CRAY
RESEARCH, INC.

CRAY-1®
COMPUTER SYSTEMS

M SERIES
MAINFRAME REFERENCE MANUAL

HR-0064

RECORD OF REVISION   CRAY RESEARCH, INC.   PUBLICATION NUMBER HR-0064

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
	December, 1982 - Original printing.

PREFACE

This publication describes the functions of CRAY-1 M Series Computer Systems. It is written to assist programmers and engineers and assumes a familiarity with digital computers.

This manual describes the overall computer system, its configurations, and its equipment. It also describes the operation of the CPU, which executes programs, runs user jobs, and oversees job flow within the CRAY-1 M Series Computer System.

In addition, appendixes contain detailed reference information.

Details of the CRAY I/O Subsystem, Solid-state Storage Device, and the Mass Storage Subsystem are given in the following publications:

HR-0030	CRAY I/O Subsystem Reference Manual
HR-0031	Solid-state Storage Device (SSD) Reference Manual
HR-0630	Mass Storage Subsystem Hardware Reference Manual

//

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operations of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

//

CONTENTS

5. CPU COMPUTATION SECTION (continued)

APPENDIX SECTION (continued)

FIGURES

FIGURES (continued)

TABLES

INDEX

SYSTEM DESCRIPTION 1

INTRODUCTION

The CRAY-1 M Series of Computer Systems has a mainframe with 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, integrated circuit memory. Vector processing, 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. Models available in the M Series of Computer Systems are: CRAY-1 M/1200 with 1 million 64-bit words of memory; CRAY-1 M/2200 with 2 million 64-bit words of memory; and CRAY-1 M/4200 with 4 million 64-bit words of memory. These models are described in greater detail in section 2 under System Configurations. System components are explained later in this section.

All models of the CRAY-1 M Series of Computer Systems include a sophisticated I/O Subsystem with two I/O Processors that matches the CPU's 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. In addition, a CRI Solid-state Storage Device (SSD) can be attached to the CRAY-1 M mainframe. An SSD provides significantly improved throughput performance of programs that access large data files repetitively.

This section briefly describes the system components. Figure 1-1 illustrates a typical system.

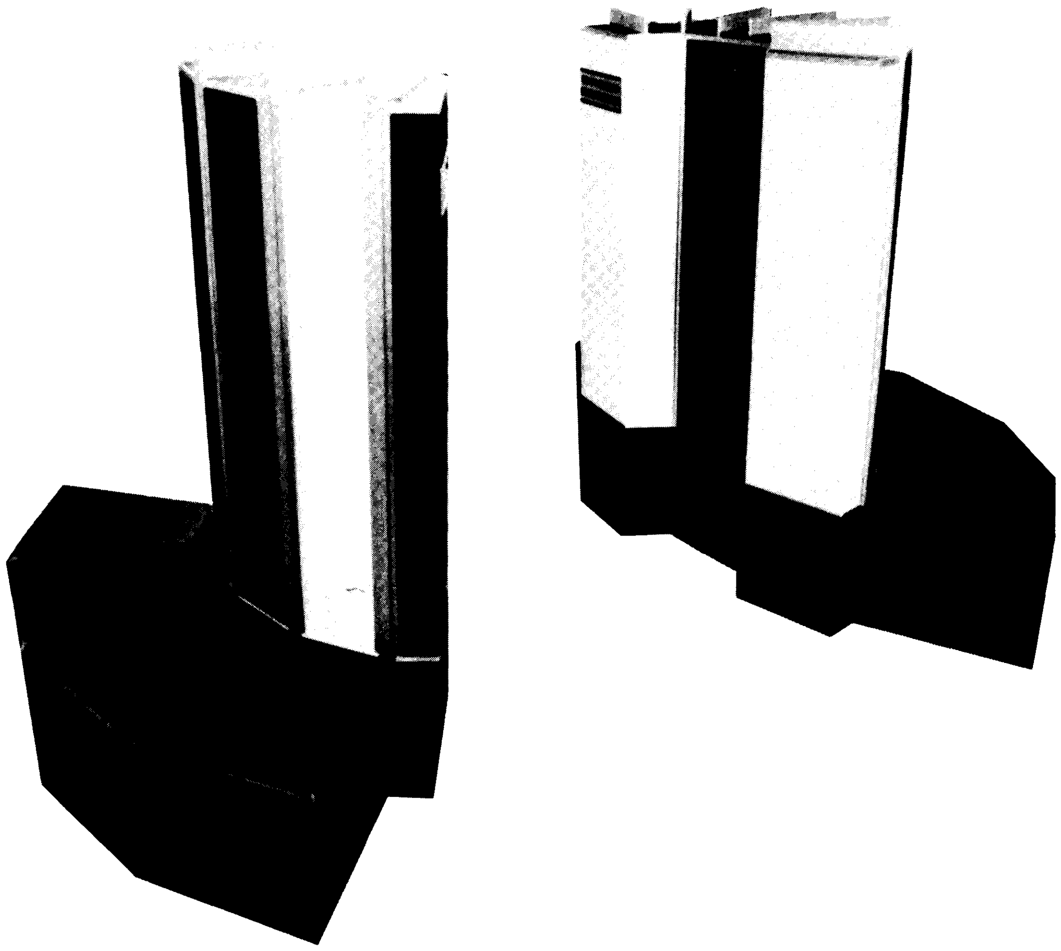


Figure 1-1. Typical CRAY-1 M Computer System

CONVENTIONS

The following conventions are used in this manual.

ITALICS

Italicized lowercase letters, such as *jk*, indicate variable information.

REGISTER CONVENTIONS

Parenthesized register names are used frequently in this manual as a form of 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."

Designations for the A, B, S, T, and V registers are used extensively. For example, "Transmit (T*jk*) to S*i*" means "Transmit the contents of the T register specified by the *jk* designators to the S register specified by the *i* designator."

Register bits are numbered right to left as powers of 2, starting with 2^0. Bit 2^{63} of an S, V, or T register value represents the most significant bit. Bit 2^{23} of an A or B register value represents the most significant bit. (A and B registers are 24 bits.)

The numbering conventions for the Exchange Package and the Vector Mask register are exceptions. Bits in the Exchange Package are numbered from left to right and are not numbered as powers of 2 but as bits 0 through 63 with 0 as the most significant and 63 as the least significant. The Vector Mask register has 64 bits, each corresponding to a word element in a vector register. Bit 2^{63} corresponds to element 0, bit 2^0 corresponds to element 63.

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, instruction parcels in instruction buffers, and instruction forms given in octal without the subscript.

SYSTEM COMPONENTS

The CRAY-1 M Series of Computer Systems is composed of a CRAY-1 M mainframe with a powerful Central Processing Unit (CPU) and an I/O Subsystem. Mass storage devices are also an integral part of a CRAY-1 M Series of Computer Systems. Optionally, a Cray Research, Inc., SSD can be a component of the CRAY-1 M Series of Computer Systems. Supporting the CRAY-1 M equipment are condensing units for refrigeration, power distribution units for the mainframe, the I/O Subsystem, and the SSD (optional), and motor-generators providing system power. Table 1-1 gives overall system characteristics.

Table 1-1. CRAY-1 M System characteristics

Configuration	- Central Processing Unit (CPU) - I/O Subsystem with 2, 3 (optional), or 4 (optional) I/O Processors - Optional Solid-state Storage Device (SSD)
CPU Speed	- 80 million floating-point additions per second - 80 million floating-point multiplications per second - Simultaneous floating-point addition and multiplication - 80 million half-precision floating-point divisions per second - 25 million full-precision floating-point divisions per second
Memories	- Up to 4 million 64-bit words in CPU Central Memory - 65,536 16-bit parcels in Local Memory of each I/O Processor in the I/O Subsystem - 6 direct memory access (DMA) ports (each I/O Processor) - 1, 4 (optional), or 8 (optional) million 64-bit words of I/O Subsystem Buffer Memory - 8, 16, or 32 million 64-bit words of SSD memory (optional)
Mass Storage	- 600 million byte disk drive - 48 disk drives maximum - 35.4 Mbits per second disk drive transfer rate
Input/Output	- Up to four 6 Mbytes per second channels - One standard and 1 optional 100 Mbytes per second channel to CPU (I/O Subsystem) - One 100 Mbytes per second channel and one 6 Mbytes per second channel per SSD - Mainframe interfaces - 40 accumulator channels per I/O Processor
Physical	- 32 sq ft floor space for CRAY-1 M mainframe - 24 sq ft floor space for I/O Subsystem - 24 sq ft floor space for SSD - 2.63 tons, mainframe weight - 1.5 tons, I/O Subsystem weight - 1.5 tons, SSD weight - Liquid refrigeration of each chassis - 400 Hz power from motor-generators

CENTRAL PROCESSING UNIT

The Central Processing Unit (CPU) is a single integrated processing unit with a memory section, a control section, a computation section, and an input/output section. Each CPU section is described in later sections of this publication.

The computation section is located in four columns of the CRAY-1 M mainframe chassis. An additional two columns contain memory. The bench at the base of each column houses the DC power supplies for that column.

Figure 1-2 represents the basic organization of the CPU; figure 1-3 illustrates the components of the CPU and presents a general view of data flow in the system. Figure 1-4 shows a CRAY-1 M mainframe chassis.

