register. The status bit assignment is listed in table 7-10.

Table 7-10. Ready waiting/error flags

Bit	Meaning
2^0	Channel parity error flag for bits 2^0-2^3
2^1	Channel parity error flag for bits 2^4-2^7
2^2	Channel parity error flag for bits 2^8-2^{11}
2^3	Channel parity error flag for bits 2^{12}-2^{15}
2^{15}	Ready waiting flag

CHANNEL FOR OUTPUT TO CRAY-1 CHANNEL

The I/O Processor may have one or more channels for sending data to a CRAY-1 input channel. Data is transferred in block mode directly from I/O Memory. The functions are listed below.

COA : 0§	Clear channel
COA : 1§	Enter I/O Memory address, start transfer
COA : 2	Enter parcel count
COA : 3	Clear error flag
COA : 4	Set/clear external control signals
COA : 6§§	Clear interrupt enable flag
COA : 7§§	Set interrupt enable flag
COA : 10	Read I/O Memory address
COA : 11	Read error flags

I/O MEMORY ADDRESS REGISTER

The I/O Memory address register contains the I/O Memory address for the next I/O Memory reference. It is loaded with the I/O Memory starting address at the initiation of the output transfer. The 2 low-order bits of the starting address are forced to 0 when entered in the register. This is done because the memory references are in bursts of four parcels, and upon completion of a memory reference, the register address is increased by 4. Upon completion of the output, the I/O Memory address in the register is one greater than the address of the first parcel sent.

COA : 0 - CLEAR CHANNEL

This function clears the channel busy and done flags and aborts any transfer. The busy and done flags are not valid for sampling until the second clock period following completion of this function.

---

§ Allow 1 CP before checking busy and done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

COA : 1 - ENTER I/O MEMORY ADDRESS

This function loads the current contents of the accumulator into the 16-bit I/O Memory address register (forcing the 2 low-order bits to 0) and starts the transfer to the CPU. The busy flag sets and the done flag clears. The interface makes a memory reference starting at the address contained in the I/O Memory address register, and outputs four parcels of data. Then another reference reads out another burst of four parcels. When the parcel count stored in the interface register has been reached, the transfer stops. The done flag sets and the busy flag clears.

COA : 2 - ENTER PARCEL COUNT

This enters the accumulator content into the interface parcel count register. This value is a positive count of the number of parcels to be transferred. The channel busy and done flags are not affected by this function.

COA : 3 - CLEAR ERROR FLAG

This command is used to clear the sequence error flag. The sequence error flag sets when there is a resume signal received from the CPU and the interface is not busy or there is an I/O Memory reference in progress. If the interface interrupt enable flag is set, the sequence error flag will cause an interrupt. The sequence error has no affect on the channel busy and done flags. The channel busy and done flags are not affected by this function.

COA : 4 - SET/CLEAR EXTERNAL CONTROL SIGNALS

This function sends control signals to the CPU. The signal selection is governed by set bits in the accumulator as shown in table 7-11.

Hold disconnect is used to stop the automatic disconnect that is sent at the end of a transfer.

Write disconnect sends a disconnect to the CPU with no data transferred.

Error channel resume is reserved for maintenance use with configurations without an error logging channel. RTC interrupt is reserved for use on systems where a programmable clock is not present.

The accumulator signal select bits are held in an interface register until they are altered by another COA : 4 function. The channel busy and done flags are not affected by this command.

Table 7-11. External control signal bits

Bit	Control Signal
2^8	Write disconnect
2^9	Hold disconnect
2^{10}	Unused
2^{11}	Dead dump
2^{12}	RTC interrupt
2^{13}	Error channel resume
2^{14}	I/O master clear
2^{15}	CPU master clear

COA : 6 - CLEAR INTERRUPT ENABLE FLAG

This function clears the channel interrupt enable flag to prevent the interface from interrupting the I/O Processor. The channel may be monitored via the busy and done flags and via the status information returned by the COA : 11 command.

COA : 7 - SET INTERRUPT ENABLE FLAG

This function sets the interrupt enable flag to allow interrupting the I/O Processor. With interrupts enabled, the interface will interrupt whenever the done flag sets or whenever a sequence error occurs. This function does not affect the busy and done flags.

COA : 10 - READ I/O MEMORY ADDRESS

This command enters the content of the I/O Memory address register into the accumulator. The address entered will be one greater than the address of the last parcel transferred from I/O Memory. The channel busy and done flags are not affected by this command.

COA : 11 - READ ERROR FLAGS

This function reads into the accumulator the status of the interface. The bit significances are listed in table 7-12.

Table 7-12. Error flags

Bit	Meaning
2^0	4-bit channel data bit 2^0
2^1	4-bit channel data bit 2^1
2^2	4-bit channel data bit 2^2
2^3	4-bit channel data bit 2^3
2^{15}	Sequence error

The channel busy and done flags are not affected by this command. The 4-bit channel is reserved for maintenance use with configurations without an error logging channel.

MEMORY CHANNEL

A Memory Channel (1 standard and 1 optional) is used for transferring data between the CPU Central Memory and an IOP I/O Memory. A Memory Channel consists of an input channel that carries data from Central Memory to I/O Memory, and an output channel that passes data from the I/O Memory to the Central Memory. Each channel carries 64 bits of data in parallel, along with 8 check bits. A separate 12-bit channel is provided for both the input channel and the output channel for carrying address information. A data rate of approximately 800 million bits per second is possible on either the input channel or the output channel.

The input channel and the output channel are more complex than the normal CRAY-1 channels. There are more control signals provided, and more data protection information. The signals and their timing are described in this section, followed by an explanation of the IOP instructions that control a Memory Channel.

Figure 7-8 shows the Memory Channel signals and the directions of the signals. First the input channel signals are described, then the output channel signals are described.

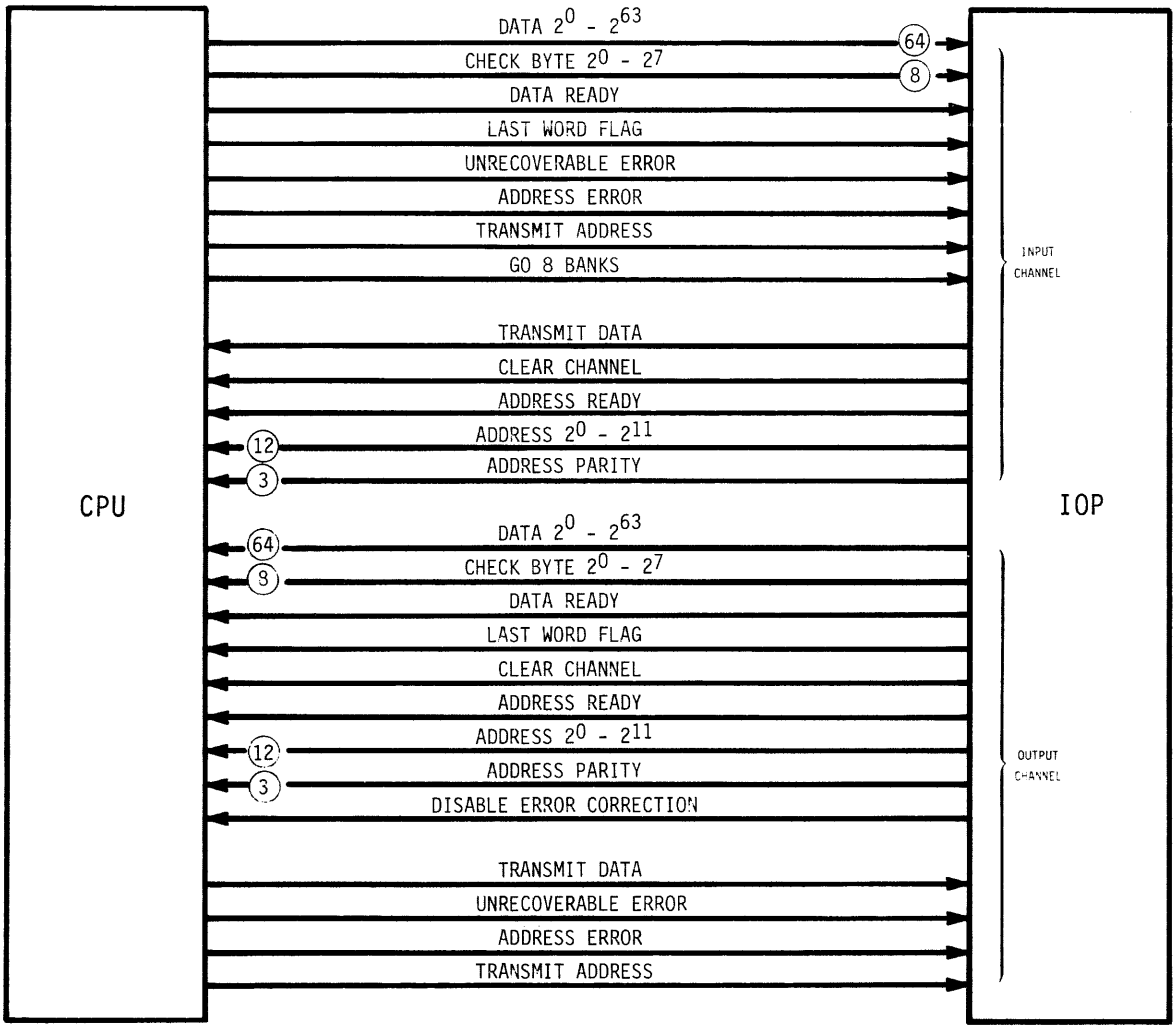


Figure 7-8. Memory Channel signals

Central Processing Unit to I/O Processor Input Channel

Data (to IOP) - The data is passed in 64-bit words over 64 lines. Data is valid 2 CPs after the leading edge of Data Ready, and stays valid for 6 CPs.

Check Byte (to IOP) - The Check Byte is the 8 error check bits read out of the CPU memory. It is passed to the I/O Processor to verify the accuracy of the received Data. The Check Byte has the same timing and valid points as the Data Signal.

Data Ready (to IOP) - This signal indicates data is valid on the cable. The leading edge of Data Ready preceded the Data signals by 2 CPs and is true for 1 CP. The minimum period for Data Ready signals is 6 CPs.

Last Word Flag (to IOP) - This signal indicates the last data word of the complete transfer is on the Data lines. This also means the CPU word count is 0. This signal has the same timing and duration as the Data signals.

Unrecoverable Error (to IOP) - This signal indicates an unrecoverable error occurred in the transfer at the CPU. The channel will go inactive and remain so until a Clear Channel signal is sent to the CPU from the IOP. Unrecoverable error conditions include:

1. Address parity error,

2. Transmit Data signal received at the CPU and less than 3 Address Ready signals were received,

3. More than 3 Address Ready signals received at the CPU,

4. Uncorrectable data error from Central Memory.

The unrecoverable error signal continues until a Clear Channel is sent.

Address Error (to IOP) - The Address Error signal indicates the unrecoverable error was an address error, from conditions 1, 2 or 3 of the error conditions listed previously in "Unrecoverable Error". The Address Error signal should be sampled when the leading edge of the Unrecoverable Error Signal appears.

Transmit Address (to IOP) - This signal indicates the CPU side is inactive and that a transfer can be initiated by sending an address on the Address line. Transmit Address returns to the cleared state after three Address Ready signals, and will not set again until after the last data word has been transferred to the IOP Processor. This signal also clears if there is an error condition on the CPU side of the channel and will not set again until a Clear Channel is sent.

Go 8 Banks (to IOP) - This signal indicates the Central Memory is operating in the 8-bank mode. All transmitted block sizes should be reduced to eight words.

Transmit Data (to CPU) - This signal indicates the I/O Processor can accept a block of data from the CPU. This signal remains set until the first Data Ready signal is received for that data block. Transmit Data is sampled at the CPU at the beginning of each data block. If Transmit Data is set, the 16-word data block will be transmitted. If this signal is cleared at that time, no data will be transmitted until the Transmit Data signal sets. The Transmit Data signal is clear when the channel is inactive and will not set until at least 100 nanoseconds after the third Address Ready signal trailing edge.

NOTE

The number of 64-bit data words sent in the first block equals 16 minus the number specified by bits 2^0 and 2^1 of the CPU starting address. This adjusts the blocks to 16-bank boundaries so that the next block begins with section 0. If the Go 8 Bank signal is set, the number of data words in the first block is 8 minus the address 2^0 value.

Clear Channel (to CPU) - This signal clears all error conditions in the channel. It remains set a minimum of 100 nanoseconds or until the Unrecoverable Error signal goes false. An Address Ready signal must lag the trailing edge of the Clear Channel signal by at least 100 nanoseconds.

Address Ready (to CPU) - This signal indicates the Address lines will soon have valid address data. The leading edge of this signal leads the address information by 25 nanoseconds and stays set for 12.5 nanoseconds. The minimum period for Address Ready signals is 75 nanoseconds, leading edge to leading edge. Three Address Ready signals are used to initiate a transfer. The first accompanies bits 2^{10} through 2^{21} of the beginning central memory address on the Address lines. The second Address Ready accompanies bits 2^0 - 2^9 of the beginning Central Memory address and bits 2^{13} - 2^{12} of the transfer word count. The third address Ready accompanies bits 2^0 - 2^{11} of the transfer word count. See figure 7-9.

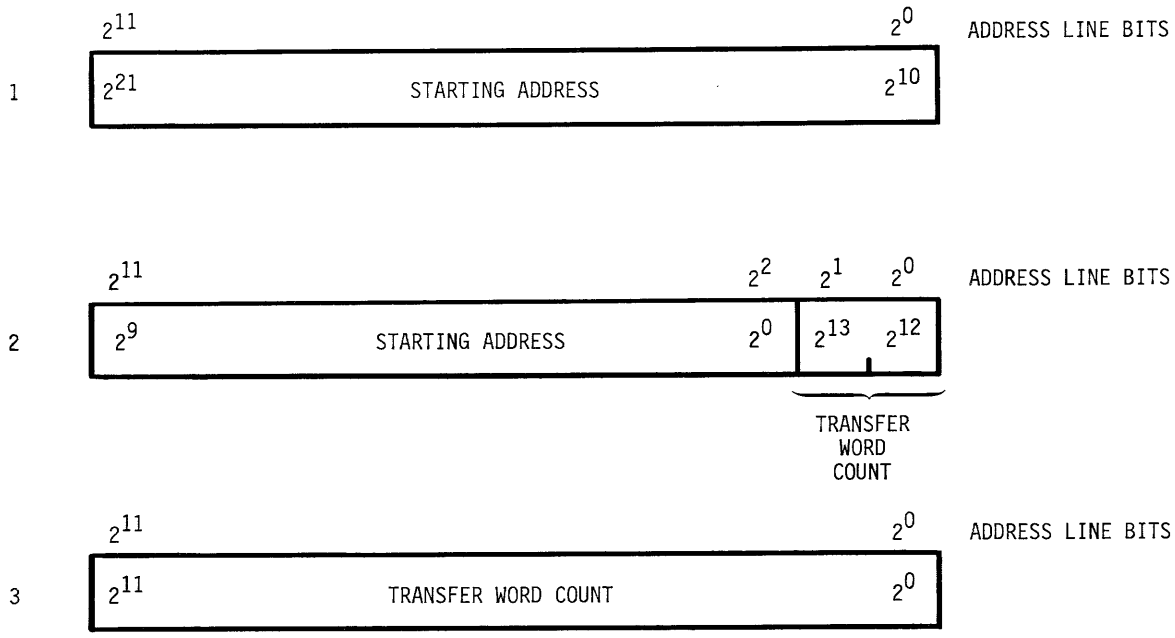


Figure 7-9. Address and word count formats

Address (to CPU) - The 12 Address lines carry the Central Memory starting address and the transfer word count. The information carried on the Address lines for each of the three address transfers is shown in figure 7-9. The information is valid for 75 nanoseconds. The Address signals lag the leading edge of Address Ready by 25 nanoseconds.

Address Parity (to CPU) - These signals provide odd parity for each group of 4 address bits as follows:

Address Parity Bit	Address Information Bits
2^0	$2^0 - 2^3$
2^1	$2^4 - 2^7$
2^2	$2^8 - 2^{11}$

The Address Parity signals have the same timing as Address signals. They are valid for 75 nanoseconds.

I/O Processor to Central Processing Unit Output Channel

Data (to CPU) - Data passes in 64-bit words over 64 lines. Data is valid 25 nanoseconds after the Data Ready signal leading edge and Data stays valid for 75 nanoseconds.

Check Byte (to CPU) - Check Byte is the 8 error check bits generated by the output side of the channel. The error code generation methods used are the same on each side of the channel. The Check Byte is passed to the CPU with the same timing and valid points as the Data signals.

Data Ready (to CPU) - This signal indicates data is valid on the cable. The leading edge of Data Ready precedes the Data signals by 25 nanoseconds and Data Ready is set for 12.5 nanoseconds. The minimum period for Data Ready signals is 75 nanoseconds.

Last Word Flag (to CPU) - This signal indicates the last data word of the complete transfer on the Data lines. This also means the I/O Processor word count is 0. This signal has the same timing and duration as the Data signals.

Clear Channel (to CPU) - This signal clears all error conditions in the channel. It remains set a minimum of 100 nanoseconds or until the Unrecoverable Error signal clears. An Address Ready signal must lag the trailing edge of the Clear Channel signal by at least 100 nanoseconds.

Address Ready (to CPU) - This signal indicates information will soon be valid on the Address lines. The leading edge of Address Ready leads the valid address information by 25 nanoseconds and is set for 12.5 nanoseconds. The minimum period for sequential Address Ready signals is 75 nanoseconds. The Address Ready signal is only sent when the Transmit Address signal is set. Three Address Ready signals are used to initiate a data block transfer. The first Address Ready signal accompanies Address information which is bits 2^{10} - 2^{21} of the starting address in Central Memory. The second Address Ready indicates the Address lines will soon contain bits 2^{0} - 2^{9} of the Central Memory starting address and bits 2^{12} - 2^{13} of the transfer word count. The third address information transmission preceded by an Address Ready, contains the 2^{0} - 2^{11} bits of the transfer word count. See preceding figure 7-9.