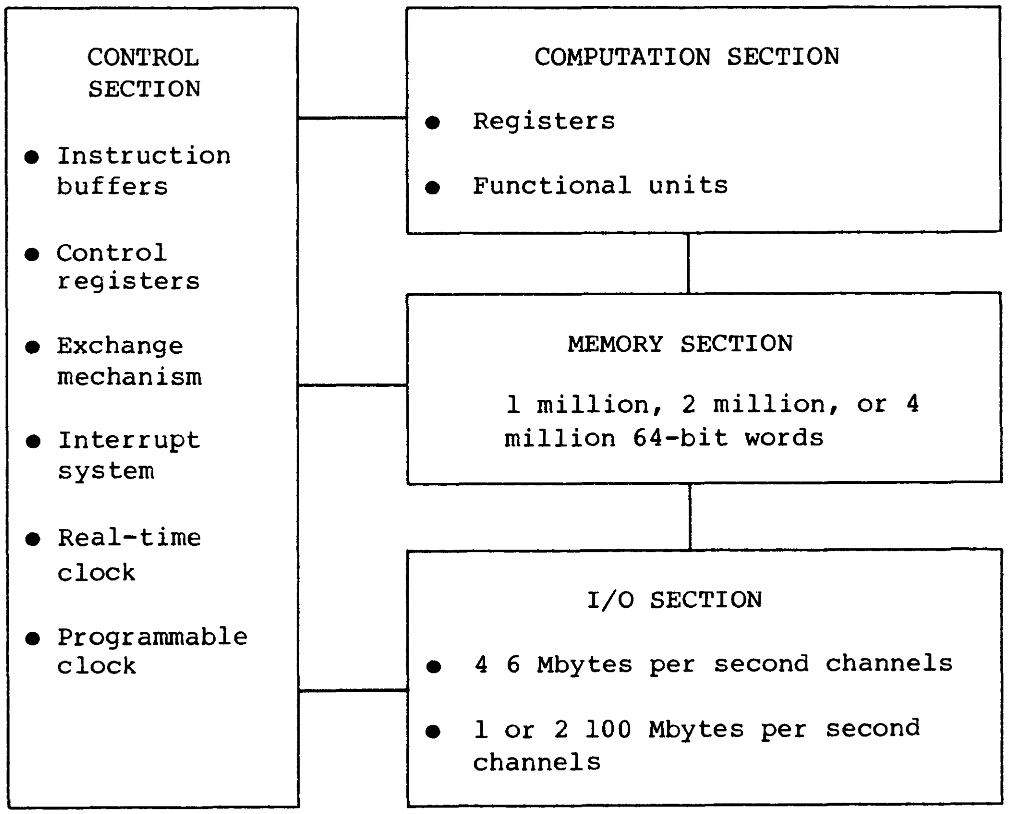


Figure 1-2. Basic organization of the CPU

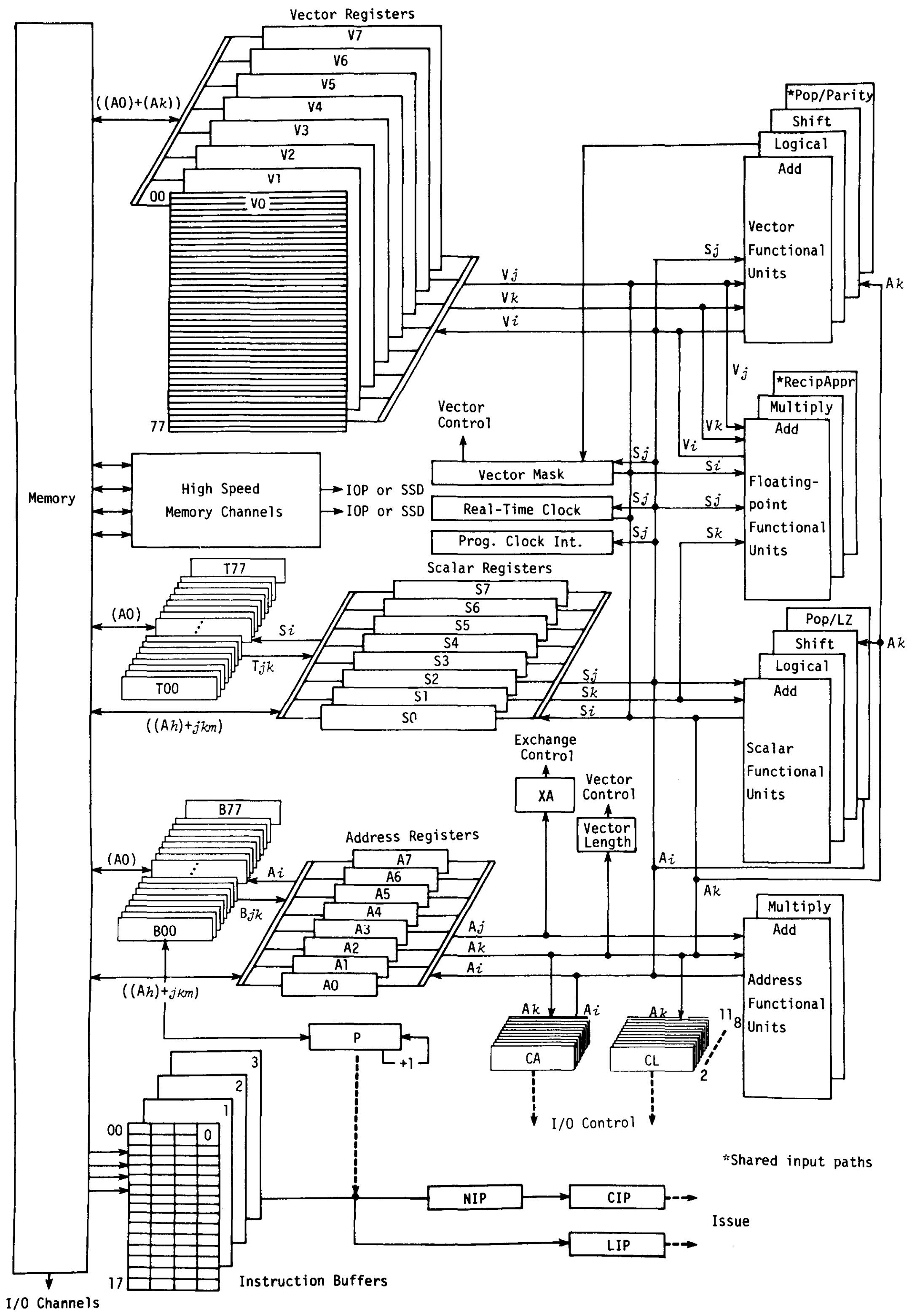


Figure 1-3. Control and data paths in the CPU

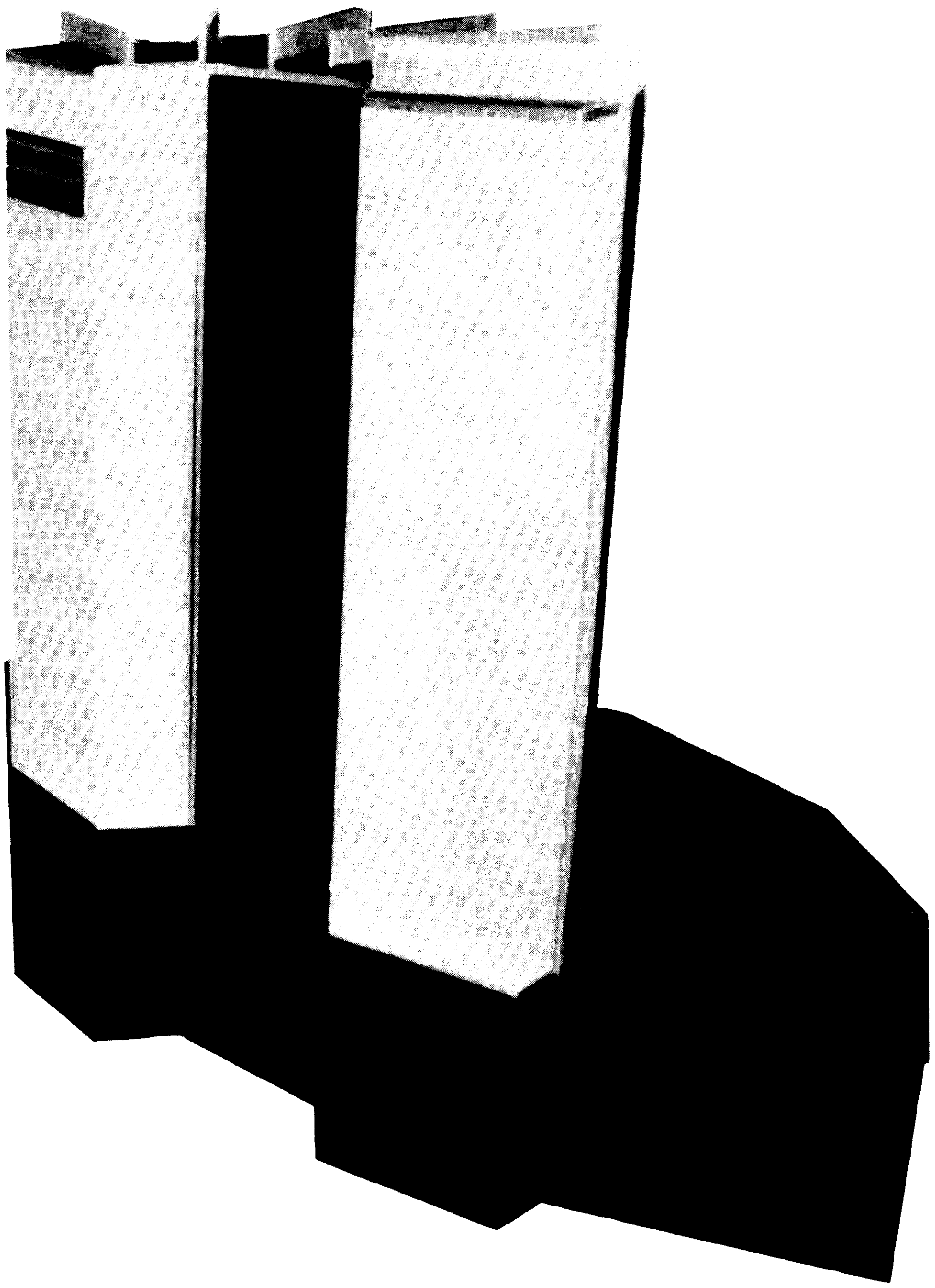


Figure 1-4. CRAY-1 M mainframe chassis

I/O SUBSYSTEM

CRAY-1 M computers are equipped with an I/O Subsystem composed of two, three (optional), or four (optional) I/O Processors, Buffer Memory, and required interfaces. I/O Processors (IOPs) are designed for fast data transfer between front-end computers, peripheral devices, storage devices, and Buffer Memory or between Buffer Memory and Central Memory of a CRAY-1 M mainframe.

Each IOP has a memory section, a control section, a computation section, and an input/output section. I/O sections are independent and handle some portion of the I/O requirements for the system. The I/O Subsystem is housed in a 4-column chassis (figure 1-5). Refer to the CRAY I/O Subsystem Reference Manual, publication HR-0030, for a detailed description of the I/O Subsystem.

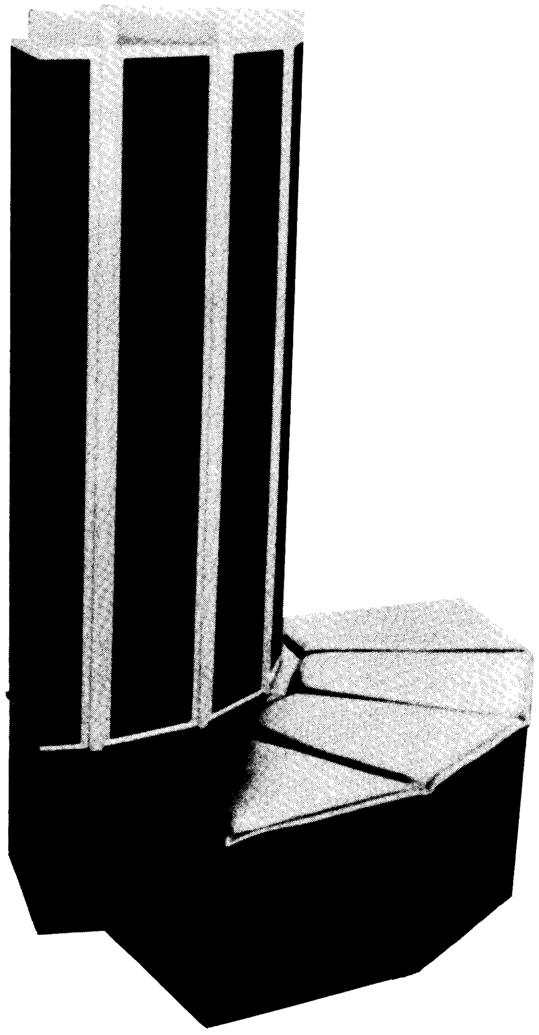


Figure 1-5. I/O Subsystem chassis

MASS STORAGE UNITS

The basic mass storage unit for the CRAY-1 M Series of Computer Systems is the DD-29 Disk Storage Unit (DSU). This unit is a 606 Mbyte disk drive with a data transfer rate of 35.4 Mbits per second. DD-29 DSU operational characteristics are summarized below.