Address (to CPU) - The 12 Address lines carry the Central Memory starting address and the transfer word count. The Address signals lag the leading edge of Address Ready by 25 nanoseconds, and then stay set for 75 nanoseconds. The formats of the information carried in each of three address information transfers are shown in the preceding figure 7-9.

Address Parity (to CPU) - These signals provide odd parity for each group of four Address signals as follows:

Address Parity Bit	Address Information Bits
2^0	$2^0 - 2^3$
2^1	$2^4 - 2^7$
2^2	$2^8 - 2^{11}$

The Address Parity signals have the same timing as Address Signals. They are valid for 75 nanoseconds.

Disable Error Correction (to CPU) - This signal is used for diagnostic purposes only. It disables the Single Error Correction, Double Error Detection (SECDED) circuitry used on the data channel.

Transmit Data (to IOP) - This signal indicates the CPU can accept a block of data from the I/O Processor. This signal remains set until the first data ready signal is received for that data block. Transmit data is sampled at the beginning of each data block. If Transmit Data is set, the 16-word data block will be transmitted. If this signal is cleared when sampled, no data will be sent until Transmit Data does go set.

NOTE

The number of 64-bit words sent in the first block of a transfer equals the number of banks (16 or 8), minus the number specified by bits 2^0 and 2^1 (16 banks) or by bit 2^0 (8 banks) of the Central Memory starting address. This adjusts the blocks to 16-bank or 8-bank boundaries so that the next block begins with bank 0.

Unrecoverable Error (to IOP) - This signal indicates an unrecoverable error occurred in the transfer at the CPU side. The channel will go inactive and remain so until a Clear Channel is sent to the CP. Unrecoverable Error conditions include:

1. Address Parity Error,
2. Data Ready received and less than three Address Ready signals,
3. More than three Address Readies received,
4. Uncorrectable data error on channel,
5. Transfer word count = 0 and no Last Word Flag,
6. Transfer word count ≠ 0 and Last Word Flag.

The Unrecoverable Error signal remains set until a Clear Channel signal is sent.

Address Error (to IOP) - The Address Error signal indicates the Unrecoverable Error was an address error, from conditions 1, 2, or 3 of the preceding Unrecoverable Error list. The Address Error signal should be sampled when the leading edge of Unrecoverable Error appears.

Transmit Address (to CPU) - This signal indicates the CPU side is inactive and a transfer can be initiated by sending data over the Address lines. This signal will clear after three Address Ready signals and will not set again until the last word of the complete channel transfer has been stored in Central Memory. Transmit Address will clear if there is an error condition on the CP side and in that case Transmit Address will not set again until Clear Channel is sent.

MEMORY CHANNEL FUNCTIONS FOR INPUT FROM CPU

The following describes the input operation functions for the Memory Channel. The functions are listed below.

HIA : 0§	Clear channel busy, done flags
HIA : 1	Enter I/O Memory address
HIA : 2	Enter upper Central Memory address
HIA : 3	Enter lower Central Memory address
HIA : 4§	Read Central Memory, enter block length
HIA : 6§§	Clear interrupt enable flag
HIA : 7§§	Set interrupt enable flag
HIA : 14	Enter diagnostic mode

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10)

Interface Registers

Three interface registers are used to control the transfer. The Central Memory starting address is held in a 22-bit register. Two separate commands, HIA : 2 and HIA : 3 load the 22-bit address. The I/O Processor I/O Memory starting address is held in a 16-bit register that is loaded by an HIA : 1 command. The third register holds 14 bits of the transfer word count, or block length, entered by the HIA : 4 function.

HIA : 0 - Clear Channel Busy, Done Flags

This function clears the channel busy and done flags. It will cause an error condition if issued while this channel is active. HIA : 0 must be used to clear an error condition on the channel.

HIA : 1 - Enter I/O Memory Starting Address

This function enters the content of the I/O Processor accumulator into the I/O Memory address register. The 2^0, 2^1 bits of the address are forced to 0. This command does not alter the state of the channel busy or done flags. If this function is given while this channel is active, an error conditiion occurs.

HIA : 2 - Enter Upper Central Memory Address

This function enters the 2^0 - 2^{12} bits of the I/O Processor accumulator into the 2^9 - 2^{21} positions of the Central Memory starting address register. The states of the channel busy and done flags are unaltered. This command will cause an error condition if issued while this channel is active.

HIA : 3 - Enter Lower Central Memory Address

This function enters the 2^0 - 2^8 bits of the I/O Processor accumulator into the 2^0 - 2^8 bits of the Central Memory starting address register. The states of the channel busy and done flags are unaltered. This command will cause an error condition if issued when the input channel is active.

HIA : 4 - Read Central Memory, Enter Block Length

This command enters the lowest 2^0 - 2^{13} bits of the I/O Processor accumulator into the block length register. (This value is treated as a count of 64-bit words with a zero value indicating the maximum 16,384 words.) The busy flag is set and the done flag is cleared and the transfer is initiated. Upon completion, the done flag will set and the busy flag will clear. If during the transfer an unrecoverable error occurs, the transfer will terminate and the done flag will set and the busy flag will remain set. This command will cause an error condition if issued while the input channel is active.

HIA : 6 - Clear Interrupt Enable

This function clears the interrupt enable flag for the channel. The states of the busy and done flags remain unaltered.

HIA : 7 - Set Interrupt Enable

This function sets the interrupt enable flag for the channel. The states of the busy and done flags remain unaltered.

HIA : 14 - Enter Diagnostic Mode

This function enters the 2^0 - 2^2 bits of the I/O Processor accumulator into the diagnostic mode register. The modes are summarized in table 7-13. Only one mode is valid at a time. All diagnostic modes are cleared with a master clear, or alternately, by doing HIA : 0, followed by HIA : 14 with the accumulator = 0. Mode 0 is active until any other mode is selected. This function is for maintenance purposes only.

Table 7-13. Input channel diagnostic modes

Mode Designator	Function
0	Set Last Word flag (only if channel is active)
1	Disable Last Word flag from CPU
2	Force constant Address Ready
3	Disable third Address Ready
4	Transfer first address with a "0" parity bit, third address with a "1" parity bit, and disable Data Ready from CPU
5	Force Clear Channel without inactivating IOP
6	Disable Block Length = 0
7	Disable IOP data error correction and detection

Memory Channel Input Error Processing

If an irrecoverable error occurs in an input transfer from the Central Memory, the busy and done flags will both set. An error code is generated and sent to an error log. The error codes are summarized in table 7-14.

Memory Channel Input Sequence

Figure 7-10 shows the sequence of signals for transferring data from Central Memory to I/O Memory.

The only recoverable error is correctable data error. This type of error is ignored by the channel because the data is corrected automatically; the syndrome information is sent to an error log.

Table 7-14. Input channel error codes

Error Code	Name	Condition
1	Function Error	Indicates a function 0, 1, 2, 3 or 4 was issued while the channel was active.
2	Active Error	Indicates the CPU side went inactive while the IOP side was still active.
3	Transmit Address Timeout	Indicates an IOP input was initiated but the CPU did not send a transmit address within 2 milliseconds.
4	CPU Address Error	Indicates the CPU side received greater or less than three address readies or there was a parity error in one of the address channel transfers.
5	CPU IOP Data Error	Indicates the CPU side has a multiple data error from memory or the IOP side received data with a multiple error.
6	Block Length Error	Indicates the IOP received the last word and the block length count was not 0 or the last word was not received and the block length count was equal to 0.
7	Data Ready Timeout	Indicates the IOP did not receive data within 2 milliseconds.

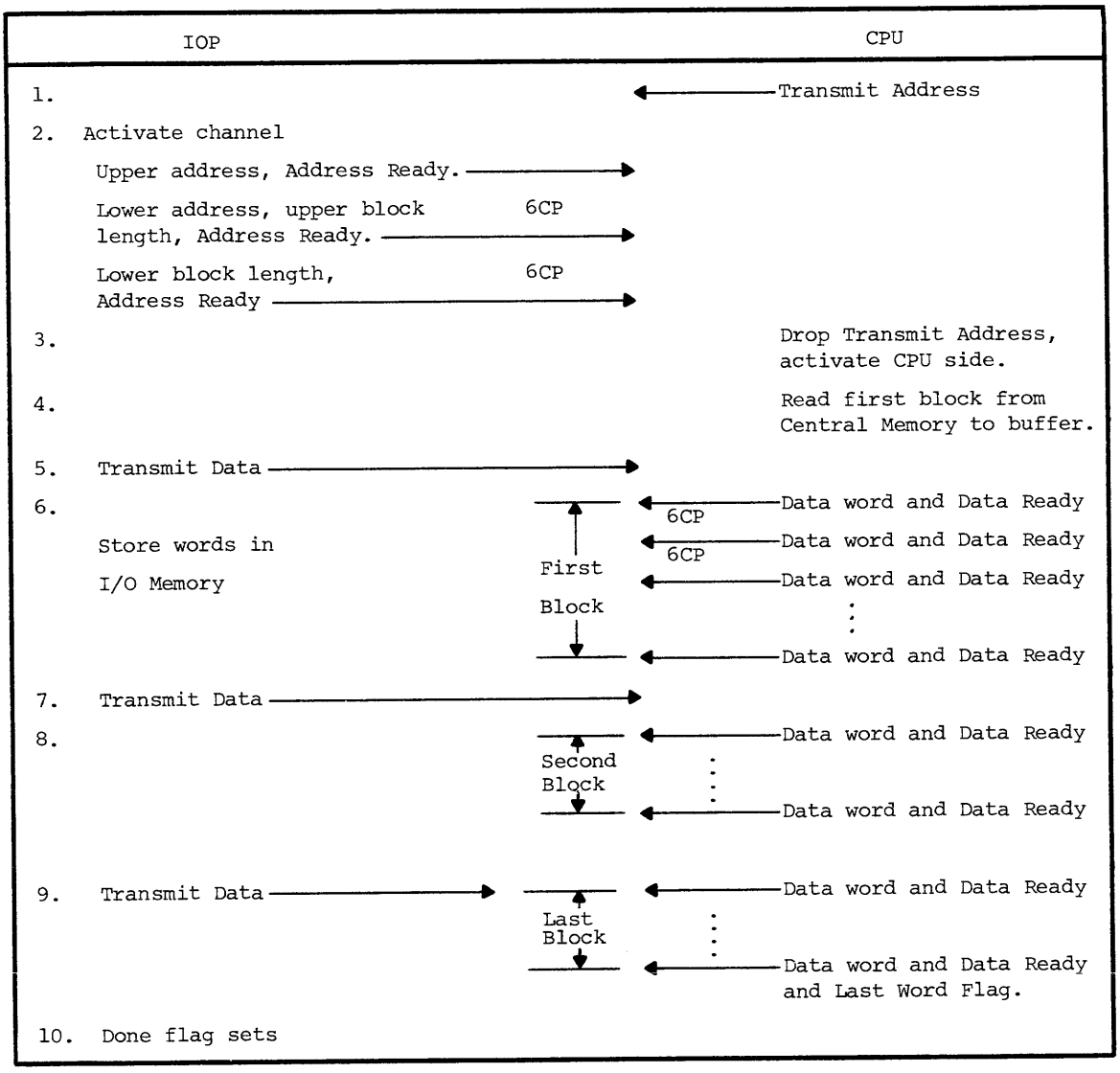


Figure 7-10. Memory Channel sequence, input to I/O Processor

MEMORY CHANNEL FUNCTIONS FOR OUTPUT TO CPU

The I/O Processor may have a channel for sending data to the CPU at over to 800 megabits per second. Transfers are done in blocks of 64-bit words under I/O Processor control. Error detection and correction are used to protect data quality. The functions are listed below.

HOA : 0§	Clear channel busy, done flags
HOA : 1	Enter I/O Memory address
HOA : 2	Enter upper Central Memory address
HOA : 3	Enter lower Central Memory address
HOA : 5§	Write Central Memory, enter block length
HOA : 6§§	Clear interrupt enable flag
HOA : 7§§	Set interrupt enable flag
HOA : 14	Enter diagnostic mode

Interface Registers

The interface has three registers used to control the output transfers. One 22-bit register holds the Central Memory starting address. The 22-bit address is loaded by the HOA : 2 and HOA : 3 functions. The I/O Processor I/O Memory starting address is held in a 16-bit register which is entered by the HOA : 1 function. A 14-bit register holds the transfer word count, or block length in 64-bit words, that is loaded by the HOA : 5 command.

HOA : 0 - Clear Channel Busy, Done Flags

This function idles the channel by clearing the busy and done flags. This function causes an error condition if issued while the output channel is active. This command must be used to clear an error condition in the channel.

HOA : 1 - Enter I/O Memory Address

This function enters the value of the I/O Processor accumulator into the I/O Memory address register. The 2^0 - 2^1 bits are forced to a zero value. The states of the busy and done flags are unaltered. This command causes an error condition if issued when the output channel is active.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

HOA : 2 - Enter Upper Central Memory Address

This function enters the 2^0 - 2^{12} bits of the I/O Processor accumulator into the 2^3 - 2^{15} bits of Central Memory starting address register. The states of the busy and done flag are unaltered. This function causes an error condition if issued when the output channel is active.

HOA : 3 - Enter Lower Central Memory Address

This function enters the 2^0 - 2^8 of the I/O Processor accumulator into the 2^0 - 2^8 bits of the Central Memory starting address register. The states of the busy and done flags are unaltered. This command causes an error condition if issued when the output channel is active.

HOA : 5 - Write Central Memory, Enter Block Length

This function enters the 2^0 - 2^{13} bits of the I/O Processor accumulator into the block length register. (This value is treated as a count of 64-bit words with a zero value indicating the maximum 16,384 words.) The busy flag is set and the done flag is cleared and the transfer is initiated. Upon completion, the done flag will set and the busy flag will clear. If during the transfer an irrecoverable error occurs, the transfer will terminate and the done flag will set and the busy flag will be left set. This command will cause an error condition if issued while the output channel is active.

HOA : 6 - Clear Interrupt Enable

This function clears the interrupt enable flag. The states of the busy and done flags are unaltered.

HOA : 7 - Set Interrupt Enable

This function sets the interrupt enable flag. The states of the busy and done flags are unaltered.

HOA : 14 - Enter Diagnostic Mode

This function enters the 2^0 - 2^7 bits of the I/O Processor accumulator into the diagnostic mode register. The modes are summarized in table 7-15. This is only for maintenance purposes.

With the exception of mode 6, only one mode is valid at a time. Mode 6 is a special case. It is held if it has been set and cannot be cleared by entering another mode. All diagnostic modes clear with master clear, or alternately by doing HOA : 0, followed by HOA : 14 with the accumulator = 0. Mode 0 is active until any other mode is selected.

Table 7-15. Output channel diagnostic modes

Mode Designator	Function
0	Set Last Word flag (only if channel is active)
1	Disable Last Word flag from CPU
2	Force constant Address Ready
3	Disable third Address Ready
4	Transfer first address with a "0" parity, third address with a "1" parity bit, and disable Data Ready to CPU
5	Force Clear Channel without inactivating IOP
6	First word 2^{56} - 2^{63} bits used as second word check byte, third word 2^{56} - 2^{64} bits as fourth word check byte, etc.
7	Disable error correction and detection as CPU

Central Memory Output Error Processing

If an irrecoverable error occurs in an output transfer to the Central Memory, the busy and done flags both set. An error code is generated and sent to an error log. The error codes are summarized in table 7-16.

The only recoverable error is a correctable data error. This type of error is ignored by the channel interface because it is corrected automatically. The syndrome information is sent to an error log.

Table 7-16. Output channel error codes

Error Code	Name	Condition
1	Function Error	Indicates a function 0, 1, 2, 3 or 5 was issued while the channel was active.
2	Active Error	Indicates the CPU side went inactive while the IOP side was still active.
3	Transmit Address Timeout	Indicates an IOP input was initiated but the CPU did not send a transmit address within 2 milliseconds.
4	CPU Address Error	Indicates the CPU side received greater or less than three address readies or there was a parity error in one of the address channel transfers.
5	CPU Data Length Error	Indicates the CPU received data with a multiple error or the last word was received and the block length count was not 0 or the last word was not received and the block length count was equal to 0.
6	Last Word Timeout	Indicates the last word was sent but the CPU did not go inactive within 2 milliseconds.
7	Transmit Data Timeout	Indicates a transmit data was not received within 2 milliseconds.

Memory Channel Output Sequence

Figure 7-11 shows the sequence of signals for transferring data from I/O Memory to Central Memory.

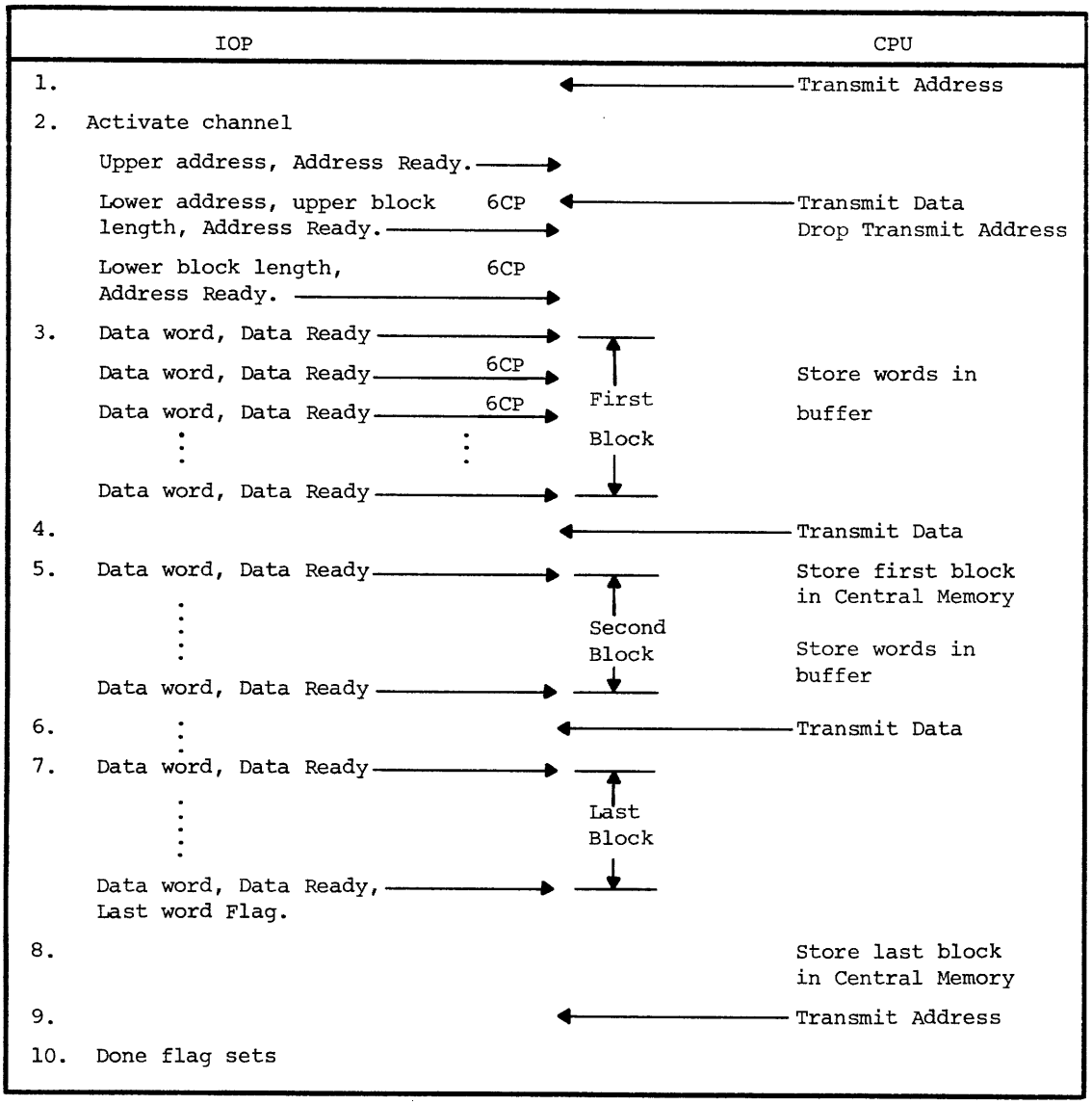


Figure 7-11. Memory Channel sequence, output from I/O Processor

ERROR LOGGING CHANNEL

The I/O Processor may have an error logging channel connected. This channel reports errors from the I/O Memories of three other I/O Processors, from the Buffer Memory, from the Central Memory, and from the Memory Channels to the Central Memory. If any error condition occurs in any of the reporting devices, the channel done flag sets. The functions are listed below.

ERA : 0§	Idle channel
ERA : 6§§	Clear interrupt enable flag
ERA : 7§§	Set interrupt enable flag
ERA : 10	Read error status
ERA : 11	Read error information (first parameter)
ERA : 12	Read error information (second parameter)
ERA : 13	Read error information (third parameter)

INTERFACE REGISTERS

The error log interface has four interface registers. The error status register is a 9-bit register that holds set bits for the device causing the error. This register is accessed by the ERA : 10 command.

The first parameter word register has 16 bits, and stores I/O Memory failing locations, or Buffer Memory error data, or Memory Channel error data.

The second parameter word register has 16 bits, and stores the lower address for the Buffer Memory or Central Memory failing locations.

The third parameter word register has 8 bits, and stores the upper address for either the Buffer Memory or the Central Memory.

ERA : 0 - IDLE CHANNEL

This function clears all error flags stored in the interface registers. The done flag is cleared.

§ Allow 1 CP before checking busy or done.
§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

ERA : 6 - CLEAR INTERRUPT ENABLE FLAG

This function clears the interrupt enable flag for this channel.

ERA : 7 - SET INTERRUPT ENABLE FLAG

This function enables the interface to interrupt the I/O Processor.

ERA : 10 - READ ERROR STATUS

This function returns the contents of the interface error status register to the accumulator. The interpretation of the set bits is given in table 7-17.

Table 7-17. Error status register bits

Bit	Control Signal
2^0	I/O Processor 1 I/O Memory error
2^1	I/O Processor 2 I/O Memory error
2^2	I/O Processor 3 I/O Memory error
2^3	Buffer Memory error
2^4	Central Memory error
2^5	Memory Channel input A error
2^6	Memory Channel output B error
2^7	Memory Channel input C error
2^8	Memory Channel output D error

The selection of physical devices such as I/O Processors and Memory Channels as A, B, C, etc. is part of the overall system definition. Appendix F has a typical system definition.

Errors occurring in the I/O Processor having the error logging channel are reported to that I/O Processor via its own I/O Memory error channel (LME).

ERA : 11 - READ ERROR INFORMATION (FIRST PARAMETER)

This function returns an error parameter word to the accumulator, selected by a bit set in the accumulator when the function issues. Only one bit among the 2^0 - 2^8 positions may be set at one time. The accumulator bits select parameters as listed in table 7-18.

When an I/O Processor I/O Memory address is read, or when a Memory Channel error data word is read, the device bit is cleared in the status register.

The address invalid flag (2^{15}) in the Buffer Memory error data is set when the address that will be given in the second and third parameter words is not the correct address for the error being reported in the first parameter word. This can happen when a later read reference follows closely after a read reference that detects an error in its data. The address for the later read reference is incorrectly reported in the second and third parameter words, and should be disregarded.

This function must be immediately preceded by a logical product instruction as follows:

```
011777  A=A&d  d=777
171xxx  ERA : 11
        or
015xxx  A=A&k
xxx777  k=xxx777
171xxx  ERA : 11
```

ERA : 12 - READ ERROR INFORMATION (SECOND PARAMETER)

This function returns the content of the second parameter word register to the accumulator. The second parameter word is selected by bits 2^3 or 2^4 in the accumulator when the function is sent. Only one bit may be set in the accumulator 2^3 and 2^4 bit positions at a time. Selection is as follows.

2^3 Buffer Memory lower address (address bits 2^0 - 2^{15})

2^4 Central Memory lower address (address bits 2^0 - 2^{15})

Table 7-18. First error parameter selection

Error Status Register Bit	First Parameter Information Returned
2^0	I/O Processor A I/O Memory failing address: 2^0 - 2^1 Bank number 2^2 - 2^3 Section number 2^4 Byte: 0 Bits 2^0-2^7 1 Bits 2^8-2^{15}
2^1	I/O Processor B I/O Memory failing address, same format as for 2^0.
2^2	I/O Processor C I/O Memory failing address, same format as for 2^0.
2^3	Buffer Memory error data: 2^0 - 2^7 Syndrome bits 2^9 Correctable error if 1 2^{10} Noncorrectable error if 1 2^{12} - 2^{13} Port number 2^{15} Address invalid if 1
2^4	Central Memory error data: 2^0 - 2^7 Syndrome bits 2^9 Correctable error 2^{10} Noncorrectable error 2^{12} - 2^{13} Read mode: 0 Scalar 1 I/O 2 Vector 3 Fetch
2^5	Input Memory Channel A error data: 2^0 - 2^7 Syndrome bits 2^8 - 2^{10} Error code (refer to Central Memory input error processing, this section)
2^6	Output Memory Channel B error data, same as for 2^5. Refer to Central Memory output error processing, this section, for error codes.
2^7	Input Memory Channel C error data, same format as for 2^5.
2^8	Output Memory Channel D error data, same format as for 2^5. Refer to Central Memory output error processing, this section, for error codes.

This function must be immediately preceded by a logical product instruction as follows:

```
011777  A=A&d  d=777
172xxx  ERA : 12
        or
015xxx  A=A&k
xxx777  k=xxx777
172xxx  ERA : 12
```

ERA : 13 - READ ERROR INFORMATION (THIRD PARAMETER)

This function returns the content of the third parameter word register to the accumulator. The third parameter word is selected by bits 2^3 or 2^4 in the accumulator when the function issues. Only one bit may be set in the field at a time. Selection is as follows.

- 2^3 Buffer Memory upper address (address bits 2^{16} - 2^{22} going into accumulator positions 2^0 - 2^6)
- 2^4 Central Memory upper address (address bits 2^{16} - 2^{21} going into accumulator positions 2^0 - 2^5)

The reading of the third parameter word clears the error bits in the status register for the Central Memory or for the Buffer Memory.

This function must be immediately preceded by a logical product instruction as follows:

```
011777  A=A&d  d=777
173xxx  ERA : 13
        or
015xxx  A=A&k
xxx777  k=xxx777
173xxx  ERA : 13
```

BLOCK MULTIPLEXER CHANNEL

The I/O Subsystem communicates with IBM-compatible equipment over the block multiplexer channels. The following functions are used.

BMA : 0	Clear channel control
BMA : 1	Send reset function
BMA : 2	Channel command
BMA : 3	Read REQUEST-IN address
BMA : 4	Asynchronous I/O
BMA : 5	Delay counter diagnostic
BMA : 6§§	Clear channel interrupt enable flag
BMA : 7§§	Set channel interrupt enable flag
BMA : 10	Read I/O Memory address
BMA : 11	Read byte counter status
BMA : 12	Read status/address
BMA : 13	Read input tags
BMA : 14	Enter I/O Memory address
BMA : 15	Enter byte count
BMA : 16	Enter device address/mode
BMA : 17	Enter output tags

GENERAL CHARACTERISTICS

Block multiplexer channels are grouped into sets of four channels, which share one Cray Research BMC-4 Block Multiplexer Controller. The controller interfaces to the Auxiliary I/O Processor (XIOP) via one DMA port. The speed capacity of the DMA port is shared among the four channels of the controller. The four channels operate asynchronously and compete only for I/O Memory references.

---

§§ Allow 1 CP before checking the interrupt channel number (IOR : 10).

A block multiplexer channel operates in either selector channel mode, byte multiplexer mode or block multiplexer mode. All IBM commands are possible. The channel features command and data chaining, and detection of RETRY STATUS, as well as the I/O ERROR ALERT and the HIGH SPEED option.

The channel drives one or more IBM-compatible control units, which in turn drive peripheral devices.

This manual does not explain or document the IBM communications protocol, the signals used, nor their sequencing. Refer to the appropriate IBM publications for detailed information and definitions of IBM terminology.

TRANSFER RATES

The block multiplexer channel data rate is determined by cable length and signal turn-around time within the control unit. For each byte of data or control information that is sent, an appropriate response signal or signals must be received. This results in a data rate limit of about 6.4 million bits a second, or 800 kilobytes a second. IBM-compatible peripheral controllers that use the high speed option (requiring an additional pair of control lines) may achieve about 12.8 million bits a second, or 1.6 million bytes a second.

DATA HANDLING

One important feature of the channel is the assembly and disassembly of data between the 8-bit channel and the 16-bit memory parcels. Figure 7-12 shows the data changes.

The BMC-4 reads four 16-bit parcels from the I/O Memory and sends out eight 8-bit bytes to the block multiplexer channel. Four more 16-bit parcels are read into buffers in the controller while the channel is transmitting the first group.

The BMC-4 reads eight 8-bit bytes from the channel and writes four 16-bit parcels into I/O Memory. If a read operation from the channel terminates with a byte length that does not evenly comprise four parcels, the last I/O Memory write will include unpredictable data in the last parcels.

RECORD SIZE

Data records may be any non-zero integer number of bytes in length. However, the data chaining feature must be used for lengths greater than 65,535 bytes.

In cases such as IBM tape records, an odd length header field can be read and stored at one memory area and the following data record can be stored at a different memory area beginning at a Buffer Memory word boundary. The data chaining feature permits a single large record to be broken into any size convenient for storage in Buffer Memory. This is under program control by means of I/O functions to the block multiplexer channel.

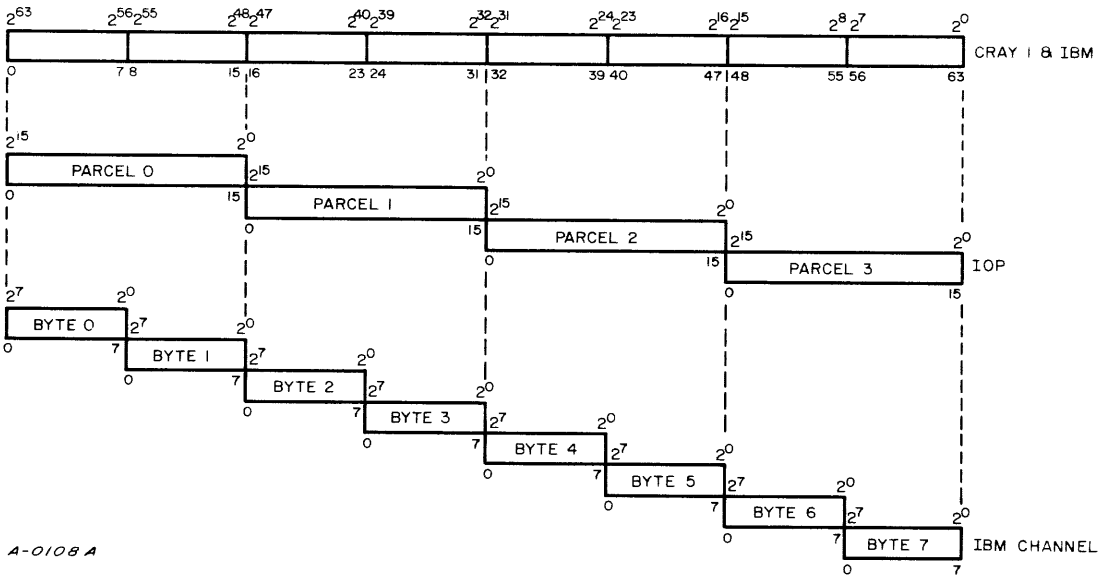


Figure 7-12. BMC-4 data assembly/disassembly

PARITY

Odd parity is checked on all address, status, and data inputs to the I/O Processor. Odd parity is generated for all address, control, or data outputs from the I/O Processor.

INTERRUPTS

All interrupts can be enabled or disabled for any block multiplexer channel. If interrupts are enabled, and a function in the range BMA : 1 through BMA : 5 completes, then an interrupt request is set. If an interrupt is selected for an input tag line such as REQUEST-IN, the interrupt request sets when the REQUEST-IN line goes to a logical 1.

When data chaining is enabled, an interrupt request is set each time the byte counter decrements to 0 and memory references are complete through that particular block of data.

BMA : 0 - CLEAR CHANNEL BUSY AND DONE FLAGS

This function clears the channel busy and done flags and all output tags except OP-OUT and SUP-OUT. No parameters are required for this action. It also clears interrupt conditions, provided interrupts are disabled by a BMA : 6 function. Since this function cannot be interlocked using the channel done flag, allow a 12-CP delay before issuing the next function to that particular channel.

BMA : 1 - SEND RESET FUNCTION

Several reset functions (table 7-19) perform various functions required by the equipment. Bits 2^1 and 2^0 of the accumulator content are used as a parameter that selects the specific function.

Table 7-19. Send reset function parameters

Parameter 2^1 - 2^0	Function
0 0	Clear all output tag lines
0 1	INTERFACE DISCONNECT
1 0	SELECTIVE RESET
1 1	SYSTEM RESET

Parameter xxxxx0 - Clear Output Tag Lines

This function clears all output tag lines and clears BUS 0 Out lines. The channel initially clears the channel done flag and sets the channel busy flag. Upon completion the channel done flag sets and the channel busy flag clears.

Parameter xxxxx1 - INTERFACE DISCONNECT

This function performs the INTERFACE DISCONNECT function to the currently selected control unit. The function clears the channel done flag and sets the channel busy flag. The control unit removes all signals from the Cray I/O Subsystem channel. When the control unit reaches the normal ending point in its sequence, it attempts to obtain selection in order to present any generated status to the Cray channel. The control unit does not generate any status as a result of the INTERFACE DISCONNECT.

The device path for the peripheral remains busy after it receives an INTERFACE DISCONNECT during an operation until the device-end status is accepted by the Cray block multiplexer channel.

The function should complete in about 7 microseconds. At that time the Cray channel done flag sets and the busy flag clears.

If the OPERATIONAL-IN signal from the equipment does not clear within 6 microseconds from the function issue, the Cray channel done flag sets and the busy flag remains set to indicate the error condition.

Parameter xxxxx2 - SELECTIVE RESET

This performs the SELECTIVE RESET function. The channel done flag clears and the channel busy flag sets. The function causes the OPERATIONAL-IN signal to clear and resets the currently selected peripheral device, along with its status. The current operation proceeds to a normal stopping point, and no data is transferred after the stop.

Only the peripheral device currently operating is affected. The device-end status is retained for transfer to the Cray channel after the reset.

The ready or not ready state of the control unit is generally not changed by the SELECTIVE RESET function.

The function should complete in about 7 microseconds, at which time the Cray channel busy flag clears and the done flag sets.

Parameter xxxxx3 - SYSTEM RESET

The SYSTEM RESET function clears the OPERATIONAL-IN signal and resets all control units along with their attached peripheral devices. The peripheral device statuses are also reset. The channel done flag clears and the channel busy flag sets at the beginning of execution.

The function should complete in about 6 microseconds at which time the Cray channel done flag sets and the busy flag clears. Input tag lines that are not reset by the SYSTEM RESET function cause the channel busy flag to remain set when the channel done flag sets.

BMA : 2 - CHANNEL COMMAND

This function sends commands to the control units. The specific command is selected by the accumulator bits at the time the function issues. Many of the commands require prior functions to set up interface registers, such as the I/O Memory address register, the byte count register, and the device address register. The command parameter bits are shown in table 7-20. The channel done flag clears and the channel busy flag sets at the beginning of execution. At completion of the function, the channel busy flag clears and the channel done flag sets.

Table 7-20. Channel command function parameter bits

Parameter Bit	Purpose
2^0	IBM-type control unit command bit 2^0
2^1	IBM-type control unit command bit 2^1
2^2	IBM-type control unit command bit 2^2
2^3	IBM-type control unit command bit 2^3
2^4	IBM-type control unit command bit 2^4
2^5	IBM-type control unit command bit 2^5
2^6	IBM-type control unit command bit 2^6
2^7	IBM-type control unit command bit 2^7
2^8	Unused
2^9	Unused
2^{10}	Unused
2^{11}	Unused
2^{12}	Unused
2^{13}	Unused
2^{14}	Unused
2^{15}	Unused

Parameter Command Bits

Bits 2^0 - 2^7 are the command bits for the BMA : 2 parameter. The command is for an IBM-type control unit. The parameter bits are shown in table 7-21.