Bytes per sector:	4096
Words per sector:	512
Sectors per logical track:	18
Words per track:	9216
Logical tracks per cylinder:	10
Words per logical cylinder:	92,160
Bits per drive:	4,854,251,520
Bytes per drive:	606,781,440
Words per drive:	75,847,680
Cylinders per drive:	823
Maximum latency:	16.6 msec.
Access time:	15 - 80 msec.

Transfer rate (maximum):

- One sector: 38.7×10^6 bits per second
- One cylinder (180 sectors): 35.4×10^6 bits per second†
- One drive (823 cylinders): 32.2×10^6 bits per second††

Up to four DD-29 DSUs can be connected to one DCU-4 Disk Controller Unit. The DCU-4 Disk Controller Unit interfaces the four disk units with an IOP of an I/O Subsystem through one direct memory access (DMA) port. The IOP and the disk controller unit can transfer data between the DMA port and four DD-29 DSUs with all DSUs operating at full speed without missing data or skipping revolutions. Depending on the CRAY-1 M Computer System configuration, a minimum of 2 and a maximum of 48 DD-29 DSUs can be configured. Figure 1-6 shows a DD-29 DSU. The DCU-4 Disk Controller Unit is housed in the I/O Subsystem chassis.

† Rate is less than one sector rate due to the time required to passover the sector address information prerecorded between sectors.

†† Rate is less than one cylinder rate due to the time required to move the heads one track (one cylinder).

Each DD-29 DSU has two accesses for connecting it to controllers. The second independent data path to each DSU can exist through another Cray Research, Inc., controller. Reservation logic provides controlled access to each DD-29 DSU.

Further information about the mass storage subsystem is included in the CRAY I/O Subsystem Reference Manual, CRI publication HR-0030, and the Mass Storage Subsystem Hardware Reference Manual, CRI publication HR-0630.

Figure 1-6. DD-29 Disk Storage Unit

SOLID-STATE STORAGE DEVICE

The Solid-state Storage Device (SSD) temporarily stores data offering significant performance improvements over disks. On a CRAY-1 M Computer System, the SSD requires a 100 Mbytes per second channel and a special controller to connect to the mainframe. This linkage also uses one of the four standard 6 Mbytes per second channels available on the mainframe.

The SSD is housed in a 4-column chassis (figure 1-7). For a detailed description of the SSD, refer to the Solid-state Storage Device (SSD) Reference Manual, CRI publication HR-0031.

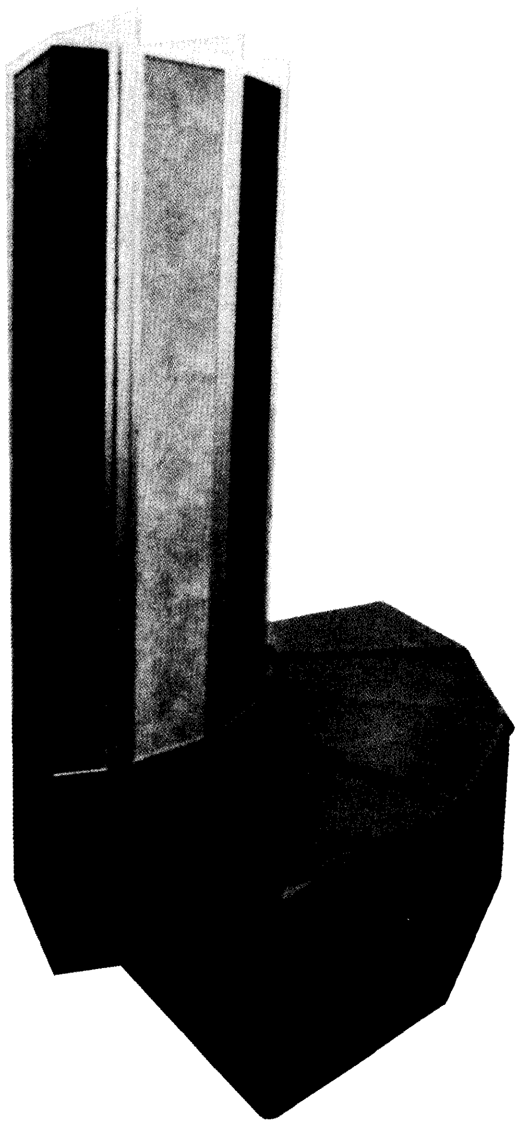
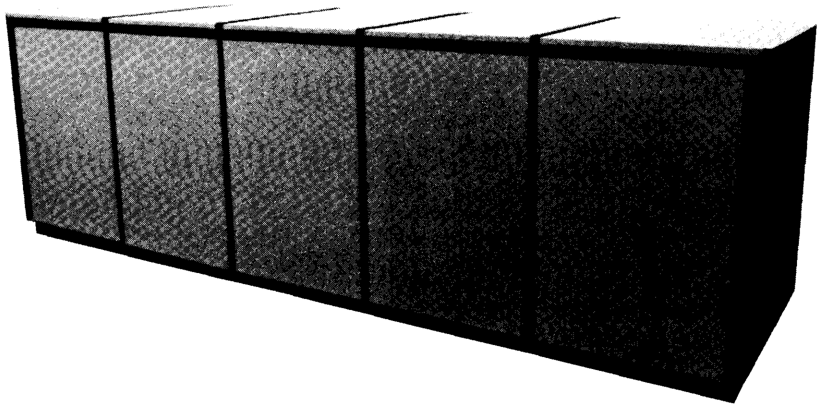


Figure 1-7. Solid-state Storage Device chassis

CONDENSING UNITS

Condensing units (figure 1-8) contain the major components of the refrigeration system used to cool the computer chassis and consist of two 25-ton condensers. Heat is removed from the condensing unit by a second level cooling system that is not part of the CRAY-1 M Computer System. Freon, which cools the computer, picks up heat, and transfers it to water in the condensing unit.



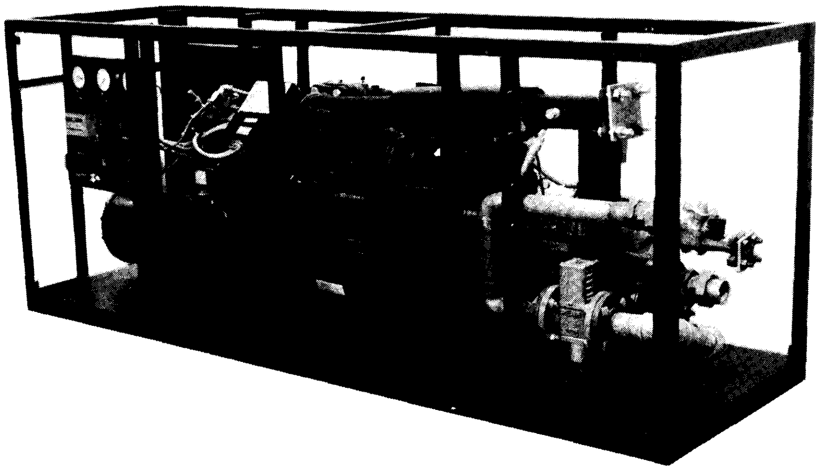


Figure 1-8. Condensing unit

POWER DISTRIBUTION UNITS

The CRAY-1 M mainframe, I/O Subsystem, and SSD all operate from 400 Hz 3-phase power. The power distribution unit (PDU-4) for the CRAY-1 M mainframe contains adjustable transformers to regulate the voltage to each power supply. The PDU-4 also contains temperature and voltage monitoring equipment that checks temperatures at strategic locations on the mainframe chassis. Automatic warning and shutdown circuitry protects the mainframe from overheating or excessive cooling. The control switches for the motor-generators and the condensing unit are mounted on the CRAY-1 M mainframe PDU-4.

A PDU-2 performs similar functions for the I/O Subsystem chassis, and the PDU-3 performs similar functions for the SSD chassis.

Figure 1-9 shows the power distribution units for the CRAY-1 M mainframe and for the I/O Subsystem.

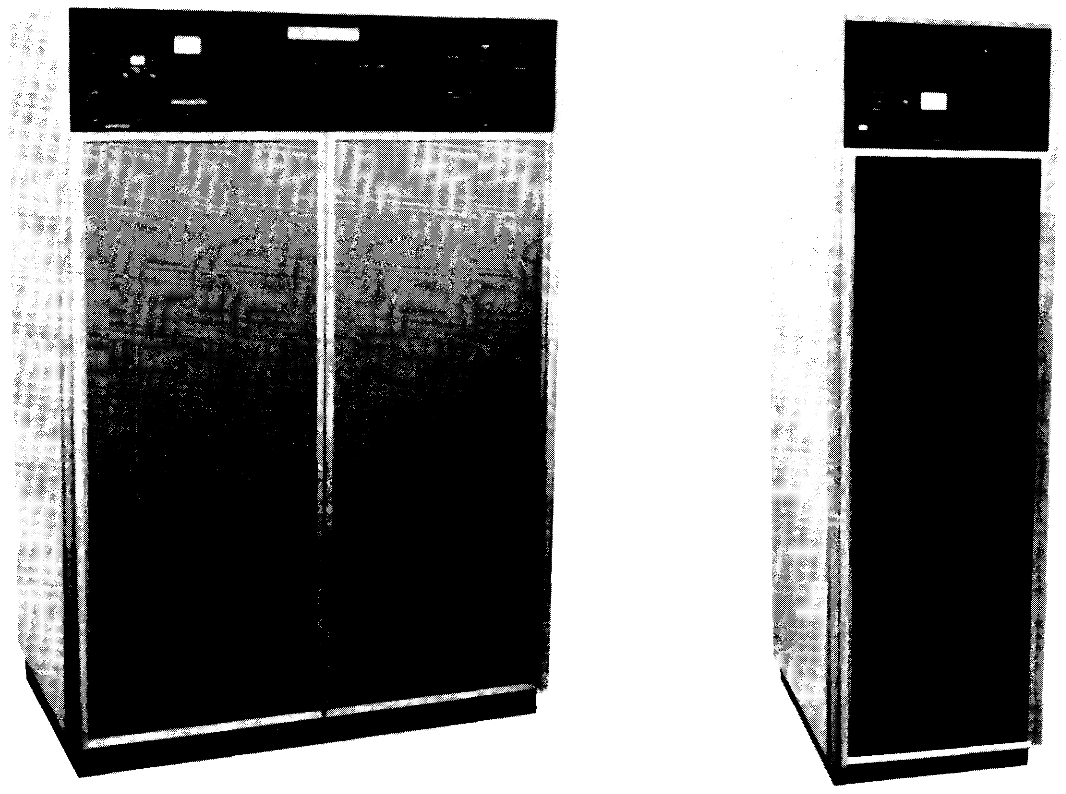


Figure 1-9. Power distribution units

MOTOR-GENERATOR UNITS

Motor-generator units convert primary power from the commercial power mains to the 400 Hz power used by the CRAY-1 M Computer System. These units isolate the system from transients and fluctuations on the commercial power mains. The equipment consists of two or three motor-generator units and a control cabinet. Figure 1-10 shows a typical motor-generator and the control cabinet.