Table 7-21. Channel command bit assignments

	P	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
TEST I/O	1	0	0	0	0	0	0	0	0
SENSE	P	M	M	M	M	0	1	0	0
READ BACKWARD	P	M	M	M	M	1	1	0	0
WRITE	P	M	M	M	M	M	M	0	1
READ	P	M	M	M	M	M	M	1	0
CONTROL	P	M	M	M	M	M	M	1	1

M = Modifier bit
P = Parity bit

The specific modifier codes and the particular mode set is dependent on the particular IBM-type control unit and the peripheral device used. The command byte is sent only during the initial selection sequence. All commands except TEST I/O may require a data transfer to satisfy the function.

All commands begin with an initial selection sequence which ends with either an ending status (channel-end or channel-end with device-end), a zero status, or an error status. To a command calling for transfer of status, control or data bytes, a zero status signifies that the transfer may begin.

A data-type transfer ends when the byte count decrements to 0 and no data chaining condition exists. The data-type transfer also ends when STATUS-IN is received in response to SERVICE-OUT or DATA-OUT.

The channel done flag is not set until processing of the control sequence is completed and data is transferred to or from I/O Memory.

Figure 7-13 shows a typical channel sequence for a read to the I/O Processor.

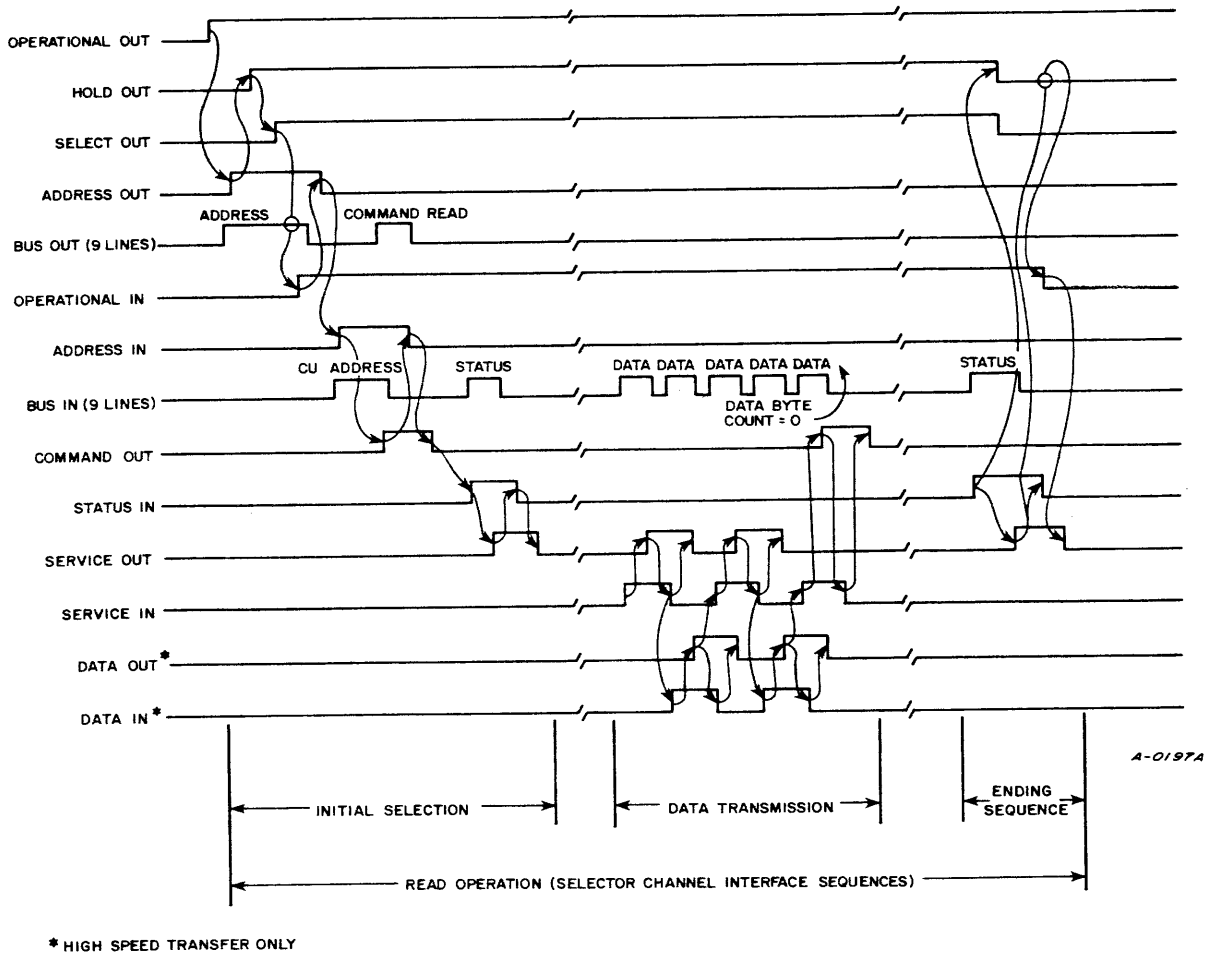


Figure 7-13. Channel read sequence

BMA : 3 - READ REQUEST-IN ADDRESS

This function first clears the channel done flag and sets the channel busy flag. If REQUEST-IN is detected, the channel accepts the requesting address. The address is checked for valid parity, the channel done flag is set, and the channel busy flag is cleared.

If REQUEST-IN is not detected, the channel done flag is set and the channel busy flag stays set. The address miscompare status is to be ignored if undetermined addresses are expected.

Figure 7-14 illustrates a typical REQUEST-IN channel sequence.

BMA : 4 - ASYNCHRONOUS I/O

Issuing the BMA : 4 function clears the channel done flag and sets the channel busy flag. If STATUS-IN is present, status is saved in the status register. If the STACK flag has been presented, COMMAND-OUT is returned in place of SERVICE-OUT to indicate STOP to the channel. When the sequence completes, the channel done flag sets and the channel busy flag clears. Refer to figure 7-14 for the typical REQUEST-IN channel sequence. See figure 7-15 for the asynchronous data and status processing. See table 7-26 for accumulator parameter bits.

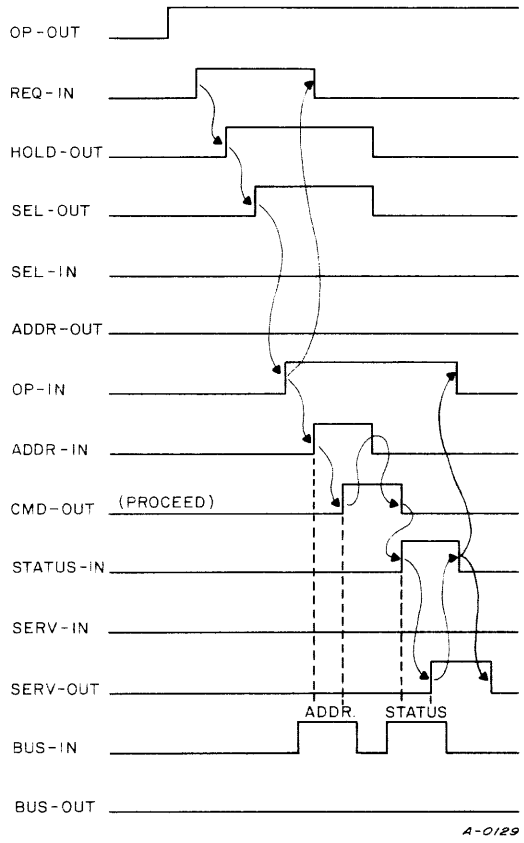


Figure 7-14. Channel ASYNCHRONOUS I/O sequence

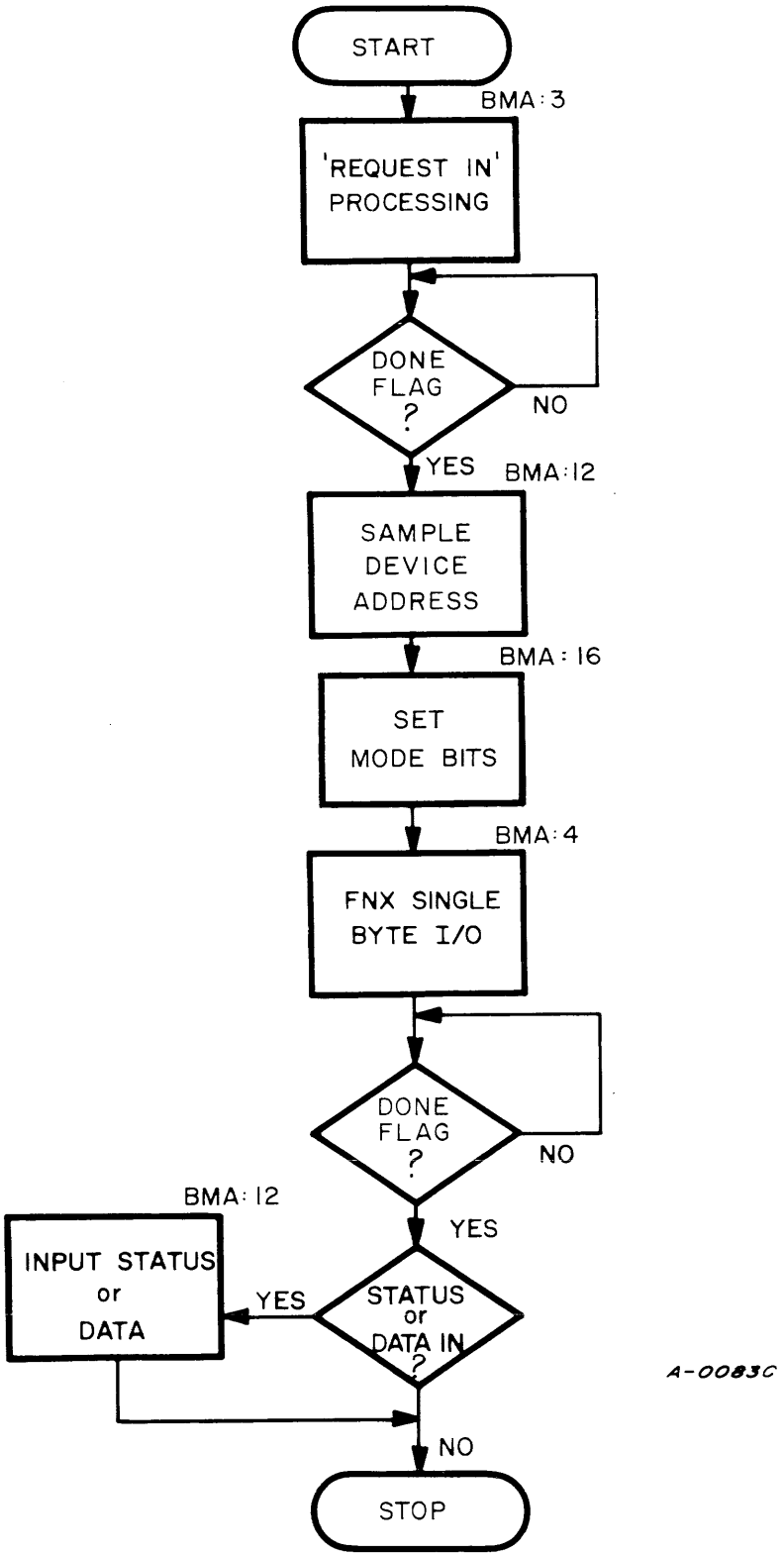


Figure 7-15. Asynchronous data and status processing

BMA : 5 - DELAY COUNTER DIAGNOSTIC

This function is for maintenance purposes only. It clears the channel done flag and sets the channel busy flag. Then the channel performs a 10 microsecond delay. After the delay times out, the channel done flag sets and the channel busy flag clears.

BMA : 6 - CLEAR CHANNEL INTERRUPT ENABLE FLAG

The BMA : 6 function clears the channel interrupt enable flag. This prevents the channel from interrupting the I/O Processor. The I/O Processor would in this case monitor the channel done flag to determine function completion. The channel busy and done flags are not affected by this function.

BMA : 7 - SET CHANNEL INTERRUPT ENABLE FLAG

This function sets the channel interrupt enable flag. This causes an I/O Processor interrupt for this channel whenever any of the following conditions occur:

- The channel done flag sets,
- The interrupt mode select, via a BMA : 16 function, has enabled an interrupt for an active input tag line, or
- During data chaining the byte counter has decremented to 0 and the last I/O Memory reference is complete for that segment of data.

BMA : 10 - READ I/O MEMORY ADDRESS

This function reads the current value in the I/O Memory address register and enters the value in the accumulator. The channel logic includes two I/O Memory address registers to support data chaining. The state of accumulator bit 2^0 when the function is issued determines which I/O Memory address register is to be read. Bit 2^0 of the value returned to the accumulator identifies from which register the I/O Memory address came.

Bit 2^1 in the value returned to the accumulator is set if data chaining is being used. Table 7-22 lists the accumulator content resulting from this function.

This function may be performed at any time relative to control functions.

During data chaining, interrupts occur after each buffer of data is transferred. This function clears the interrupt after a data transfer.

The BMA : 10 function has a programming restriction due to timing in the adder and shifter. The restriction applies if the instruction preceding a BMA : 10 function is any of the following: 4-7, 12, 13, 16, 17, 22, 23, 32, 33, 44-47, 52, 53, 62, or 63. In these cases an 011 or 015 logical product instruction, with the d or k fields set to all ones, should be inserted between the above instruction and the BMA : 10 function.

Table 7-22. Read I/O Memory address response bits

Accumulator Bit	Meaning
2^0	Register select status
2^1	Data chaining flag status (1=data chaining)
2^2	I/O Memory address 2^2
2^3	I/O Memory address 2^3
2^4	I/O Memory address 2^4
2^5	I/O Memory address 2^5
2^6	I/O Memory address 2^6
2^7	I/O Memory address 2^7
2^8	I/O Memory address 2^8
2^9	I/O Memory address 2^9
2^{10}	I/O Memory address 2^{10}
2^{11}	I/O Memory address 2^{11}
2^{12}	I/O Memory address 2^{12}
2^{13}	I/O Memory address 2^{13}
2^{14}	I/O Memory address 2^{14}
2^{15}	I/O Memory address 2^{15}

BMA : 11 - READ BYTE COUNTER

The byte counter records the number of bytes remaining in a data transfer. This counter is initially loaded with a value by a BMA : 15 load byte counter function. When the counter decrements byte-by-byte to 0, the channel interrupts the IOP and terminates the transfer. A transfer terminated by the control unit and not by decrementing to zero may result in a nonzero value in the counter. If data chaining is requested the byte counter is reloaded after decrementing to zero.

Due to the fixed timing in the IOP, this function must be executed twice in immediate succession to get a current value byte counter status to the IOP accumulator. The first execution moves the current byte counter status to the block multiplexer controller. The second execution moves the status to the IOP accumulator.

This function can be used to verify the accumulator fanout, the intermediate byte count status register in the block multiplexer controller, and the status path back to the IOP accumulator. Issuing a BMA : 15 function (explained later) loads the byte count into the byte counter status register in the block multiplexer controller. The next BMA : 11 function reads the byte count back from the status register to the IOP accumulator. The second BMA : 11 function performs as described above.

The channel busy and done flags are not affected by this function.

BMA : 12 - READ STATUS/ADDRESS

The status register holds the address and status mode bits read from the block multiplexer channel.

Due to the fixed timing in the IOP, this function must be executed twice in immediate succession to get valid current status/address information to the IOP accumulator. The first execution moves the current status/address to the block multiplexer controller. The second execution moves the status/address to the IOP accumulator.

This function can be used to verify the accumulator fanout, the intermediate status register in the block multiplexer controller, and the status path back to the IOP accumulator. Issuing a BMA : 16 function (explained later) loads address and mode bits into the block multiplexer controller status register. The next BMA : 12 function reads the status back to the IOP accumulator. The second BMA : 12 function performs as described above.

The channel busy and done flags are not affected by this function.

The status/address returned is the status/address information from the channel, and is entered in the IOP accumulator as shown in table 7-23.

Table 7-23. Status register bits

Accumulator Bits	Meaning
2^0	Status 2^0
2^1	Status 2^1
2^2	Status 2^2
2^3	Status 2^3
2^4	Status 2^4
2^5	Status 2^5
2^6	Status 2^6
2^7	Status 2^7
2^8	Address 2^0
2^9	Address 2^1
2^{10}	Address 2^2
2^{11}	Address 2^3
2^{12}	Address 2^4
2^{13}	Address 2^5
2^{14}	Address 2^6
2^{15}	Address 2^7

BMA : 13 - READ INPUT TAGS

This function reads the input tags from the channel to the I/O Processor accumulator.

Due to the fixed timing in the IOP, this function must be executed twice in immediate succession to get valid current input tags to the IOP accumulator. The first execution moves the current input tags to the block multiplexer controller. The second execution moves the input tags to the IOP accumulator.

This function can be used to verify the accumulator fanout, the intermediate output tags register, and the path back to the accumulator. Issuing a BMA : 17 function (explained later) loads the output tags into the output tags register. The next BMA : 13 function reads the stored output tags back to the IOP accumulator. The second BMA : 13 function brings the current input tags to the IOP accumulator.

The channel busy and done flags are not affected by this function.

The input tags status bits are shown in table 7-24.

Table 7-24. Input tags status bits

Accumulator Bits	Meaning
2^0	OPERATIONAL-IN
2^1	ADDRESS-IN
2^2	DISCONNECT-IN
2^3	SELECT-IN
2^4	REQUEST-IN
2^5	SERVICE-IN
2^6	DATA-IN
2^7	STATUS-IN
2^8	METERING-IN
2^9	MARK 0 IN
2^{10}	Unused - "0"
2^{11}	Data Buffer Pointer
2^{12}	Address Miscompare
2^{13}	Byte Count Zero
2^{14}	P-ROM Parity Error
2^{15}	BUS-IN Parity Error

BMA : 14 - ENTER I/O MEMORY ADDRESS

This function enters the current accumulator content into the I/O Memory address (IOMA) register. This address is the starting address in I/O Memory for the data transfer. The channel busy and done flags are not altered in the process. Two I/O Memory address registers are maintained for data chaining purposes, and are addressed by the 2^0 bit of the accumulator content. Data chaining must begin with register 0 and alternate between the 0 and 1 registers. Bit 2^1 of the accumulator content is the data chaining select flag for the chosen register and, if set, selects data chaining for that register/address.