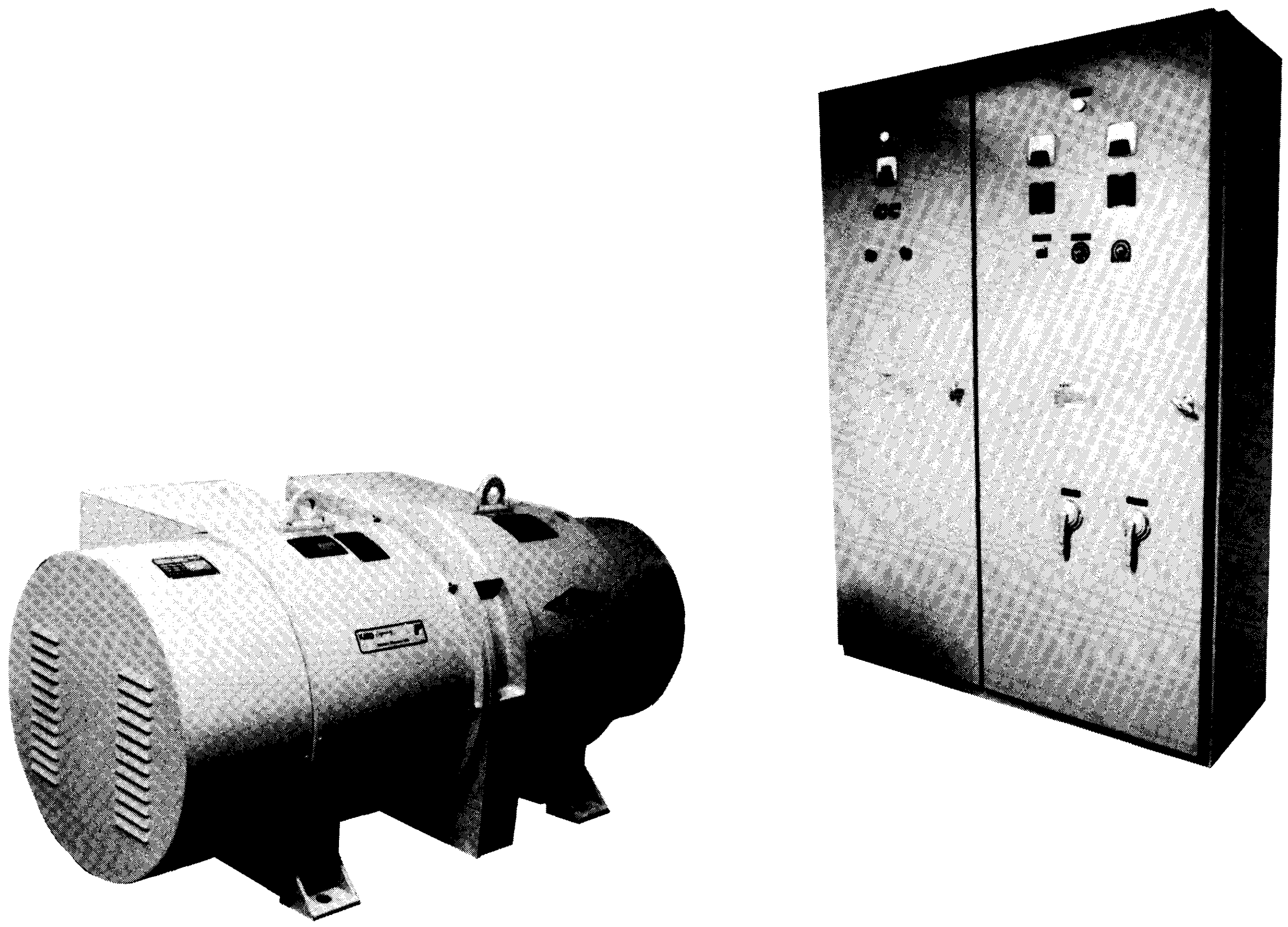


Figure 1-10. Motor-generator equipment

INTERFACES

The CRAY-1 M Computer is designed for use with a network of front-end computers. Standard front-end interfaces connect to the Master I/O Processor of a Cray I/O Subsystem via channels with a transfer rate of 6 Mbytes per second.

Most interfaces are housed in a stand-alone cabinet (figure 1-11) located near the host computer. The cabinet is air cooled and operates directly from the 60 Hz AC power mains. Power consumption and the heat generated by the interface cabinet vary 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 the unit can be easily integrated into a front-end computer with its specific grounding requirements. The interface uses hardware logic to perform command translation and protocol conversion needed to transfer data. Its operation is invisible to the front-end computer user and the CRAY-1 M user.

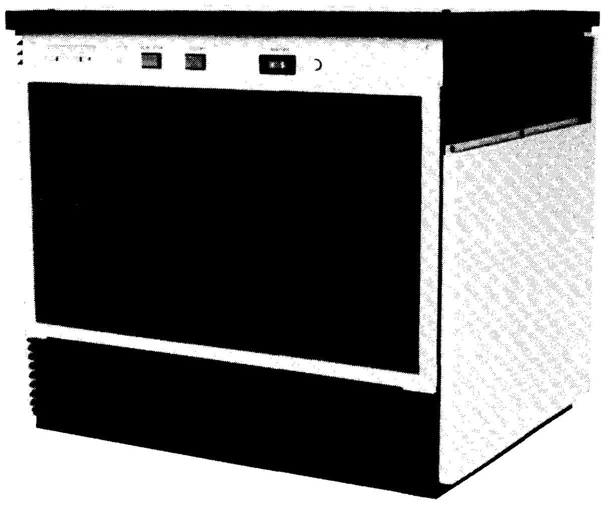


Figure 1-11. Typical interface cabinet

SYSTEM CONFIGURATION 2

INTRODUCTION

Several combinations of the basic system components are supported in the CRAY-1 M Series of Computer Systems. Central Memory of the CRAY-1 M mainframe is available in several different sizes. The standard I/O Subsystem consists of two processors. A Solid-state Storage Device (SSD) can also be included in the configuration of a CRAY-1 M system. The following paragraphs describe the standard models available in the CRAY-1 M Series.

M/1200, M/2200, AND M/4200 MODELS

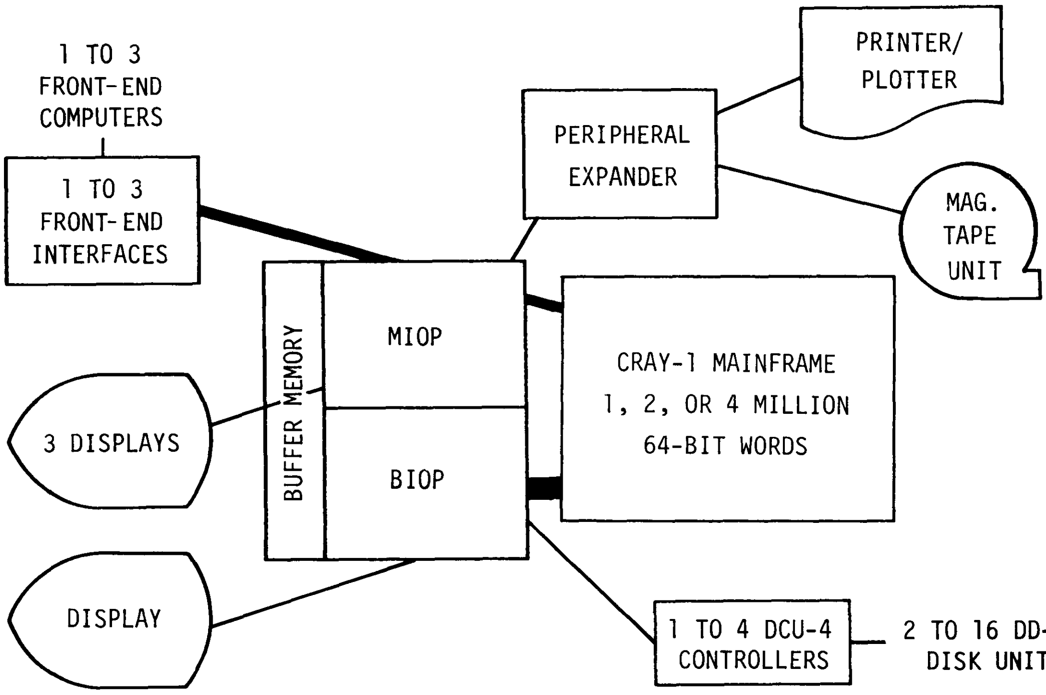
M/x200 systems share the characteristic of a 2-processor I/O Subsystem but differ in size of Central Memory: the M/1200 has 1 million words; the M/2200 has 2 million words; and the M/4200 has 4 million words. The M/x200 system mainframe chassis have 6 columns as standard. Figure 2-1 shows a configuration for these systems.

The Master I/O Processor (MIOP) controls front-end interfaces and the standard station peripherals. The Peripheral Expander interfaces the station peripherals to one direct memory access (DMA) port of the MIOP. The MIOP also connects to Buffer Memory and to Central Memory of the CRAY-1 M over a CRAY-1 M channel pair.

In M/x200 systems, the Buffer I/O Processor (BIOP) is the main link between Central Memory and the mass storage devices and is the only IOP having a standard 100 Mbytes per second channel to Central Memory. The M/x200 systems support up to 16 disk storage units.

M/1300, M/2300, AND M/4300 MODELS (OPTIONAL)

M/x300 systems share the characteristic of a 3-processor I/O Subsystem but differ in size of Central Memory: the M/1300 has 1 million words; the M/2300 has 2 million words; and the M/4300 has 4 million words. These configurations are the same as those described previously, except for the addition of a third IOP in the I/O Subsystem and an optional second 100 Mbytes per second channel.



▬ CRAY-1 M 6 Mbytes per second channel

■ CRAY-1 M 100 Mbytes per second channel

Figure 2-1. Block diagram of M/1200, M/2200, and M/4200 systems

The standard third IOP is a Disk I/O Processor (DIOP), which can have an optional second 100 Mbytes per second channel. A DIOP is used for additional disk storage units and handles 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 M/x200 models; up to 32 disk storage units can be used. The configuration for the systems having a DIOP as the third processor in the I/O Subsystem and the optional second 100 Mbytes per second channel is shown in figure 2-2.