Table 7-25 lists the accumulator bits for this function.

Table 7-25. I/O Memory address register bits

Accumulator Bits	Meaning
2^0	I/O Memory address register file select
2^1	Data chaining flag
2^2	I/O Memory address 2^2
2^3	I/O Memory address 2^3
2^4	I/O Memory address 2^4
2^5	I/O Memory address 2^5
2^6	I/O Memory address 2^6
2^7	I/O Memory address 2^7
2^8	I/O Memory address 2^8
2^9	I/O Memory address 2^9
2^{10}	I/O Memory address 2^{10}
2^{11}	I/O Memory address 2^{11}
2^{12}	I/O Memory address 2^{12}
2^{13}	I/O Memory address 2^{13}
2^{14}	I/O Memory address 2^{14}
2^{15}	I/O Memory address 2^{15}

BMA : 15 - ENTER BYTE COUNT

This function enters the accumulator content into the byte counter or into the next byte count register. The first BMA : 15 function following a BMA : 14 enters the accumulator content into the byte counter. Immediately following the first BMA : 15 function with a second BMA : 15 function enters the accumulator data into the next byte counter register. The transfer from the next byte count register to byte counter is done automatically between data segments in data chaining. Channel commands requiring no data, parameters, or status must establish a byte count of 0. The maximum count is 65,535 bytes. The channel busy and done flags are not altered in this process.

BMA : 16 - ENTER DEVICE ADDRESS/MODE

This function enters the accumulator content into the device address register. The device address register contains mode select bits and device address bits. The device address may be some combination of controller address bits and peripheral device address bits. The channel busy and done flags are not altered by this function. Table 7-26 shows the device address register bits.

Table 7-26. Device address register bits

Accumulator Bits	Meaning
2^0	Address/data out 2^0
2^1	Address/data out 2^1
2^2	Address/data out 2^2
2^3	Address/data out 2^3
2^4	Address/data out 2^4
2^5	Address/data out 2^5
2^6	Address/data out 2^6
2^7	Address/data out 2^7
2^8	Skip flag
2^9	Stack status flag
2^{10}	Command chaining mode select 2^0
2^{11}	Command chaining mode select 2^1
2^{12}	Interrupt mode select 2^0
2^{13}	Interrupt mode select 2^1
2^{14}	Channel mode select 2^0
2^{15}	Channel mode select 2^1

Parameter Mode Bits

Bits 2^8 - 2^{15} of the parameter are called mode bits. They are re-established during each BMA : 16 function. They operate as follows.

Skip flag - Bit 2^8 is the mode bit for the Skip flag. When set, it prohibits storing data into I/O Memory during read data transfers.

Stack status flag - Bit 2^9 is the mode bit for the stack status flag.

Command chaining mode select - Bits 2^{10} and 2^{11} select the mode in which command chaining will be used. Table 7-27 shows the translation of these mode bits.

Table 7-27. Command chaining mode selection

Parameter Bits 2^{11}-2^{10}	Selection
0 0	No chaining
0 1	Chain if channel-end status is detected
1 0	Chain if device-end status is detected
1 1	Chain if either channel-end status or device-end status is detected

Interrupt mode select - Parameter bits 2^{12} - 2^{13} select the mode in which interrupts are generated. Table 7-28 shows the translation of the interrupt mode bits.

Table 7-28. Interrupt mode selection

Parameter Bits 2^{13}-2^{12}	Selection
0 0	No interrupt mode selected, interrupts disabled
0 1	Interrupt on REQUEST-IN
1 0	Interrupt on STATUS-IN
1 1	Interrupt on DISCONNECT-IN

Channel type mode select - Parameter bits 2^{14} - 2^{15} are the mode bits that select which type of channel operation is to be used. The selection is shown in table 7-29.

Table 7-29. Channel type mode selection

Parameter Bits 2^{15}-2^{14}	Selection
0 0	Selector channel
0 1	Byte multiplexer channel
1 0	Block multiplexer channel
1 1	Reserved for future use

BMA : 17 - ENTER OUTPUT TAGS

This function enters the accumulator content into the output tags register, allowing direct program control of the output tags for special control sequences such as diagnostics. The channel busy and done flags are not altered by this function. Table 7-30 shows the accumulator bits for each output tag.

Table 7-30. Output tags register bits

Accumulator Bits	Meaning
2^0	OPERATIONAL-OUT
2^1	ADDRESS-OUT
2^2	HOLD-OUT
2^3	SELECT-OUT
2^4	COMMAND-OUT
2^5	SERVICE-OUT
2^6	DATA-OUT
2^7	SUPPRESS-OUT
2^8	METERING-OUT
2^9	MARK 0 OUT
2^{10}	Unused
2^{11}	Unused
2^{12}	CLOCK-OUT
$^{-13}$	Inhibit Parity Error
2^{14}	Force BUS-OUT Parity
2^{15}	Unused

PROGRAMMING EXAMPLES

The following examples illustrate two programming sequences for the block multiplexer channel. Example 1 shows the function sequence used to read a tape record of unknown length. Example 2 shows how to rewind the tape.

Example 1:

$Accumulator_8$	Function	Description
000000	BMA : 14	Set IOMA = 0 (arbitrary)
000000	BMA : 15	Set byte count = 0
000025	BMA : 16	Set device address (0-255)
000323	BMA : 2	Select GCR tape mode
	Wait for "Done"	
BUFA+2	BMA : 14	Set IOMA, chaining flag Buffer A
BUFB+3	BMA : 14	Set IOMA, chaining flag Buffer B
000005	BMA : 15	Set byte count Buffer A (header)
001000	BMA : 15	Set byte count Buffer B (1-65535)
000002	BMA : 2	Function read forward command
	Wait for interrupt	
BUFA+2	BMA : 14	Set IOMA, chaining flag, Buffer A
001000	BMA : 15	Set byte count Buffer A (1-65535)
	Wait for interrupt	Process header
BUFB+3	BMA : 14	Set IOMA, chaining flag Buffer B
001000	BMA : 15	Set byte count Buffer B
	Wait for interrupt	Transfer Buffer B to Buffer Memory
BUFA+2	BMA : 14	Set IOMA, chaining flag Buffer A
001000	BMA : 15	Set byte count Buffer A (1-65535)
	Wait for interrupt	Transfer Buffer A to Buffer Memory
	"Done" set	
	BMA : 11	
1000-X	BMA : 11	Read byte counter for residue transfer X bytes, Buffer B to Buffer Memory

Example 1 (continued):

$Accumulator_8$	Function	Description
	BMA : 12	
	BMA : 12	Verify device status
	BMA : 13	
	BMA : 13	Check input P.E. etc.

Example 2:

$Accumulator_8$	Function	Description
000000	BMA : 14	Set IOMA = 0 (arbitrary)
000000	BMA : 15	Set byte count = 0
000025	BMA : 16	Set device address (0-255)
000007	BMA : 2	REWIND function
	Wait for "Done"	
	Check for "Busy"	"Busy" set indicates error
	BMA : 12	
	BMA : 12	Verify device address and status
	BMA : 13	
	BMA : 13	Check for other errors

BUFFER MEMORY 8

INTRODUCTION

The Buffer Memory assists data transfer between peripheral devices and Central Memory. It is implemented as NMOS (negative channel metal oxide semiconductor) LSI storage circuits. Buffer Memory capacity ranges from 1 million to 8 million words of 64 bits each. The Buffer Memory is housed in the same chassis as the I/O Processors, in order to keep the intercabling delays small. Modules are of the same construction as the I/O Processor modules -- four printed circuit boards to each module. This section describes the speeds, organization, access, and addressing of the Buffer Memory.

MEMORY SPEEDS

The memory is a dynamic random access memory that refreshes its data once every 2 milliseconds. Refreshing is transparent and does not affect the random access capability. Access time is 200 nanoseconds. Memory cycle time is defined as the waiting time required after referencing a storage location before that location may be referred to again. The Buffer Memory cycle time is 40 CPs.

MEMORY ORGANIZATION

Two options are available for memory organization. The Buffer Memories with 1 to 4 million words use an 8-bank organization, and the 8-million word size uses the 16-bank phasing. The advantages of the 16-bank organization are that the number of bank conflicts (more than one reference to the same bank at the same time, causing one reference to wait) are greatly reduced and the memory bandwidth is doubled.

Each address in Buffer Memory contains 64 data bits and 8 bits of error correction information. Data is transferred to and from Buffer Memory in 16-bit parcels. Four parcels are stored in one 64-bit word. Parcels are received and sent in a 0,1,2,3 sequence. The packing arrangement is shown in figure 8-1.

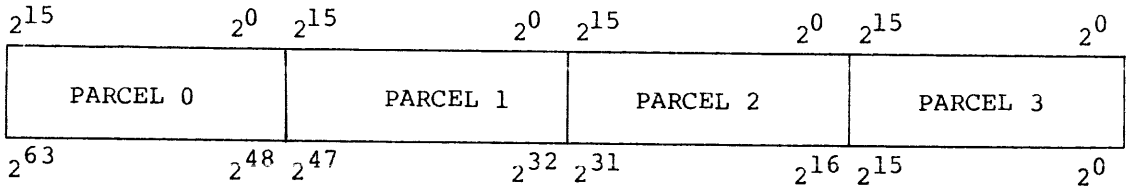


Figure 8-1. Parcel packing in memory word

MEMORY ACCESS

Buffer Memory has four ports, each of which can be connected to an I/O Processor, as shown in figure 8-2. An interface adapts one I/O Processor I/O Memory DMA port to one Buffer Memory port. Communication speed depends upon the number of banks in the Buffer Memory, the activity competing for the I/O Memory, and the number of other I/O Processors attempting to use the Buffer Memory. Each of the four Buffer Memory ports has access to all banks through a time-sharing scanner. If one port requests a reference to a bank that is busy, a reservation is made for the new port to gain access to the bank as soon as the bank becomes available. A similar scanning arrangement is used at the I/O Memory to share the six DMA ports.

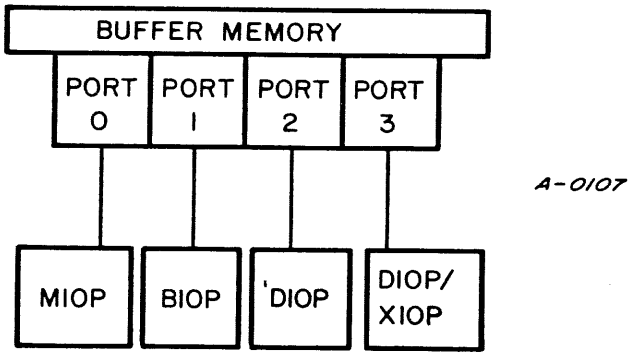


Figure 8-2. Buffer Memory port assignments

MEMORY ADDRESSING

The Buffer Memory capacity is either 1,048,576 or 4,194,304 or 8,388,608 words, requiring an address width of 23 bits. The 23-bit address is provided from an I/O Processor accumulator in two functions to the interface as shown in figure 8-3. The Buffer Memory does not address to the parcel level. The address represents a physical location in Buffer Memory.

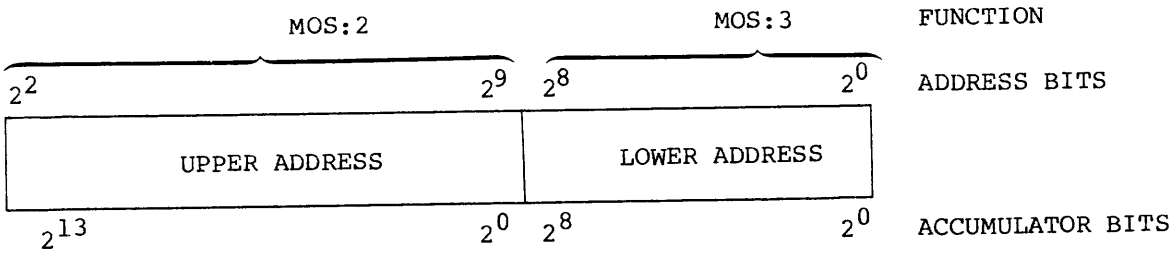


Figure 8-3. Buffer Memory address formation

ERROR PROTECTION

The Buffer Memory uses single error correction/double error detection (SECDED) logic to determine if the data has been altered by the storage cycle. When the data is written into memory, a checkword is generated for the word and stored with that word. The checkword is an 8-bit Hamming code. When the word is read from memory, the checkword and data word are processed to determine if any bits were altered. If no errors occurred, the word is passed to the I/O Processor.

If an error did occur, the 8 bits of the checkword are analyzed by the logic to find out if only 1 bit has been altered, or if more than 1 bit has changed. If it is only a single-bit error, the correction logic resets the bit in error to the correct state and passes the corrected word out to the I/O Processor.

If more than 1 bit of the stored word has been altered, the logic cannot correct the word. The interface signals the error condition by leaving the channel busy signal set after the block transfer. This interrupts the I/O Processor for an error handling routine. The error handling routine can include a call to the Master I/O Processor to have that processor retrieve the error information. This is explained in more detail in Channel for Error Logging in the Interfaces section.

The Buffer Memory uses the same SECDED logic design as used in the Central Memory. Refer to the Central Memory section for more details on error protection.

PART 4

APPENDIX SECTION

SUMMARY OF CPU TIMING INFORMATION A

When issue conditions are satisfied, an instruction completes in a fixed amount of time (memory references are exceptions). Instruction issue may cause reservations to be placed on a functional unit or registers. Knowledge of the issue conditions, instruction execution times and reservations permit accurate timing of code sequences. Memory bank conflicts due to I/O activity are the only element of unpredictability.

SCALAR INSTRUCTIONS

Four conditions must be satisfied for issue of a scalar instruction:

1. The functional unit must be free. No conflicts can arise with other scalar instructions; however, vector floating-point instructions reserve the floating-point units. Memory references may be delayed due to conflicts.

2. The result register must be free.

3. The operand register must be free.

4. One input path exists for each group of the four register groups (A, B, S, and T). The result register group input path must be free at the time the results would be stored. A previous instruction with a longer execution time could still be occupying the input path.

Scalar instructions place reservations only on result registers. A result register is reserved for the execution time of the instruction. No reservations are placed on the functional unit or operand registers.

Scalar instruction execution times in clock periods are given below.

where:
- A = A register
- B = B register
- C = Channel
- f = Floating-point
- I = Immediate
- lzc = Leading zero count
- M = Memory
- pop = Population count or population count parity
- RTC = Real-time clock
- ra = Reciprocal approximation
- S = S registers
- V = V registers
- VM = Vector mask

24-bit results:

A ← M	11§	A ← C	4
M ← A	1§	A ← A+A	2
A ← B	1	A ← AxA	6
B ← A	1	A ← pop(S)	4
A ← S	1	A ← lzc(S)	3
A ← I	1	VL ← A	1

64-bit results:

S ← M	11§	S ← S+S	3
M ← S	1§	S ← S(f.add)S	6§
S ← T	1	S ← S(f.mult)S	7§
T ← S	1	S ← S(r.a.)	14§
S ← I	1	S ← V	5
S ← S(log.)S	1	V ← S	3
S ← S(shift)I	2	S ← VM	1
S ← S(shift)A	3	S ← RTC	1
S ← S(mask)I	1	S ← A	2
RTC ← S	1	VM ← S	3

The following is an example of the use of this chart of execution times to optimize timing.

§ Issue may be delayed because of a functional unit reservation by a vector instruction. Memory may be considered a functional unit for timing considerations.

CAL Code			Execution Time	Reservations				
1	S1	S2+S3	3	S1				
2	A2	0 (immed.)	1	S1	A2			
3	S5	A2	2	S1		S5		
4	S4	S1+S3	3			S5	S4	
5	S6	S5&S1	1				S4	S6
6							S4	
7								

VECTOR INSTRUCTIONS

Four conditions must be satisfied for issue of a vector instruction:

1. The functional unit must be free. (Conflicts may occur with vector operations.)

2. The result register must be free. (Conflicts may occur with vector operations.)

3. The operand registers must be free or at chain slot time.

4. Memory must be quiet if the instruction references memory.

Vector instructions place reservations on functional units and registers for the duration of execution.

1. Functional units are reserved for (VL)+4 CPs. Memory is reserved for (VL)+5 CPs on a write operation, (VL)+4 CPs on a read operation.

2. The result register is reserved for the functional unit time +(VL) +2 CPs. The result register is reserved for the functional unit time +7 CPs if the vector length is less than 5. At functional unit time +2 (chain slot time) a subsequent vector instruction, which has met all other issue conditions, may issue. This process is called chaining. Several vector instructions using different functional units may be chained in this manner to attain a significant enhancement of processing speed.

3. Vector operand registers are reserved for (VL) CPs. Vector operand registers are reserved for 5 CPs if the vector length is less than 5. The vector register used in a block store to memory (177 instruction) is reserved for (VL) clock periods. Scalar operand registers are not reserved.

Vector instructions produce one result per clock period. The functional unit times are given below. The vector read and write instructions (176,177) produce results more slowly if bank conflicts arise due to the increment value (Ak) being a multiple of 8[§]. Chaining cannot occur for the vector read operation in this case.

If (Ak) is an odd multiple of 8[§], results are produced every 2 CPs.

If (Ak) is an even multiple of 8[§], results are produced every 4 CPs.

Functional unit	Time (CP)
Vector logical	2
Vector shift	4
Vector integer add	3
Floating-point add	6
Floating-point multiply	7
Reciprocal approximation	14
Memory	7
Vector population count	6

A transmit vector mask to Si (073) instruction is delayed by (VL) +6 CPs from the issue of a previous vector mask (175) instruction and is delayed by 6 CPs from the issue of preceding transmit (Sj) to VM (003) instruction.

HOLD ISSUE

A delay of issue results if a 100 - 137 instruction is the NIP register and a hold memory condition exists. The delay depends on the hold memory delay.

A delay of issue results if a 100 - 137 instruction is the NIP register and a 100 - 137 instruction in process senses a conflict with rank A, B, or C.

An additional 1 CP delay is added to a hold memory condition if a 070 instruction destination register conflict is sensed.

[§] Multiple of 4 for 8-bank phasing; refer to part 2 section 2.

Memory must be quiet before issue of the B and T register block copy instructions (034-037). The low-order 7 bits (Ai) affect the timing. Subsequent instructions may not issue for 14+(Ai) CPs if (Ai)≠0 and 5 CPs if (Ai)=0 when reading data to the B and T registers (034,036). They may not issue for 6+(Ai) CPs when storing data (035,037).

The B and T register block read (034,036) instructions require that there be no register reservation on the A and S registers, respectively, before issue.

Conditional branch instructions cannot issue until an A0 or S0 operand register has been free for 2 CPs. Fall-through in buffer requires 2 CPs. Branch-in-buffer requires 5 CPs. When an "out of buffer" condition occurs the execution time for a branch instruction is 14 CPs§.

A 2-parcel instruction takes 2 CPs to issue.

Instruction issue is delayed 2 CPs when the next instruction parcel is in a different instruction parcel buffer. Instruction issue is delayed 12 CPs§§ if the next instruction parcel is not in an instruction parcel buffer.

HOLD MEMORY

A delay of 1, 2, or 3 CPs will be added to an A, B, S or T register memory read if a bank conflict occurs with rank C, B, or A, respectively, of the memory access network. A conflict occurs if the address is in the same bank as the address in rank C, B, or A. Conflicts can occur only with scalar or I/O references. The scalar instruction senses the conflict condition at issue time + 1 CP. The scalar instruction address enters rank A of the memory access network at issue time + 1 CP. The scalar instruction address enters rank B at issue + 2 CPs. The scalar instruction address enters rank C at issue + 3 CPs.

§ 18 CPs for 8-bank phasing
§§ 16 CPs for 8-bank phasing

Scalar memory instruction timing (no conflict):

CP n	Issue, reserve Ai or Si register
CP n+1	Address rank A, sense conflict
CP n+2	Address rank B
CP n+3	Address rank C
.	
.	
.	
CP n+10	Clear register reservation
CP n+11	Complete

INTERRUPT TIMING

After a sensed interrupt condition, a minimum of 3 CPs + 2 parcel issues must occur before the interrupt is generated. During the first 3 CPs, if no hold issue conditions exist, instruction parcels may issue. At the end of the 3 CPs, the NIP register parcel is examined. If the NIP instruction is a 2-parcel instruction, 3 parcel issues occur before the interrupt. If the NIP instruction is a 1-parcel instruction, only 2 parcel issues occur before the interrupt.

PHYSICAL ORGANIZATION OF CPU B

MAINFRAME

The CPU mainframe is one of two sizes as shown in figure B-1. The logic chassis are arranged two in each column in an arc that is about 2.5 feet in radius. The larger mainframe with 12 columns extends 270° around the arc, while the 8-column version subtends 180° of the arc. The columns are about 6 1/2 feet tall. At the base of the columns, 1 1/2 feet high and extending outward about 2 1/2 feet, are cabinets for power supplies and cooling distribution systems.

Viewing the mainframe from the top, the upper chassis are labeled A through L (A through H in the 8-column version) proceeding counterclockwise. In the same manner, the lower chassis are named M through X (M through T in the 8-column version). The general chassis layout is shown in figure B-2. In the 8-column version, the I,J,K,L,U,V,W and X logic chassis are omitted.

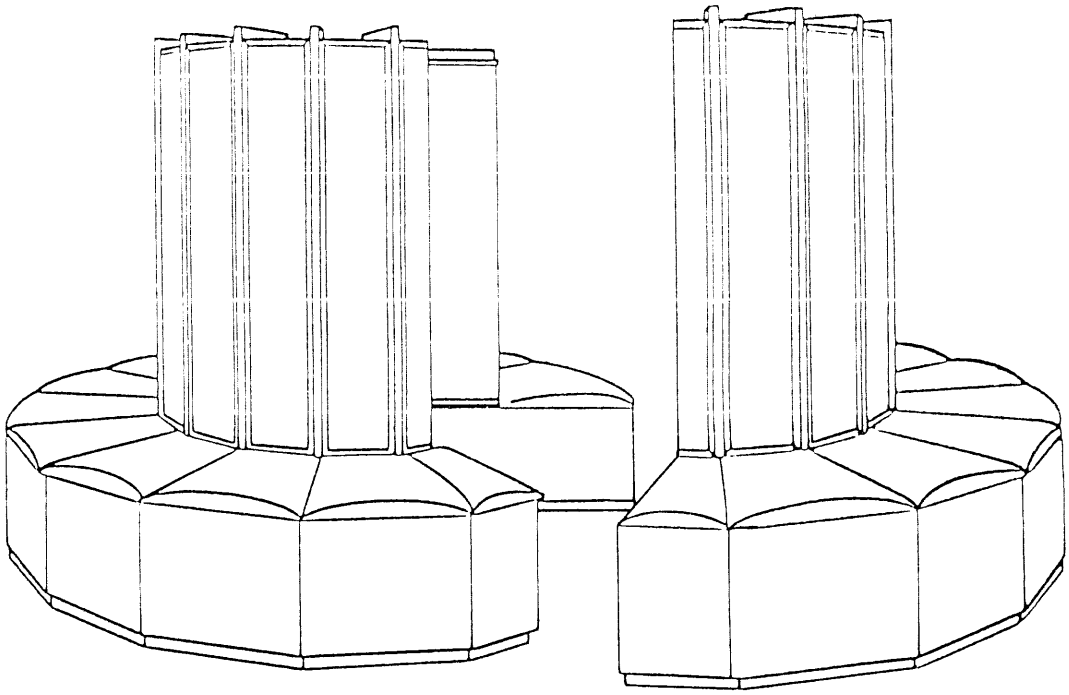
MODULES

The CRAY-1 computer system uses a basic module construction throughout the entire machine. The module consists of two 6 x 8 inch printed circuit boards mounted on opposite sides of a heavy copper heat transfer plate. Each printed circuit board has capacity for a maximum of 144 integrated circuit (IC) packages and approximately 300 resistor packages.

A 2- to 4-million word CPU has 1684 modules. Modules are arranged 72 per chassis as illustrated in figure B-2. There are over 131 module types. Usage varies from 1 to 568 modules per type. Each module type is identified by two letters. The first indicates the module series (A, D, F, G, H, J, M, R, S, T, V, and Z). The second letter identifies types of modules within a series.

The computation and I/O modules are on the eight chassis forming the center four columns. Each of the eight chassis on either side of the four center columns contains one of the memory banks.

Two supply voltages are used for each module: -5.2 volts for IC power; -2.0 volts for line termination.



- Dimensions

  Base - approximately 9 ft diameter by 1 1/2 ft high

  Columns - approximately 5 ft diameter by 6 1/2 ft high including height of base

- 24 chassis arranged two per column in 12 columns, or 16 chassis in 8 columns

- Approximately 1700 modules (12 column), 1100 for 8 column

- Approximately 130 standard module types

- Up to 288 IC packages per module

- Power consumption approximately 118 kW input for maximum memory size

- Refrigerant-22 cooled with refrigerant/water heat exchange

- Five memory options

- Weight 10,500 lbs (maximum memory size)

Figure B-1. Physical organization of CPU

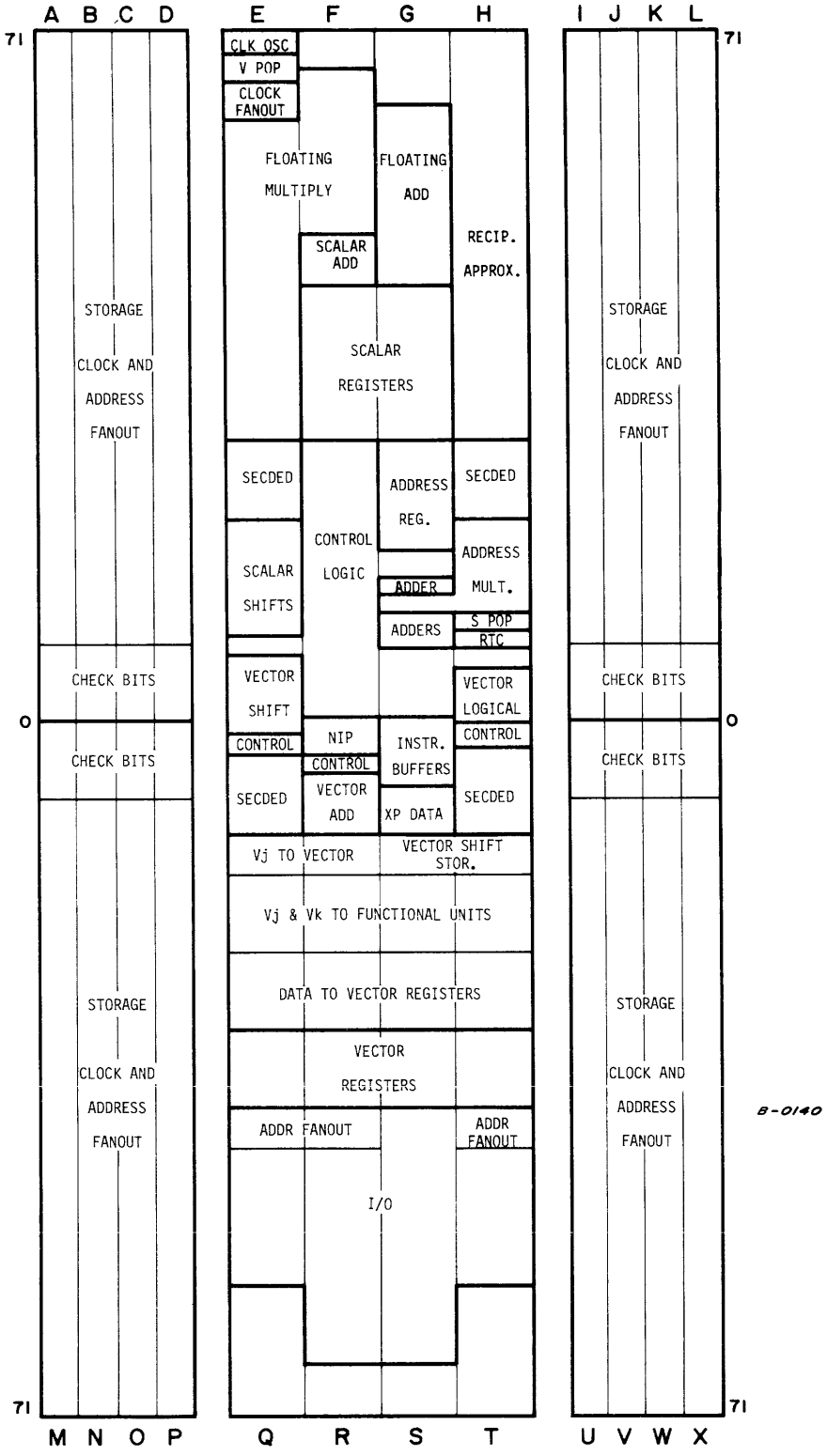


Figure B-2. General chassis layout

Each module has 96 pin pairs for interconnecting to other modules. All interconnections are via twisted pair wire. The average utilization of pins is approximately 60 percent.

Each module has 144 available test points used for trouble shooting. Test points are driven by circuits that do not drive other loads.

CLOCK

All timing within the CPU is controlled by a single-phase synchronous clock network. This clock has a period of 12.5 nanoseconds. All of the lines that carry the clock signal from the central clock source to the individual modules of the CPU are of uniform length so that the leading edge of a clock signal arrives at all parts of the CPU cabinet at the same time. A 3-nanosecond pulse (figure B-3) is formed on each module.

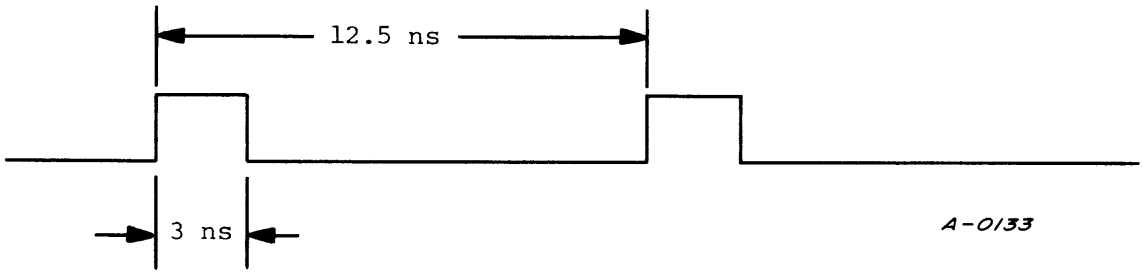


Figure B-3. Clock pulse waveform

References to clock periods in this manual are often given in the form CPn where n indicates the number of the clock period during which an event occurs. Clock periods are numbered beginning with CP 0. Thus, the third clock period would be referred to as CP 2.

POWER SUPPLIES

Thirty-six power supplies are used for the 12-column mainframe, or 24 for the 8-column models. There are twenty -5.2 volt supplies and sixteen -2.0 volt supplies (in the 12 column version). There are 12 of each voltage in the 8-column version. The supplies are divided into 12 groups (or 8) of 3, each group suppling one column. A logic column uses one -5.2 volt supply and two -2.0 volt supplies. A memory column uses two -5.2 volt supplies and one -2.0 volt supply. The power supply design assumes a constant load. The power supplies do

not have internal regulation but depend on the motor-generator to isolate and regulate incoming power. The power supplies use a 12-phase transformer, silicon diodes, balancing coil, and a filter choke to supply low ripple DC voltages. The entire supply is mounted on a refrigerant-22 cooled heat sink. Power is distributed via bus bars to the load.

COOLING

Modules in the CPU are cooled by the exchange of heat from the module heat sink to the refrigerated cold bars. The module heat sink is wedged along both 8-inch edges to the cold bars. Cold bars are arranged in vertical columns, with each column having capacity for 144 modules. The cold bar is cast aluminum and contains a stainless steel refrigerant tube.

SOFTWARE CONSIDERATIONS C

References to software in this publication are limited to those features of the CPU that provide for software or take it into consideration.

SYSTEM MONITOR

A monitor program is loaded at system deadstart and remains in Central Memory for as long as the system is used. Only the monitor program executes in CPU monitor mode and can execute monitor instructions. A program executing in monitor mode cannot be interrupted. A monitor program is designed to reference all of memory.

USER PROGRAM

A user program or object program, as referred to in this publication, means any program other than the monitor program. Generally, the term describes a job-oriented program but may also describe an operating system task that does not execute in monitor mode. A user program may be a machine language program such as a FORTRAN compiler or it may be a program resulting from compilation of FORTRAN statements by the compiler.

OPERATING SYSTEM

The operating system consists of a monitor program, object programs that perform system-related functions, compilers, assemblers, and various utility programs. The operating system is loaded into Central Memory and possibly onto mass storage during system deadstart. Features of the operating system and organization of storage, which is a function of the operating system, is described in CRAY-OS Version 1 Reference Manual, CRI publication SR-0011.

SYSTEM OPERATION

System operation begins at CPU deadstart. Deadstart is that sequence of operations required to start a program running in the computer after power has been turned off and then turned on again.

The deadstart sequence is initiated from the I/O Subsystem or the Maintenance Control Unit (MCU) depending on the model of the CRAY-1. The sequence is described in detail in part 2 section 3. During the deadstart sequence, a program containing an exchange package is loaded at absolute address 0 in the Central Memory. A signal from the MCU or I/O Subsystem causes the CRAY-1 to begin execution of the program pointed to by the exchange package.

FLOATING-POINT RANGE ERRORS

Detection of the floating-point range error initiates an interrupt if the floating-point mode flag is set in the mode register and monitor mode is not in effect. The programmer has the capability via the 0022 instruction to clear the floating-point mode flag so that results going out of range are prevented from interrupting. This is especially useful for the vector merge instruction usage in subroutines such as TANGENT, where some results may be known to go out-of-range. At the end of the code sequence, the programmer normally resets the floating-point mode via a 0021 instruction.