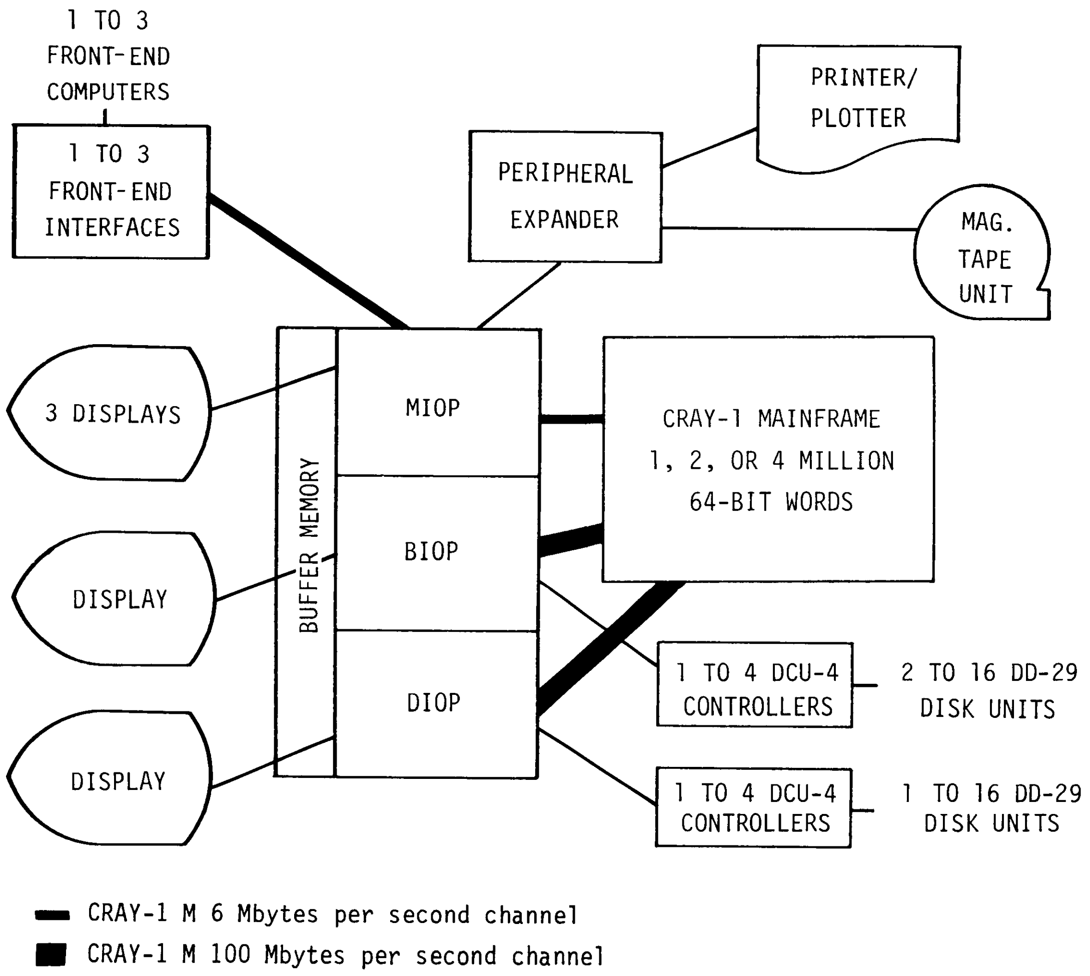


Figure 2-2. Block diagram of M/1300, M/2300, and M/4300 systems with increased disk capacity

An optional third IOP can be an Auxiliary I/O Processor (XIOP). The XIOP is used for block multiplexer channels and interfaces to a maximum of four BMC-4 Block Multiplexer Controllers, each of which can handle up to four block multiplexer channels. An XIOP uses one DMA port for each controller and another DMA port to connect with the Buffer Memory. An XIOP can have an optional second 100 Mbytes per second channel; however, software is not available to support this channel operation. The configuration for the systems having an XIOP as the third processor in the I/O Subsystem is shown in figure 2-3.

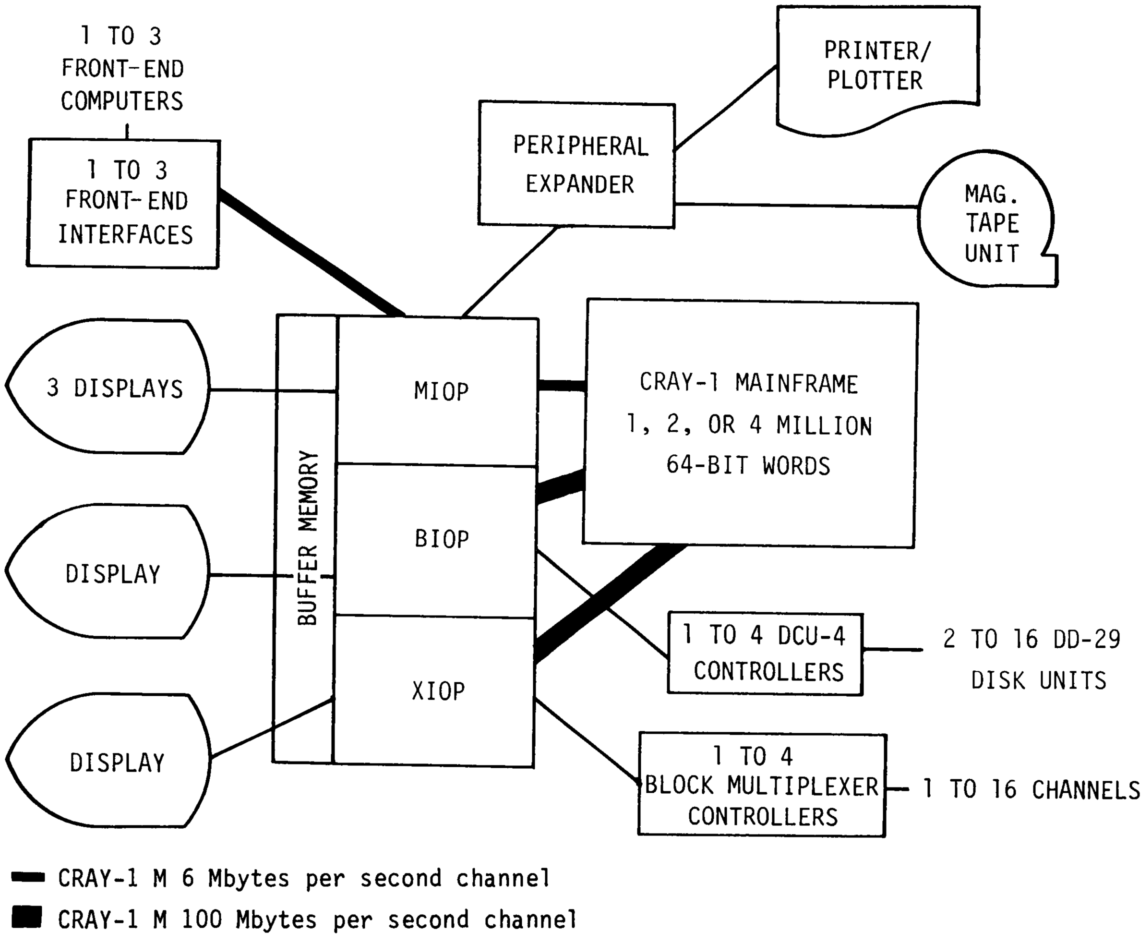


Figure 2-3. Block diagram of M/1300, M/2300, and M/4300 systems with block multiplexer channels

M/1400, M/2400, AND M/4400 MODELS (OPTIONAL)

M/x400 systems share the characteristic of a 4-processor I/O Subsystem but differ in the size of Central Memory: the M/1400 has 1 million words; the M/2400 has 2 million words; and the M/4400 has 4 million words. For the M/x400 systems, the third IOP handles disk storage units and can have an optional second 100 Mbytes per second channel. The fourth IOP is assigned to either block multiplexer controllers (standard) or to additional disk storage units (optional).

Figure 2-4 shows the configuration for the increased disk capacity. This configuration makes available the maximum mass storage resource; up to 48 disk storage units can be used.

Figure 2-5 shows the configuration for the block multiplexer channels. This configuration handles up to 16 channels via a maximum of four block multiplexer controllers.