CPU INSTRUCTION SUMMARY

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
000xxx	ERR		6-7	-	Error exit
†000ijk	ERR	exp	6-7	-	Error exit
††0010jk	CA,Aj	Ak	6-8	-	Set the channel (Aj) current address to (Ak) and begin the I/O sequence
††0011jk	CL,Aj	Ak	6-8	-	Set the channel (Aj) limit address to (Ak)
††0012jx	CI,Aj		6-8	-	Clear channel (Aj) interrupt flag
††0013jx	XA	Aj	6-8	-	Enter XA register with (Aj)
††0014j0	RT	Sj	6-10	-	Enter RTC register with (Sj)
††§0014j4	PCI	Sj	6-10	-	Enter interval register with (Sj)
††§0014j5	CCI		6-10	-	Clear PCI request
††§0014j6	ECI		6-10	-	Enable PCI request
††§0014j7	DCI		6-10	-	Disable PCI request
0020xk	VL	Ak	6-12	-	Transmit (Ak) to VL register
†0020x0	VL	1	6-12	-	Transmit 1 to VL register
0021xx	EFI		6-13	-	Enable interrupt on floating point error
0022xx	DFI		6-13	-	Disable interrupt on floating point error
003xjx	VM	Sj	6-14	-	Transmit (Sj) to VM register
†003x0x	VM	0	6-14	-	Clear VM register
004xxx	EX		6-15	-	Normal exit
†004ijk	EX	exp	6-15	-	Normal exit
005xjk	J	Bjk	6-16	-	Jump to (Bjk)
006ijkm	J	exp	6-17	-	Jump to exp
007ijkm	R	exp	6-18	-	Return jump to exp; set B00 to P
010ijkm	JAZ	exp	6-19	-	Branch to exp if (A0) = 0
011ijkm	JAN	exp	6-19	-	Branch to exp if (A0) ≠ 0
012ijkm	JAP	exp	6-19	-	Branch to exp if (A0) positive
013ijkm	JAM	exp	6-19	-	Branch to exp if (A0) negative
014ijkm	JSZ	exp	6-21	-	Branch to exp if (S0) = 0
015ijkm	JSN	exp	6-21	-	Branch to exp if (S0) ≠ 0
016ijkm	JSP	exp	6-21	-	Branch to exp if (S0) positive
017ijkm	JSM	exp	6-21	-	Branch to exp if (S0) negative
020ijkm	Ai	exp	6-23	-	Transmit exp = jkm to Ai
021ijkm	Ai	exp	6-23	-	Transmit exp = 1's complement of jkm to Ai
022ijk	Ai	exp	6-24	-	Transmit exp = jk to Ai
023ijx	Ai	Sj	6-25	-	Transmit (Sj) to Ai
024ijk	Ai	Bjk	6-26	-	Transmit (Bjk) to Ai
025ijk	Bjk	Ai	6-26	-	Transmit (Ai) to Bjk
026ij0	Ai	PSj	6-27	Pop/LZ	Population count of (Sj) to Ai
026ij1	Ai	QSj	6-27	Pop/LZ	Population count parity of (Sj) to Ai
027ijx	Ai	ZSj	6-28	Pop/LZ	Leading zero count of (Sj) to Ai
030ijk	Ai	Aj+Ak	6-29	A Int Add	Integer sum of (Aj) and (Ak) to Ai
†030i0k	Ai	Ak	6-29	A Int Add	Transmit (Ak) to Ai
†030ij0	Ai	Aj+1	6-29	A Int Add	Integer sum of (Aj) and 1 to Ai
031ijk	Ai	Aj-Ak	6-29	A Int Add	Integer difference of (Aj) less (Ak) to Ai

† Special syntax form
†† Priviledged to monitor mode
§ Programmable Clock Option only

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
†031i00	Ai	-1	6-29	A Int Add	Transmit -1 to Ai
†031i0k	Ai	-Ak	6-29	A Int Add	Transmit the negative of (Ak) to Ai
†031ij0	Ai	Aj-1	6-29	A Int Add	Integer difference of (Aj) less 1 to Ai
032ijk	Ai	Aj*Ak	6-31	A Int Mult	Integer product of (Aj) and (Ak) to Ai
033i0x	Ai	CI	6-32	-	Channel number to Ai (j=0)
033ij0	Ai	CA,Aj	6-32	-	Address of channel (Aj) to Ai (j≠0; k=0)
033ij1	Ai	CE,Aj	6-32	-	Error flag of channel (Aj) to Ai (j≠0; k=1)
034ijk	Bjk,Ai	,A0	6-34	Memory	Read (Ai) words to B register jk from (A0)
†034ijk	Bjk,Ai	0,A0	6-34	Memory	Read (Ai) words to B register jk from (A0)
035ijk	,A0	Bjk,Ai	6-34	Memory	Store (Ai) words at B register jk to (A0)
†035ijk	0,A0	Bjk,Ai	6-34	Memory	Store (Ai) words at B register jk to (A0)
036ijk	Tjk,Ai	,A0	6-34	Memory	Read (Ai) words to T register jk from (A0)
†036ijk	Tjk,Ai	0,A0	6-34	Memory	Read (Ai) words to T register jk from (A0)
037ijk	,A0	Tjk,Ai	6-34	Memory	Store (Ai) words at T register jk to (A0)
†037ijk	0,A0	Tjk,Ai	6-34	Memory	Store (Ai) words at T register jk to (A0)
040ijkm	Si	exp	6-37	-	Transmit jkm to Si
041ijkm	Si	exp	6-37	-	Transmit exp = 1's complement of jkm to Si
042ijk	Si	<exp	6-38	S Logical	Form 1's mask exp = 64-jk bits in Si from the right
	Si	#>exp	6-38		
†042i77	Si	1	6-38	S Logical	Enter 1 into Si
†042i00	Si	-1	6-38	S Logical	Enter -1 into Si
043ijk	Si	>exp	6-38	S Logical	Form 1's mask exp = jk bits in Si from the left
	Si	#<exp			
†043i00	Si	0	6-38	S Logical	Clear Si
044ijk	Si	Sj&Sk	6-39	S Logical	Logical product of (Sj) and (Sk) to Si
†044ij0	Si	Sj&SB	6-39	S Logical	Sign bit of (Sj) to Si
†044ij0	Si	SB&Sj	6-39	S Logical	Sign bit of (Sj) to Si (j≠0)
045ijk	Si	#Sk&Sj	6-39	S Logical	Logical product of (Sj) and 1's complement of (Sk) to Si
†045ij0	Si	#SB&Sj	6-39	S Logical	(Sj) with sign bit cleared to Si
046ijk	Si	Sj\Sk	6-39	S Logical	Logical difference of (Sj) and (Sk) to Si
†046ij0	Si	Sj\SB	6-39	S Logical	Toggle sign bit of Sj, then enter into Si
†046ij0	Si	SB\Sj	6-39	S Logical	Toggle sign bit of Sj, then enter into Si (j≠0)
047ijk	Si	#Sj\Sk	6-39	S Logical	Logical equivalence of (Sk) and (Sj) to Si
†047i0k	Si	#Sk	6-39	S Logical	Transmit 1's complement of (Sk) to Si
†047ij0	Si	#Sj\SB	6-39	S Logical	Logical equivalence of (Sj) and sign bit to Si
†047ij0	Si	#SB\Sj	6-39	S Logical	Logical equivalence of (Sj) and sign bit to Si (j≠0)
†047i00	Si	#SB	6-39	S Logical	Enter 1's complement of sign bit into Si
050ijk	Si	Sj!Si&Sk	6-39	S Logical	Logical product of (Si) and (Sk) complement ORed with logical product of (Sj) and (Sk) to Si
†050ij0	Si	Sj!Si&SB	6-39	S Logical	Scalar merge of (Si) and sign bit of (Sj) to Si
051ijk	Si	Sj!Sk	6-39	S Logical	Logical sum of (Sj) and (Sk) to Si
†051i0k	Si	Sk	6-39	S Logical	Transmit (Sk) to Si
†051ij0	Si	Sj!SB	6-39	S Logical	Logical sum of (Sj) and sign bit to Si
†051ij0	Si	SB!Sj	6-39	S Logical	Logical sum of (Sj) and sign bit to Si (j≠0)
†051i00	Si	SB	6-39	S Logical	Enter sign bit into Si
052ijk	S0	Si<exp	6-43	S Shift	Shift (Si) left exp = jk places to S0
053ijk	S0	Si>exp	6-43	S Shift	Shift (Si) right exp = 64-jk places to S0
054ijk	Si	Si<exp	6-43	S Shift	Shift (Si) left exp = jk places
055ijk	Si	Si>exp	6-43	S Shift	Shift (Si) right exp = 64-jk places
056ijk	Si	Si,Sj<Ak	6-44	S Shift	Shift (Si and Sj) left (Ak) places to Si
†056ij0	Si	Si,Si<1	6-44	S Shift	Shift (Si and Sj) left one place to Si
†056i0k	Si	Si<Ak	6-44	S Shift	Shift (Si) left (Ak) places to Si

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
057ijk	Si	Sj,Si>Ak	6-44	S Shift	Shift (Sj and Si) right (Ak) places to Si
†057ij0	Si	Sj,Si>1	6-44	S Shift	Shift (Sj and Si) right one place to Si
†057i0k	Si	Si>Ak	6-44	S Shift	Shift (Si) right (Ak) places to Si
060ijk	Si	Sj+Sk	6-46	S Int Add	Integer sum of (Sj) and (Sk) to Si
061ijk	Si	Sj-Sk	6-46	S Int Add	Integer difference of (Sj) and (Sk) to Si
†061i0k	Si	-Sk	6-46	S Int Add	Transmit negative of (Sk) to Si
062ijk	Si	Sj+FSk	6-47	F.P. Add	Floating sum of (Sj) and (Sk) to Si
†062i0k	Si	+FSk	6-47	F.P. Add	Normalize (Sk) to Si
063ijk	Si	Sj-FSk	6-47	F.P. Add	Floating difference of (Sj) and (Sk) to Si
†063i0k	Si	-FSk	6-47	F.P. Add	Transmit normalized negative of (Sk) to Si
064ijk	Si	Sj*FSk	6-49	F.P. Mult	Floating product of (Sj) and (Sk) to Si
065ijk	Si	Sj*HSk	6-49	F.P. Mult	Half precision rounded floating product of (Sj) and (Sk) to Si
066ijk	Si	Sj*RSk	6-49	F.P. Mult	Full precision rounded floating product of (Sj) and (Sk) to Si
067ijk	Si	Sj*ISk	6-49	F.P. Mult	2 - Floating product of (Sj) and (Sk) to Si
070ijx	Si	/HSj	6-51	F.P. Rcpl	Floating reciprocal approximation of (Sj) to Si
071i0k	Si	Ak	6-52	-	Transmit (Ak) to Si with no sign extension
071i1k	Si	+Ak	6-52	-	Transmit (Ak) to Si with sign extension
071i2k	Si	+FAk	6-52	-	Transmit (Ak) to Si as unnormalized floating point number
071i3x	Si	0.6	6-52	-	Transmit constant 0.75*2**48 to Si
071i4x	Si	0.4	6-52	-	Transmit constant 0.5 to Si
071i5x	Si	1.	6-52	-	Transmit constant 1.0 to Si
071i6x	Si	2.	6-52	-	Transmit constant 2.0 to Si
071i7x	Si	4.	6-52	-	Transmit constant 4.0 to Si
072ixx	Si	RT	6-55	-	Transmit (RTC) to Si
073ixx	Si	VM	6-55	-	Transmit (VM) to Si
074ijk	Si	Tjk	6-55	-	Transmit (Tjk) to Si
075ijk	Tjk	Si	6-55	-	Transmit (Si) to Tjk
076ijk	Si	Vj,Ak	6-56	-	Transmit (Vj, element (Ak)) to Si
077ijk	Vi,Ak	Sj	6-56	-	Transmit (Sj) to Vi element (Ak)
†077i0k	Vi,Ak	0	6-56	-	Clear Vi element (Ak)
10hijkm	Ai	exp,Ah	6-57	Memory	Read from ((Ah) + exp) to Ai (A0=0)
†100ijkm	Ai	exp,0	6-57	Memory	Read from (exp) to Ai
†100ijkm	Ai	exp,	6-57	Memory	Read from (exp) to Ai
†10hi000	Ai	,Ah	6-57	Memory	Read from (Ah) to Ai
11hijkm	exp,Ah	Ai	6-57	Memory	Store (Ai) to (Ah) + exp (A0=0)
†110ijkm	exp,0	Ai	6-57	Memory	Store (Ai) to exp
†110ijkm	exp,	Ai	6-57	Memory	Store (Ai) to exp
†11hi000	,Ah	Ai	6-57	Memory	Store (Ai) to (Ah)
12hijkm	Si	exp,Ah	6-57	Memory	Read from ((Ah) + exp) to Si (A0=0)
†120ijkm	Si	exp,0	6-57	Memory	Read from (exp) to Si
†120ijkm	Si	exp,	6-57	Memory	Read from (exp) to Si
†12hi000	Si	,Ah	6-57	Memory	Read from (Ah) to Si
13hijkm	exp,Ah	Si	6-57	Memory	Store (Si) to (Ah) + exp (A0=0)
†130ijkm	exp,0	Si	6-57	Memory	Store (Si) to exp
†130ijkm	exp,	Si	6-57	Memory	Store (Si) to exp
†13hi000	,Ah	Si	6-57	Memory	Store (Si) to (Ah)
140ijk	Vi	Sj&Vk	6-59	V Logical	Logical products of (Sj) and (Vk) to Vi
141ijk	Vi	Vj&Vk	6-59	V Logical	Logical products of (Vj) and (Vk) to Vi
142ijk	Vi	Sj!Vk	6-59	V Logical	Logical sums of (Sj) and (Vk) to Vi
†142i0k	Vi	Vk	6-59	V Logical	Transmit (Vk) to Vi

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
143ijk	Vi	Vj!Vk	6-59	V Logical	Logical sums of (Vj) and (Vk) to Vi
144ijk	Vi	Sj\Vk	6-59	V Logical	Logical differences of (Sj) and (Vk) to Vi
145ijk	Vi	Vj\Vk	6-59	V Logical	Logical differences of (Vj) and (Vk) to Vi
†145iii	Vi	0	6-59	V Logical	Clear Vi
146ijk	Vi	Sj!Vk&VM	6-59	V Logical	Transmit (Sj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
†146i0k	Vi	#VM&Vk	6-59	V Logical	Vector merge of (Vk) and 0 to Vi
147ijk	Vi	Vj!Vk&VM	6-59	V Logical	Transmit (Vj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
150ijk	Vi	Vj<Ak	6-63	V Shift	Shift (Vj) left (Ak) places to Vi
†150ij0	Vi	Vj<1	6-63	V Shift	Shift (Vj) left one place to Vi
151ijk	Vi	Vj>Ak	6-63	V Shift	Shift (Vj) right (Ak) places to Vi
†151ij0	Vi	Vj>1	6-63	V Shift	Shift (Vj) right one place to Vi
152ijk	Vi	Vj,Vj<Ak	6-65	V Shift	Double shift (Vj) left (Ak) places to Vi
†152ij0	Vi	Vj,Vj<1	6-65	V Shift	Double shift (Vj) left one place to Vi
153ijk	Vi	Vj,Vj>Ak	6-65	V Shift	Double shift (Vj) right (Ak) places to Vi
†153ij0	Vi	Vj,Vj>1	6-65	V Shift	Double shift (Vj) right one place to Vi
154ijk	Vi	Sj+Vk	6-70	V Int Add	Integer sums of (Sj) and (Vk) to Vi
155ijk	Vi	Vj+Vk	6-70	V Int Add	Integer sums of (Vj) and (Vk) to Vi
156ijk	Vi	Sj-Vk	6-70	V Int Add	Integer differences of (Sj) and (Vk) to Vi
†156i0k	Vi	-Vk	6-70	V Int Add	Transmit negative of (Vk) to Vi
157ijk	Vi	Vj-Vk	6-70	V Int Add	Integer differences of (Vj) and (Vk) to Vi
160ijk	Vi	Sj*FVk	6-72	F.P. Mult	Floating products of (Sj) and (Vk) to Vi
161ijk	Vi	Vj*FVk	6-72	F.P. Mult	Floating products of (Vj) and (Vk) to Vi
162ijk	Vi	Sj*HVk	6-72	F.P. Mult	Half precision rounded floating products of (Sj) and (Vk) to Vi
163ijk	Vi	Vj*HVk	6-72	F.P. Mult	Half precision rounded floating products of (Vj) and (Vk) to Vi
164ijk	Vi	Sj*RVk	6-72	F.P. Mult	Rounded floating products of (Sj) and (Vk) to Vi
165ijk	Vi	Vj*RVk	6-72	F.P. Mult	Rounded floating products of (Vj) and (Vk) to Vi
166ijk	Vi	Sj*IVk	6-72	F.P. Mult	2 - floating products of (Sj) and (Vk) to Vi
167ijk	Vi	Vj*IVk	6-72	F.P. Mult	2 - floating products of (Vj) and (Vk) to Vi
170ijk	Vi	Sj+FVk	6-75	F.P. Add	Floating sums of (Sj) and (Vk) to Vi
†170i0k	Vi	+FVk	6-75	F.P. Add	Normalize (Vk) to Vi
171ijk	Vi	Vj+FVk	6-75	F.P. Add	Floating sums of (Vj) and (Vk) to Vi
172ijk	Vi	Sj-FVk	6-75	F.P. Add	Floating differences of (Sj) and (Vk) to Vi
†172i0k	Vi	-FVk	6-75	F.P. Add	Transmit normalized negatives of (Vk) to Vi
173ijk	Vi	Vj-FVk	6-75	F.P. Add	Floating differences of (Vj) and (Vk) to Vi
174ij0	Vi	/HVj	6-77	F.P. Rcpl	Floating reciprocal approximations of (Vj) to Vi
174ij1	Vi	PVj	6-78	F.P. Rcpl	Population counts of (Vj) to Vi
174ij2	Vi	QVj	6-78	F.P. Rcpl	Population count parities of (Vj) to Vi
175xj0	VM	Vj,Z	6-79	V Logical	VM=1 where (Vj) = 0
175xj1	VM	Vj,N	6-79	V Logical	VM=1 where (Vj) ≠ 0
175xj2	VM	Vj,P	6-79	V Logical	VM=1 where (Vj) positive
175xj3	VM	Vj,M	6-79	V Logical	VM=1 where (Vj) negative
176ixk	Vi	,A0,Ak	6-81	Memory	Read (VL) words to Vi from (A0) incremented by (Ak)
†176ix0	Vi	,A0,1	6-81	Memory	Read (VL) words to Vi from (A0) incremented by 1
177xjk	,A0,Ak	Vj	6-81	Memory	Store (VL) words from Vj to (A0) incremented by (Ak)
†177xj0	,A0,1	Vj	6-81	Memory	Store (VL) words from Vj to (A0) incremented by 1

† Special syntax form

I/O PROCESSOR INSTRUCTION SUMMARY E

IOP	APML	Description
000	PASS	No operation
001	EXIT	Exit from subroutine
002	I = 0	Disable system interrupts
003	I = 1	Enable system interrupts
004	A = A > d	Right shift C and A by d places, end off
005	A = A < d	Left shift C and A by d places, end off
006	A = A >> d	Right shift C and A by d places, circular
007	A = A << d	Left shift C and A by d places, circular
010	A = d	Transmit d to A
011	A = A & d	Logical product of A and d to A
012	A = A + d	Add d to A
013	A = A - d	Subtract d from A
014	A = k	Transmit k to A
015	A = A & k	Logical product of A and k to A
016	A = A + k	Add k to A
017	A = A - k	Subtract k from A
020	A = dd	Transmit operand register d to A
021	A = A & dd	Logical product of A and operand register d to A
022	A = A + dd	Add operand register d to A
023	A = A - dd	Subtract operand register d from A
024	dd = A	Transmit A to register d
025	dd = A + dd	Add operand register d to A, result to operand register d
026	dd = dd + 1	Transmit register d to A, add 1, result to operand register d
027	dd = dd - 1	Transmit register d to A, subtract 1, result to operand register d
030	A = (dd)	Transmit contents of memory addressed by register d to A
031	A = A & (dd)	Logical product of A and contents of memory addressed by register d, result to A
032	A = A + (dd)	Add contents of memory addressed by register d to A, result to A
033	A = A - (dd)	Subtract contents of memory addressed by register d from A, result to A