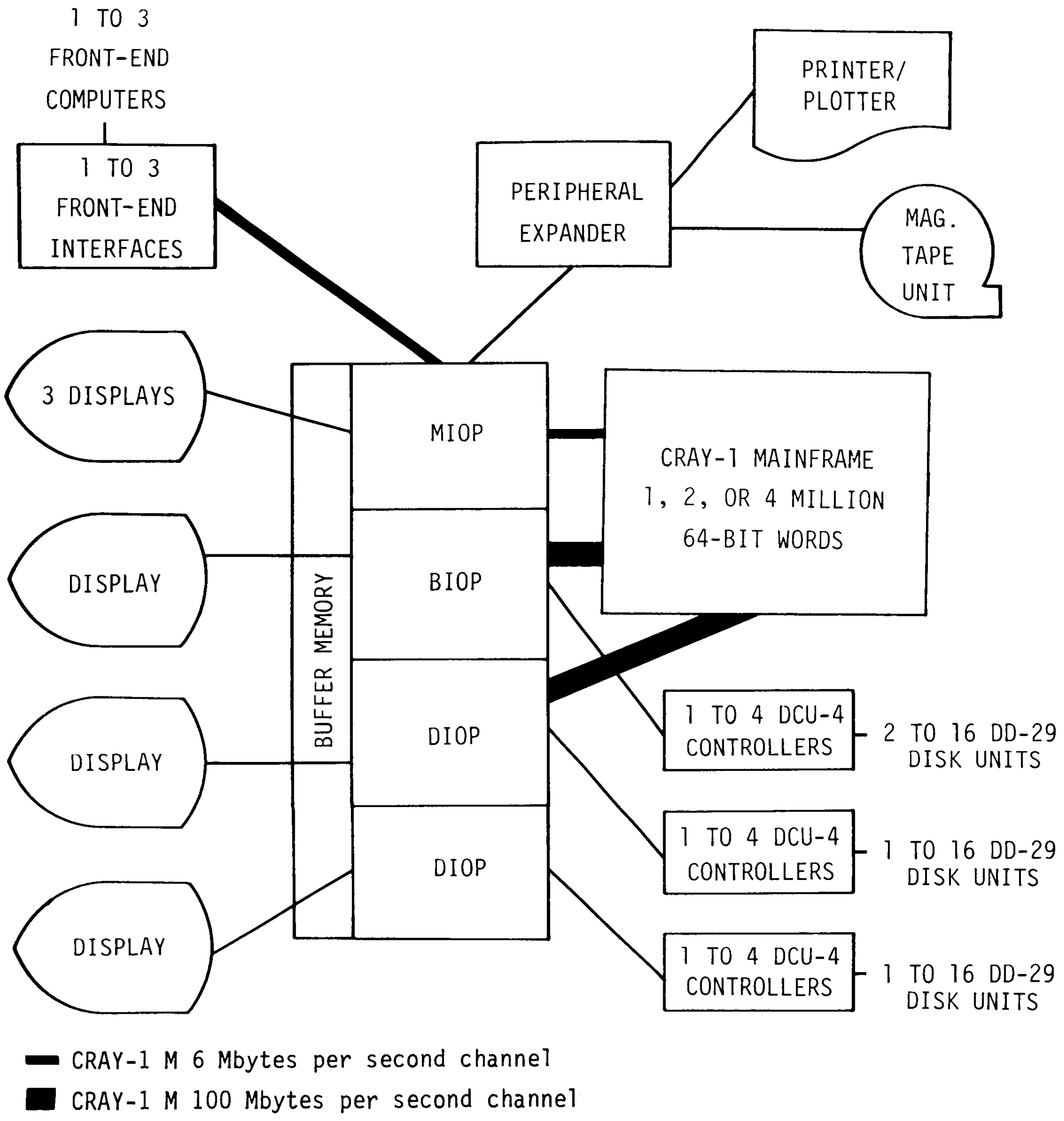
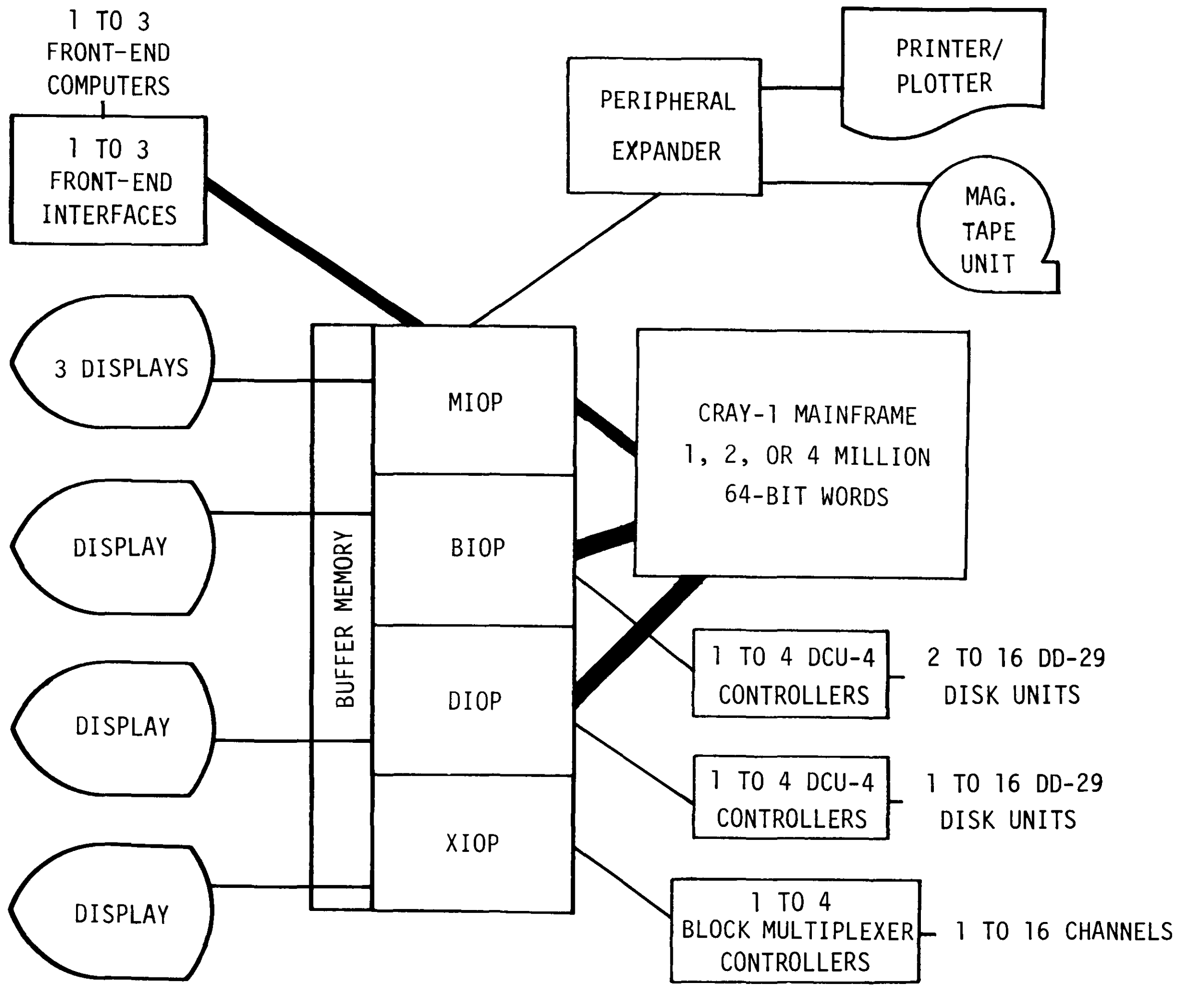


Figure 2-4. Block diagram of M/1400, M/2400, and M/4400 systems with increased disk capacity



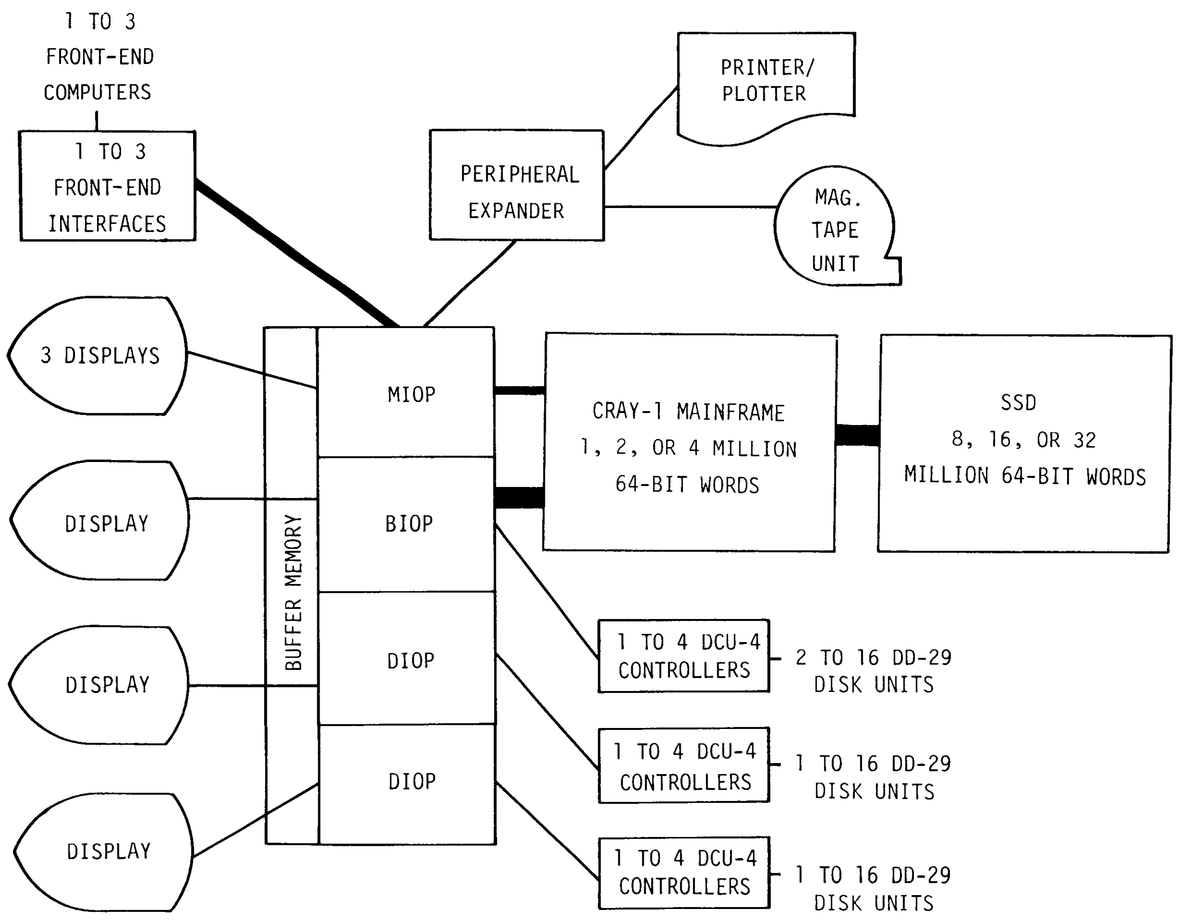
▬ CRAY-1 M 6 Mbytes per second channel

■ CRAY-1 M 100 Mbytes per second channel

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

CRAY-1 M AND SSD CONFIGURATION

The CRAY-1 M Computer System can be configured with an SSD using a 100 Mbytes per second channel, a standard 6 Mbytes per second channel, and a special controller to connect the SSD to the mainframe. Figure 2-6 shows a CRAY-1 M Computer System configured with an SSD.



▬ CRAY-1 M 6 Mbytes per second channel

■ CRAY-1 M 100 Mbytes per second channel

Figure 2-6. CRAY-1 M Computer System with SSD

INTERFACES TO FRONT-END COMPUTER

A front-end computer system is self contained and executes under the control of its own operating system. Standard interfaces connect the CRAY-1 M 6 Mbytes per second channels to channels of a variety of other computers providing input data to the CRAY-1 M Computer System and receiving output from it for distribution to peripheral equipment. Interfaces compensate for differences in channel widths, machine word size, electrical logic levels, and control signals. The MIOP communicates through a CRAY-1 M 6 Mbytes per second channel to a channel adapter module.

Interfaces to front-end computers allow the front-end computers to service the CRAY-1 M Computer System in the following ways:

- As a master operator station
- 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
- As an interactive communication station

Detailed information about the front-end system and the front-end communication protocol is outside the scope of this publication.

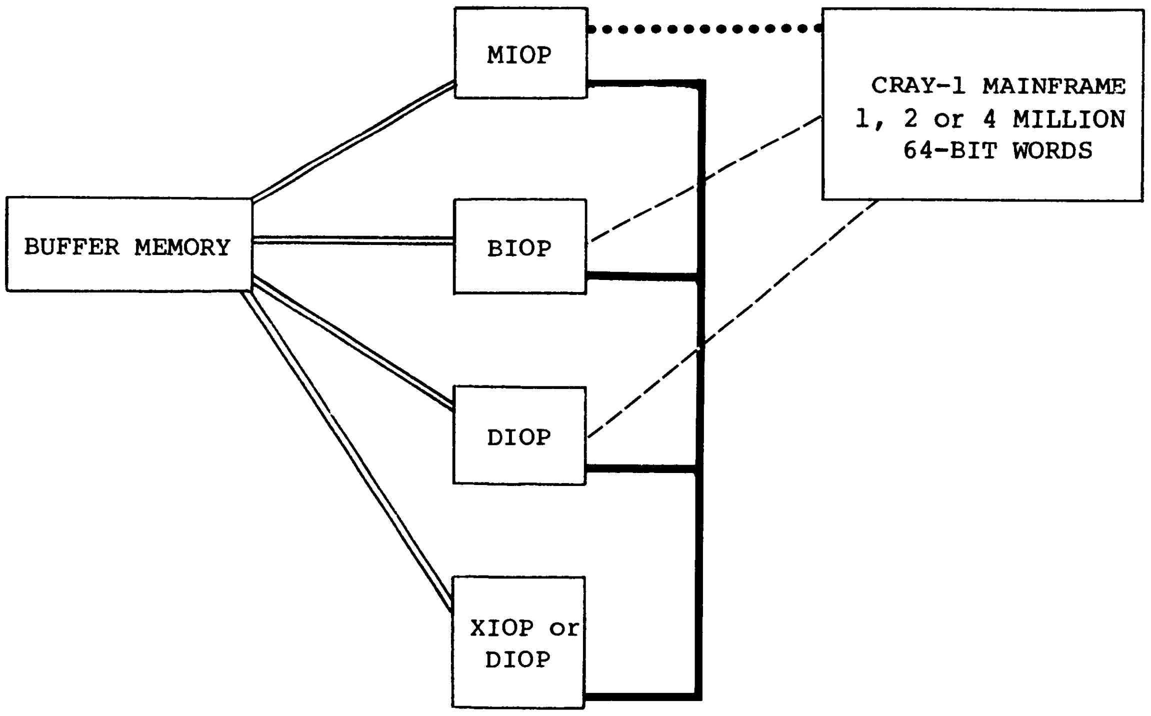
SYSTEM OPERATION

The CRAY-1 M Computer System consists of the components 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. The deadstart process (system initialization procedure) that brings the system to an operational state is described later in this section.

I/O SUBSYSTEM COMMUNICATION

The CRAY-1 M Series Computer System provides communication paths between Central Memory and the MIOP and BIOP (and between Central Memory and a DIOP or an XIOP if a second 100 Mbytes per second channel is present); between each IOP and Buffer Memory; and among all IOPs. The arrangement is shown in figure 2-7.

Communication between Central Memory and the IOPs is over one CRAY-1 M 6 Mbytes per second channel to the MIOP and over one or two 100 Mbytes per second channels to the BIOP and DIOP or XIOP. The CRAY-1 M 6 Mbytes per second channel exchanges system control information with the MIOP, while the 100 Mbytes per second channels transfer data through the BIOP and DIOP or XIOP.



●●●●●●●● 50 Mbit/s CRAY-1 I/O channel pair

- - - - - - Approximately 850 Mbit/s Memory Channel

═══════ Approximately 850 Mbit/s DMA channel

━━━━━━ Accumulator channel

Figure 2-7. I/O Subsystem communication

One DMA port of each IOP is connected with Buffer Memory through a channel with an approximate rate of 850 Mbits/second. Buffer Memory receives data from one IOP and stores it until the BIOP (or DIOP or XIOP if a second 100 Mbytes per second channel is present) can remove that data and pass it to Central Memory. In this way, each IOP communicates with every other IOP in high-speed data block transfers.

Additionally, each IOP is connected with the other IOPs by channels called accumulator channels. These channels pass one 16-bit parcel at a time from the accumulator of one IOP to the accumulator of another IOP and are used primarily for control and status reporting.

Any errors occurring in system memories or in the 100 Mbytes per second channel are reported through a special error channel separate from the data channels.

The resulting communications network among the processors speeds the flow of data from the front-end computers, peripheral devices, and mass storage units; stores the data as necessary; and passes the data to Central Memory. The network also facilitates transfer of results from Central Memory to the final destination. The CRAY I/O Subsystem Reference Manual, publication HR-0030, provides additional information on I/O Subsystem communication.

DEADSTART

The I/O Subsystem is initially deadstarted from the Peripheral Expander. Subsequent I/O Subsystem deadstarts can be from a device attached to the Peripheral Expander or a DD-29 Disk Storage Unit (DSU). Once the I/O Subsystem is operating, the CRAY-1 M mainframe can be deadstarted from a device attached to the Peripheral Expander or the DD-29 DSU. The startup command and procedures for installing deadstart files on the DD-29 DSU are described in the I/O Subsystem (IOS) Operator's Guide, CRI publication SG-0051.

CENTRAL MEMORY 3

INTRODUCTION

Central Memory consists of 8 or 16 independent banks of MOS integrated circuit memory. Three memory sizes are available:

- 1,048,576 words with 8 banks
- 2,097,152 words with 16 banks
- 4,194,304 words with 16 banks

Memory cycle time is 8 clock periods (CPs) or 100 nanoseconds (ns). Access time, the time required to fetch an operand from memory to an operating register, is 13 CPs. There is no inherent memory speed degradation for a 16-bank memory 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. For a 16-bank machine, transfer of instructions to the instruction buffers occurs at a rate of 16 parcels (four words) per CP.

Central Memory features are summarized below and described in detail in the following paragraphs.

- From 1 million to 4 million words of MOS integrated circuit memory
- 64 data bits and 8 error correction bits per word
- 8 or 16 interleaved banks
- 8-CP bank cycle time
- Transfer rate
  - 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 (16 bank)
- Single error correction/double error detection (SECDED)

MEMORY ACCESS

Central Memory is shared by the computation section and the I/O section with single port access.

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

Block transfers require completion of all memory requests before the block transfers can issue. Once issued, block transfers inhibit all other 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. Vector logic provides for identifying these conditions (speed control) and for slowing vector operations that would be affected by the slowed memory referencing rate. Vector logic identifies 1/8 speed (8 CPs), 1/4 speed (4 CPs), 1/2 speed (2 CPs), and full speed (1 CP) data rates from memory.

Fetch operations bring instructions from 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.

Memory must be quiet before exchange operations can reference it. After the exchange has issued, all other memory references are inhibited.

Scalar memory references are examined in six registers for possible memory conflicts. These six registers contain the low-order bits of each of the referenced memory addresses. These registers, plus the address register, represent 7 CPs between referencing any one bank. The first register is rank A, the second is rank B, the third is rank C, the fourth is rank D, the fifth is rank E, and the sixth is rank F. At each CP, contents of the registers are shifted down one rank until they are discarded. If a scalar conflict arises, the conflicting scalar address is held in rank B until the conflict is resolved.

I/O requests are held until memory is quiet. While I/O is being held, scalar memory references have access to memory. If four I/O requests are made with none being honored, scalar memory references hold off for one I/O memory reference.

For an I/O memory request to be processed, the following conditions must be present:

- I/O request
- Memory quiet or three previous I/O requests with none being honored
- No fetch request
- No block transfer instructions 034 through 037 (between memory and B or T registers) or block transfer instructions 176 or 177 (between memory and V registers) in progress

- No exchange sequence or request
- No instruction 033 request for channel status information (not a memory conflict)

A scalar reference cannot conflict with a scalar reference in rank A (CP 2 of a scalar instruction) 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 5 CPs. At the same time, 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 4 CPs. A Hold Issue signal blocks issue of another scalar reference instruction.

A scalar conflict in rank D (CP 5) causes a hold storage on this instruction for 3 CPs. A Hold Issue signal blocks issue of another scalar reference instruction.

A scalar conflict in rank E (CP 6) causes a hold storage on this instruction for 2 CPs. A Hold Issue signal blocks issue of another scalar reference instruction.

A scalar conflict in rank F (CP 7) causes a hold storage on this instruction for 1 CP. A Hold Issue signal blocks issue of another scalar reference instruction.

The 100 Mbytes per second channel shares the same access with 6 Mbytes per second channels, but 6 Mbytes per second channels have priority. The 100 Mbytes per second channel operates in blocks of 16 words with a 1-CP pause between blocks to allow other memory operations to break the 100 Mbytes per second channel transfer.

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

MEMORY ORGANIZATION

To minimize memory conflicts and to exploit the speed of the memory chips, Central Memory is organized into 8 or 16 banks. Each four banks occupy half a column and contain 36 memory modules and 17 address and data fan-in/fan-out modules. Each module contributes 8 data or check bits to each 72-bit word in the bank; a memory word consists of 64 data bits and 8 check bits.

The 8-bank phasing is required if only 1 column of memory is used. Although 8-bank phasing is possible on a 16-bank system (for maintenance purposes), the 16-bank phasing is required on 2-million or 4-million word machines.

MEMORY ADDRESSING

A word in an 8-bank memory is addressed in a maximum of 21 bits as shown in figure 3-1. The low-order 3 bits specify one of the 8 banks. The next field specifies an address within the chip. The high-order 4 bits specify one of the chips on the module.

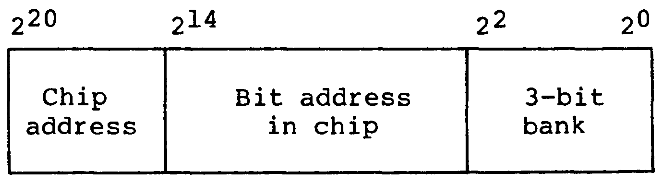


Figure 3-1. Memory address (8 banks)

A word in a 16-bank memory is addressed in a maximum of 22 bits as shown in figure 3-2. The low-order 4 bits specify one of the 16 banks. The next field specifies an address within the chip. The high-order 6 bits specify one of the chips on the module.

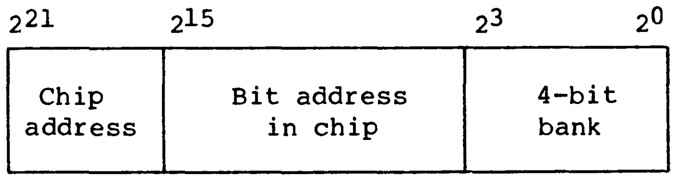


Figure 3-2. Memory address (16 banks)

SPEED CONTROL

For vector read and vector store instructions, the low-order 4 bits of (A*k*) determine speed control (see table 3-1).

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

Table 3-1. Vector memory rate x 80 x 10^6 references per second

Phasing	Increment or multiple in (A*k*)							
	1	2	3	4	5	6	7	8
8-bank	1	1/2	1	1/4	1	1/2	1	1/8
16-bank	1	1	1	1/2	1	1	1	1/4

Phasing	Increment or multiple in (A*k*)							
	9	10	11	12	13	14	15	16
8-bank	1	1/2	1	1/4	1	1/2	1	1/8
16-bank	1	1	1	1/2	1	1	1	1/8

MEMORY ERROR CORRECTION

A single error correction/double error detection (SECDED) network is used between the CPU and memory. SECDED assures that data written into memory is returned to the CPU with consistent precision (see figure 3-3).

If a single bit of a data word is altered, the single error alteration is automatically corrected before passing the data word to the CPU. If 2 bits of the same data word are altered, the error is detected but not corrected. In either case, the CPU can be interrupted depending on interrupt options selected to allow processing of the error. For 3 or more bits in error, results are ambiguous.

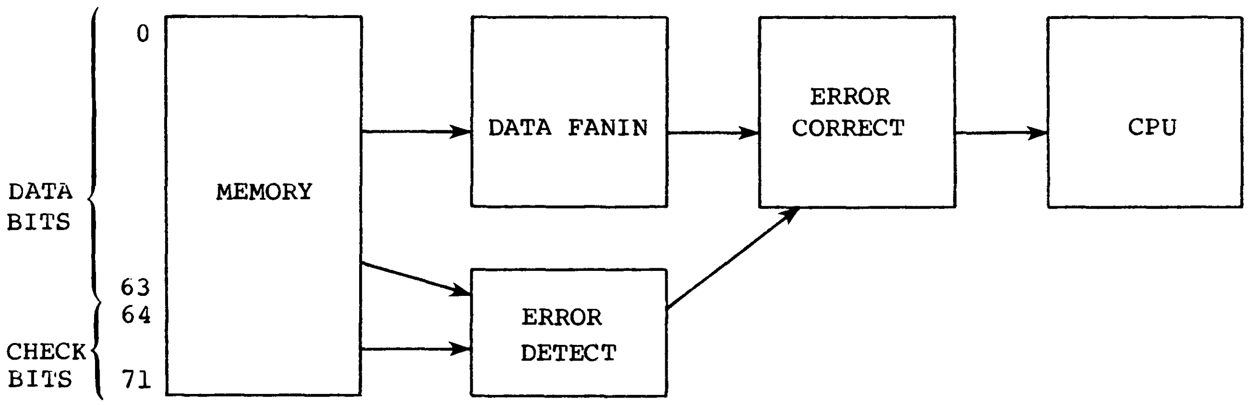


Figure 3-3. Memory data path with SECDED

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 as the data is written in memory. The 8 check bits are each generated as an even parity bits for a specific group of data bits. Figure 3-4 shows the bits of the data word that 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 and the data word are stored in memory at the same location. When read from memory, the same 64-bit matrix of figure 3-4 is used to generate a new set of check bits, which is compared with the old check bits. The resulting 8 comparison bits are called syndrome†† bits (S bits). The states of these S bits are all symptoms of any error that occurred (1=no compare). If all syndrome bits are 0, no memory error is assumed.