IOP	APML	Description
034	(dd) = A	Transmit A to memory addressed by register d
035	(dd) = A + (dd)	Add memory addressed by register d to A, result to same memory location
036	(dd) = (dd) + 1	Transmit memory addressed by register d to A, add 1, result to same memory location
037	(dd) = (dd) - 1	Transmit memory addressed by register d to A, subtract 1, result to same memory location
040	C = 1, iod = DN	Set carry equal to channel d done
041	C = 1, iod = BZ	Set carry equal to channel d busy
042	C = 1, IOB = DN	Set carry equal to channel B done
043	C = 1, IOB = BZ	Set carry equal to channel B busy
044	A = A > B	Right shift C and A by B places, end off
045	A = A < B	Left shift C and A by B places, end off
046	A = A >> B	Right shift C and A by B places, circular
047	A = A << B	Left shift C and A by B places, circular
050	A = B	Transmit B to A
051	A = A & B	Logical product of A and B to A
052	A = A + B	Add B to A, result to A
053	A = A - B	Subtract B from A, result to A
054	B = A	Transmit A to B
055	B = A + B	Add B to A, result to B
056	B = B + 1	Transmit B to A, add 1, result to B
057	B = B - 1	Transmit B to A, subtract 1, result to B
060	A = (B)	Transmit operand register B to A
061	A = A & (B)	Logical product of A and operand register B to A
062	A = A + (B)	Add operand register B to A, result to A
063	A = A - (B)	Subtract operand register B from A, result to A
064	(B) = A	Transmit A to operand register B
065	(B) = A + (B)	Add operand register B to A, result to operand register B
066	(B) = (B) + 1	Transmit operand register B to A, add 1, result to operand register B
067	(B) = (B) - 1	Transmit operand register B to A, subtract 1, result to operand register B
070	P = P + d	Jump to P + d
071	P = P - d	Jump to P - d
072	P = P + d	Return jump to P + d
073	P = P - d	Return jump to P - d
074	P = dd	Jump to address in operand register d
075	P = dd + k	Jump to sum of k and operand register d
076	R = dd	Return jump to address in operand register d
077	R = dd + k	Return jump to sum of k and operand register d

IOP	APML	Description
100	P = P + d, C = 0	Jump to P + d if carry = 0
101	P = P + d, C # 0	Jump to P + d if carry ≠ 0
102	P = P + d, A = 0	Jump to P + d if A = 0
103	P = P + d, A # 0	Jump to P + d if A ≠ 0
104	P = P - d, C = 0	Jump to P - d if carry = 0
105	P = P - d, C # 0	Jump to P - d if carry ≠ 0
106	P = P - d, A = 0	Jump to P - d if A = 0
107	P = P - d, A # 0	Jump to P - d if A ≠ 0
110	R = P + d, C = 0	Return jump to P + d if carry = 0
111	R = P + d, C # 0	Return jump to P + d if carry ≠ 0
112	R = P + d, A = 0	Return jump to P + d if A = 0
113	R = P + d, A # 0	Return jump to P + d if A ≠ 0
114	R = P - d, C = 0	Return jump to P - d if carry = 0
115	R = P - d, C # 0	Return jump to P - d if carry ≠ 0
116	R = P - d, A = 0	Return jump to P - d if A = 0
117	R = P - d, A # 0	Return jump to P - d if A ≠ 0
120	P = dd, C = 0	Jump to address in operand register d if carry = 0
121	P = dd, C # 0	Jump to address in operand register d if carry ≠ 0
122	P = dd, A = 0	Jump to address in operand register d if A = 0
123	P = dd, A # 0	Jump to address in operand register d if A ≠ 0
124	P = dd + k, C = 0	Jump to address in operand register d + k if carry = 0
125	P = dd + k, C # 0	Jump to address in operand register d + k if carry ≠ 0
126	P = dd + k, A = 0	Jump to address in operand register d + k if A = 0
127	P = dd + k, A # 0	Jump to address in operand register d + k if A ≠ 0
130	R = dd, C = 0	Return jump to address in operand register d if carry = 0
131	R = dd, C # 0	Return jump to address in operand register d if carry ≠ 0
132	R = dd, A = 0	Return jump to address in operand register d if A = 0
133	R = dd, A # 0	Return jump to address in operand register d if A ≠ 0

IOP	APML	Description
134	R = dd + k, C = 0	Return jump to address in operand register d + k if carry = 0
135	R = dd + k, C # 0	Return jump to address in operand register d + k if carry ≠ 0
136	R = dd + k, A = 0	Return jump to address in operand register d + k if A = 0
137	R = dd + k, A # 0	Return jump to address in operand register d + k if A ≠ 0
140	iod : 0	Channel d function 0
141	iod : 1	Channel d function 1
142	iod : 2	Channel d function 2
143	iod : 3	Channel d function 3
144	iod : 4	Channel d function 4
145	iod : 5	Channel d function 5
146	iod : 6	Channel d function 6
147	iod : 7	Channel d function 7
150	iod : 10	Channel d function 10
151	iod : 11	Channel d function 11
152	iod : 12	Channel d function 12
153	iod : 13	Channel d function 13
154	iod : 14	Channel d function 14
155	iod : 15	Channel d function 15
156	iod : 16	Channel d function 16
157	iod : 17	Channel d function 17
160	IOB : 0	Channel B function 0
161	IOB : 1	Channel B function 1
162	IOB : 2	Channel B function 2
163	IOB : 3	Channel B function 3
164	IOB : 4	Channel B function 4
165	IOB : 5	Channel B function 5
166	IOB : 6	Channel B function 6
167	IOB : 7	Channel B function 7
170	IOB : 10	Channel B function 10
171	IOB : 11	Channel B function 11
172	IOB : 12	Channel B function 12
173	IOB : 13	Channel B function 13
174	IOB : 14	Channel B function 14
175	IOB : 15	Channel B function 15
176	IOB : 16	Channel B function 16
177	IOB : 17	Channel B function 17

SYSTEM CHANNEL ASSIGNMENTS

F

The channel assignments for a typical Model 4400 system are shown in table F-1.

Table F-1. Typical Model 4400 system channel assignments

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Master I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Buffer I/O Processor
	7	AOA	Output to Buffer I/O Processor
	10	AIB	Input from Disk I/O Processor
	11	AOB	Output to Disk I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14		
	15		
	16	ERA	Error log
	17	EXB	Peripheral Expander (DMA 0)
	20	CIA	Input from CRAY-1 channel (DMA 1)
	21	COA	Output to CRAY-1 channel (DMA 1)
	22		
	23		
	24	CIB	Input from F.-E. Interface (DMA 2)
	25	COC	Output to F.-E. Interface (DMA 2)
	26		
	27		
	30	CIC	Input from F.-E. Interface (DMA 4)
	31	COC	Output to F.-E. Interface (DMA 4)
	32		
	33		
	34	CID	Input from F.-E. Interface (DMA 5)
	35	COD	Output to F.-E. Interface (DMA 5)
	36		
	37		
	40	TIA	Console 0 keyboard
	41	TOA	Console 0 display

Table F-1. Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Master I/O Processor (continued)	42	TIB	Console 1 keyboard
	43	TOB	Console 1 display
	44		
	45		
	46		
	47		
Buffer I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output from Master I/O Processor
	10	AIB	Input from Disk I/O Processor
	11	AOB	Output to Disk I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14	HIA	Input from Memory Channel (DMA 4)
	15	HOA	Output to Memory Channel (DMA 4)
	16		
	17		
	20	DKA	Disk Storage Unit 0 (DMA 0)
	21	DKB	Disk Storage Unit 1 (DMA 0)
	22	DKC	Disk Storage Unit 2 (DMA 1)
	23	DKD	Disk Storage Unit 3 (DMA 1)
	24	DKE	Disk Storage Unit 4 (DMA 2)
	25	DKF	Disk Storage Unit 5 (DMA 2)
	26	DKG	Disk Storage Unit 6 (DMA 5)
	27	DKH	Disk Storage Unit 7 (DMA 5)
	30		
	31		
	32		
	33		
	34		
	35		
	36		
	37		
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

Table F-1. Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Disk I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXR	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output to Master I/O Processor
	10	AIB	Input from Buffer I/O Processor
	11	AOB	Output to Buffer I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14		
	15		
	16		
	17		
	20	DKA	Disk Storage Unit 0 (DMA 0)
	21	DKB	Disk Storage Unit 1 (DMA 0)
	22	DKC	Disk Storage Unit 2 (DMA 0)
	23	DKD	Disk Storage Unit 3 (DMA 0)
	24	DKE	Disk Storage Unit 4 (DMA 1)
	25	DKF	Disk Storage Unit 5 (DMA 1)
	26	DKG	Disk Storage Unit 6 (DMA 1)
	27	DKH	Disk Storage Unit 7 (DMA 1)
	30	DKI	Disk Storage Unit 8 (DMA 2)
	31	DKJ	Disk Storage Unit 9 (DMA 2)
	32	DKK	Disk Storage Unit 10 (DMA 2)
	33	DKL	Disk Storage Unit 11 (DMA 2)
	34	DKM	Disk Storage Unit 12 (DMA 5)
	35	DKN	Disk Storage Unit 13 (DMA 5)
	36	DKO	Disk Storage Unit 14 (DMA 5)
	37	DKP	Disk Storage Unit 15 (DMA 5)
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

Table F-1. Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Auxiliary I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output to Master I/O Processor
	10	AIB	Input from Buffer I/O Processor
	11	AOB	Output to Buffer I/O Processor
	12	AIC	Input from Disk I/O Processor
	13	AOC	Output to Disk I/O Processor
	14		
	15		
	16		
	17		
	20	BMA	Block Multiplexer Channel 0 (DMA 0)
	21	BMB	Block Multiplexer Channel 1 (DMA 0)
	22	BMC	Block Multiplexer Channel 2 (DMA 0)
	23	BMD	Block Multiplexer Channel 3 (DMA 0)
	24	BME	Block Multiplexer Channel 4 (DMA 1)
	25	BMF	Block Multiplexer Channel 5 (DMA 1)
	26	BMG	Block Multiplexer Channel 6 (DMA 1)
	27	BMH	Block Multiplexer Channel 7 (DMA 1)
	30	BMI	Block Multiplexer Channel 8 (DMA 2)
	31	BMJ	Block Multiplexer Channel 9 (DMA 2)
	32	BMK	Block Multiplexer Channel 10 (DMA 2)
	33	BML	Block Multiplexer Channel 11 (DMA 2)
	34	BMM	Block Multiplexer Channel 12 (DMA 5)
	35	BMN	Block Multiplexer Channel 13 (DMA 5)
	36	BMO	Block Multiplexer Channel 14 (DMA 5)
	37	BMP	Block Multiplexer Channel 15 (DMA 5)
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

IOP PROGRAMMING CONSIDERATIONS G

Several special cases must be considered when programming the I/O Processor. These cases are explained below.

EXIT STACK TIMING

When issuing PXS : 4 or PXS : 15, allow 4 CPs[§] before enabling system interrupts, return jumps or exit instructions. PXS : 14 and PXS : 15 instructions should only be used when system interrupts are disabled.

EXIT STACK INTERRUPT HANDLING

If return jumps are used in an interrupt handler, verifiy that enough levels are left available in the stack. An interrupt with the exit stack pointer at 13_{10} causes the pointer to go to 14_{10} and leave only one location open. A worse case exists if a return jump which causes a program fetch request (PFR) interrupt is issued with the stack pointer at 13_{10}. The return address goes in 14_{10} and the interrupt address goes into 15_{10}. This condition now leaves two interrupts present--both the exit stack boundary and PFR, with the PFR being the highest priority and no stack locations available. If the stack pointer is allowed to increment from 15_{10}, it clears to 0, and incorrect return addresses are used.

SYSTEM INTERRUPT ENABLE

When issuing an I=1, the system interrupt enable is delayed until the next non-branch or non-I/O instruction is issued. The instructions that do not enable interrupts are the 40-43 and 70-137.

---

§ The term clock periods refers to processor instruction times taken up by issuing pass instructions, or some other instruction or group of instructions whose execution time equals or exceeds the delays noted. Any instruction or group of instructions may be used as long as they are not included in any of the special cases stated in this list.

This allows the executive/monitor to get back to the interruptible activity before an interrupt is accepted.

Use an I=0 instruction at the interrupt handler entrance. If a redundant I=1 is executed and an interrupt occurs before a non-branch or non-I/O instruction is encountered, the interrupt handler is entered (with interrupts disabled). But interrupts are re-enabled when the first non-branch or non-I/O instruction is issued within the interrupt handler.

SYSTEM INTERRUPT DISABLE

The instructions following an I=0 instruction may be skipped if an interrupt occurs (while I=0 is executing). Hence, a pass instruction should follow every I=0.

SYSTEM INTERRUPT CLEARED OR SET BY THE ENABLES FOR INDIVIDUAL CHANNELS

After issuing a command 6 or 7 to any I/O channel, allow 3 CPs[§] before seeing its effect on system interrupt. (Assuming system interrupts are, or will be, enabled.)

I/O CHANNEL TIMING

When issuing any command to the I/O channels, allow 1 CP[§] before checking busy or done status.

Also allow 1 CP[§] after any command 6 or 7 before checking for interrupt number (IOR : 10).

BUFFER MEMORY ERRORS

If a Buffer Memory multiple bit error has occurred, an MOS : 0 instruction is required prior to the next read or write command. The interface operation waits indefinitely on the error if it is not cleared by the MOS : 0.

---

§ The term clock periods refers to processor instruction times taken up by issuing pass instructions, or some other instruction or group of instructions whose execution time equals or exceeds the delays noted. Any instruction or group of instructions may be used as long as they are not included in any of the special cases stated in this list.

BUFFER MEMORY DEADSTART TIME

Deadstarting a processor via Buffer Memory requires apporoximately 2 ms to transfer the full 64K into I/O memory. This time assumes no other Buffer Memory activity other than refresh.

ERROR LOGGING AND BLOCK MULTIPLEXER CHANNELS

The commands : 10-13 to the error logging channel or the block multiplexer channel decode the present accumulator data on the interface. If the IOP instruction previous to the commands : 10-13 is any of the following: 4-7, 12, 13, 16, 17, 22, 23, 32, 33, 44-47, 52, 53, 62, 63; then an 11 or 15 instruction (with the d or k field set to all ones) should be inserted between the instruction and the interface command. This avoids a 1-CP timing restriction caused by the adder and shifter.

I/O INSTRUCTIONS AFTER DEADSTART

The first instruction executed after a deadstart cannot be an I/O instruction. This includes 40-43, 140-147, 154-157, 160-167, 174-177. The accumulator must be loaded before executing any of these instructions. This avoids a special control sequence condition after deadstart.

NOTE

150-153, 170-173 instructions may be executed after deadstart as the instructions do load the accumulator.

PERIPHERAL EXPANDER CHANNEL TRANSFERS

The expander channel supports block transfers to only the first $100{,}000_8$ parcels of I/O memory.

LIST OF ABBREVIATIONS

A, An	CPU address register n, n = 0 to 7 (CPU); IOP accumulator
Addr	Address
Adv.	Advance
Ai,Aj,Ak	Address register specified by instruction i,j,k fields
APML	A Programming Machine Language
B, Bn	CPU intermediate address register n, n = 0 to 77_8; IOP B register
BA	Bank address, buffer address
BIOP	Buffer I/O Processor
Bjk	Buffer register specified by instruction j,k fields
BM	Buffer Memory
CA	Current address register
CAL	Cray Assembly Language
Ch	Channel
CL	Channel Limit register
CIP	Current Instruction Parcel register
CLK	Clock
Contr.	Control
CP	Clock period, central processor
CPU	Central processing unit
CRI	Cray Research, Incorporated
DCU	Disk controller unit
Distr.	Distribution
DIOP	Disk I/O Processor
DMA	Direct memory access
DP	Destination Pointer register
DSU	Disk Storage Unit
Exch.	Exchange
F	Flag register (exchange package); IOP instruction function field

F.E.	Front end
F.P.	Floating-point
F.U.	Functional Unit
FWA	First word address
gh	g and h fields, CPU instruction operation code
GR	Group
h	h field, CPU instruction
Hz	Hertz, cycles per second
i	i field of CPU instruction
II	Interrupt Interval register (CPU); Instruction Issue register (IOP)
IC	Integrated Circuit
ICD	Interrupt Countdown counter
I/O	Input/Output
IOP	I/O Processor
IOR	I/O request
k	Kilo, 1024, k field CPU instruction
jk	j and k fields, CPU instruction
LA	Limit address
LIP	Last instruction parcel
LSI	Large scale integration
LWA	Last word address
M	Million; mode bit field in exchange package; instruction field
MBits	Megabits or million bits
Mbyte	Megabyte or million bytes
MCU	Maintenance Control Unit
MG	Motor-Generator
MHz	MegaHertz, or million cycles per second
MIOP	Master I/O Processor
MMI	Monitor Mode Interrupt
MOS	Metal oxide semiconductor
MSEC	Millisecond
MS	Mass storage
MSKO	Mask out
NIP	Next instruction parcel

nmos	Negative channel metal oxide semiconductor
ns	Nanosecond
OS	Operating system
Osc	Oscillator
P	Program address register; Program parcel counter
PDU	Power Distribution Unit
PFR	Program fetch request
POP	Population count
PCI	Programmable clock interrupt
R	Request; response
RA	Read address
Recip	Reciprocal
Reg.	Register
Req.	Request
Resp.	Response
Ref.	Reference
RP	Register Pointer register
R'RAB	R' = high-order bits of read address, RA = Read Address, B = Bank low-order bits of address in exchange package
RTC	Real-time clock
S	Scalar
s	Second
Seq.	Sequence
Sn	Scalar register, n = 0 to 7
Si, Sj, Sk	Scalar register specified by instruction i or j or k field
SECDED	Single error correction/double error detection
Stor.	Storage
T, Tn	Intermediate scalar register n, n = 0 to 77_8
Tjk	Temporary register indicated by instruction j,k fields
V, Vn	Vector register n, n = 0 to 7
Vi, Vj, Vk	Vector register specified by instruction i or j or k field
VL	Vector Length register
VM	Vector Mask register
XA	Exchange Address register
XIOP	Auxiliary I/O Processor

READERS COMMENT FORM

CRAY-1 S Series Hardware Reference Manual HR-0808 B

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention:
PUBLICATIONS

Hwy 178 North
Chippewa Falls, WI 54729
U.S.A.

FOLD

STAPLE

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444