Any change of state of a single bit in memory causes an odd number of syndrome bits to be set to 1. A double error (an error in 2 bits) appears as an even number of syndrome bits set to 1.

The matrix is designed so that:

- If all syndrome bits are 0, no error is assumed.
- 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.

- 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.
- 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.
- If 3 or more memory bits are in error, the parity of syndrome bits will be odd or even and results are ambiguous.

	CHECK BYTE																							
	2^{71}	2^{70}	2^{69}	2^{68}	2^{67}	2^{66}	2^{65}	2^{64}	2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}
check bit 0								x									x	x	x	x	x	x	x	x
check bit 1							x		x	x	x	x	x	x	x	x								
check bit 2						x			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
check bit 3					x				x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
check bit 4				x					x		x		x		x		x		x		x		x	
check bit 5			x						x	x			x	x			x	x			x	x		
check bit 6		x							x	x	x	x					x	x	x	x				
check bit 7	x								x			x		x	x		x			x		x	x	

	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x		x		x	
	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			x	x		
									x	x	x	x	x	x	x	x	x	x	x	x				
	x	x	x	x	x	x	x	x									x			x		x	x	
	x		x		x		x		x		x		x		x									
	x	x			x	x			x	x			x	x			x	x	x	x	x	x	x	x
	x	x	x	x					x	x	x	x					x	x	x	x	x	x	x	x
	x			x		x	x		x			x		x	x		x	x	x	x	x	x	x	x

	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
	x		x		x		x		x		x		x		x		x		x		x		x	
	x	x			x	x			x	x			x	x			x	x			x	x		
	x	x	x	x					x	x	x	x					x	x	x	x				
	x			x		x	x		x			x		x	x		x			x		x	x	
	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
									x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x									x	x	x	x	x	x	x	x
	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x								

Figure 3-4. Error correction matrix

CPU CONTROL SECTION 4

INTRODUCTION

The control section of the CRAY-1 M CPU contains registers and instruction buffers for instruction issue and control and uses an exchange mechanism for switching instruction execution from program to program. These registers and buffers and the exchange mechanism are described in this section. Memory field protection, real-time clock, programmable clock, and deadstart sequence are also discussed.

INSTRUCTION ISSUE AND CONTROL

The registers and instruction buffers involved with instruction issue and control are described in the following paragraphs. Figure 4-1 illustrates the general flow of instruction parcels through the registers and buffers.

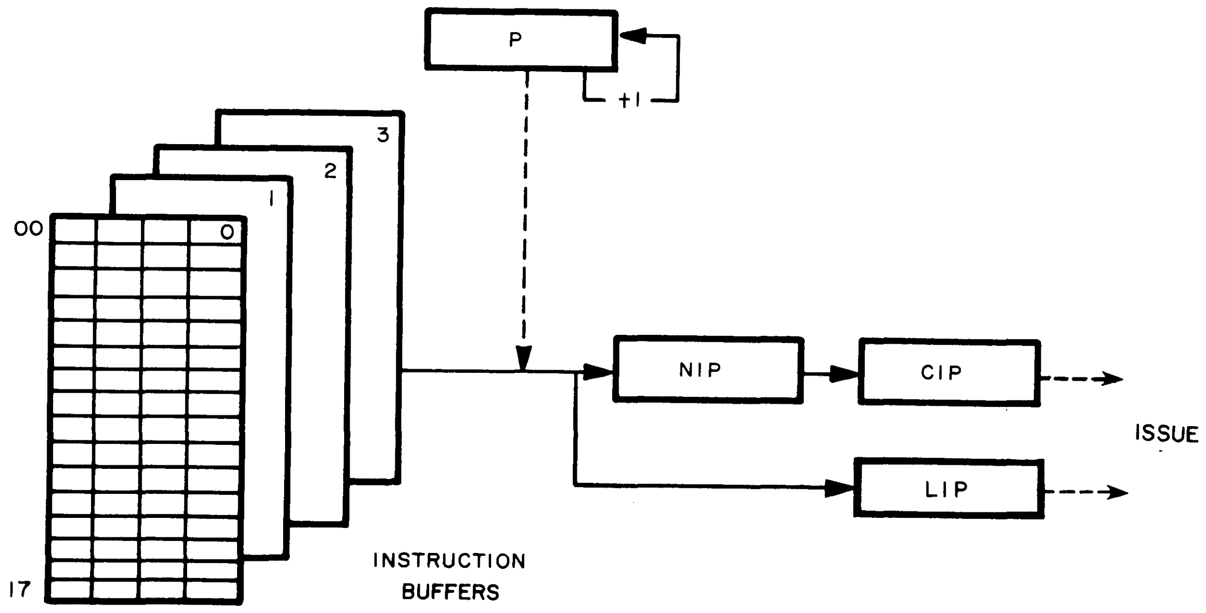


Figure 4-1. Instruction issue and control elements

PROGRAM ADDRESS REGISTER

The 24-bit Program Address (P) register indicates the next parcel of program code to enter the Next Instruction Parcel (NIP) register. 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. Except on a branch, the contents of the P register are advanced by 1 when an instruction parcel successfully enters the NIP register.

New data enters the P register on an instruction branch or on an exchange sequence. (The exchange sequence is described under Exchange Mechanism later in this section.) 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.

NEXT INSTRUCTION PARCEL REGISTER

The 16-bit Next Instruction Parcel (NIP) register holds a parcel of program code before it enters the Current Instruction Parcel (CIP) register. A parcel of program code entering the NIP register must issue, since there is no mechanism to discard it.

The NIP register is not master cleared. An undetermined instruction can issue during the master clear interval before the interrupt condition blocks data entry into the NIP register. At deadstart, instruction 000 is entered into the NIP register.

CURRENT INSTRUCTION PARCEL REGISTER

The 16-bit Current Instruction Parcel (CIP) register holds the instruction waiting to issue. If this instruction is a 2-parcel instruction, the CIP register holds the first parcel of the instruction and the Lower Instruction Parcel (LIP) holds the second parcel. Once an instruction enters the CIP register, it must issue; however, issue can 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. Indicators making up the instruction are distributed to all modules having mode selection requirements when the instruction issues.

Control flags associated with the CIP register are master cleared; the register itself is not. An undetermined instruction can issue during the master clear sequence.

LOWER INSTRUCTION PARCEL REGISTER

The 16-bit Lower Instruction Parcel (LIP) register holds the second parcel of a 2-parcel instruction when the first parcel of the 2-parcel instruction is in the CIP register.

INSTRUCTION BUFFERS

The CPU has four instruction buffers, each can hold 64 consecutive 16-bit instruction parcels (see figure 4-2). Instruction parcels are held in the buffers before being delivered to the NIP or LIP registers.

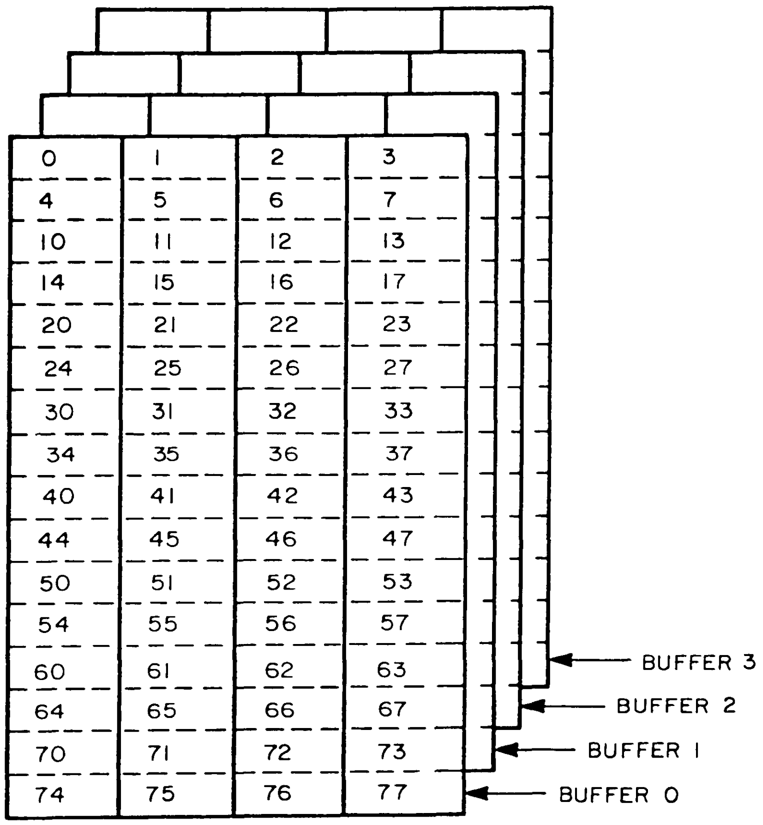


Figure 4-2. Instruction buffers

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

Beginning address registers are scanned each clock period (CP). 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 that instruction buffer. An instruction parcel to be executed normally is sent to the NIP register. However, the second parcel of a 2-parcel instruction is blocked from entering the NIP register and is sent to the LIP register instead. The second parcel of the 2-parcel instruction issues at the same time the first parcel issues from the CIP register. At the same time as the second half of a 2-parcel instruction is entering the LIP register, an all-zero parcel is entered into the NIP register.

On an in-buffer condition, if the instruction is in a different buffer than the previous instruction, a change of buffers occurs requiring 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 condition occurs, instructions must be loaded from memory into one of the instruction buffers before execution can continue. A 2-bit counter determines the instruction buffer receiving the instructions. Each 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 at the rate of four words per CP, fully occupying memory. The first group of 16 parcels delivered to the buffer always contains the instruction required for execution.

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 instruction buffers by setting their beginning address registers to all ones, preventing a match with the P register and causing the buffers to be loaded as needed. Therefore, the P register value in the new Exchange Package must not be all ones because there would be no fetch for the new Exchange Package.

Forward and backward branching are possible within buffers. Branching does not cause reloading of an instruction buffer if the address of the instruction being branched to is within one of the buffers. Multiple copies of instruction addresses cannot occur in instruction buffers.

Because instructions are held in instruction buffers before issue (and after until the buffer is reloaded), self-modifying code should be used very carefully. As long as the address of the unmodified instruction is in an instruction buffer, the modified instruction in memory is not loaded into an instruction buffer.

Although optimizing 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 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. An alternative is for a main program sequence in one or two of the buffers to make repeated calls to short subroutines maintained in the other buffers. The program and subroutines remain undisturbed in the buffers as long as no out-of-buffer condition causes reloading of a buffer.

EXCHANGE MECHANISM

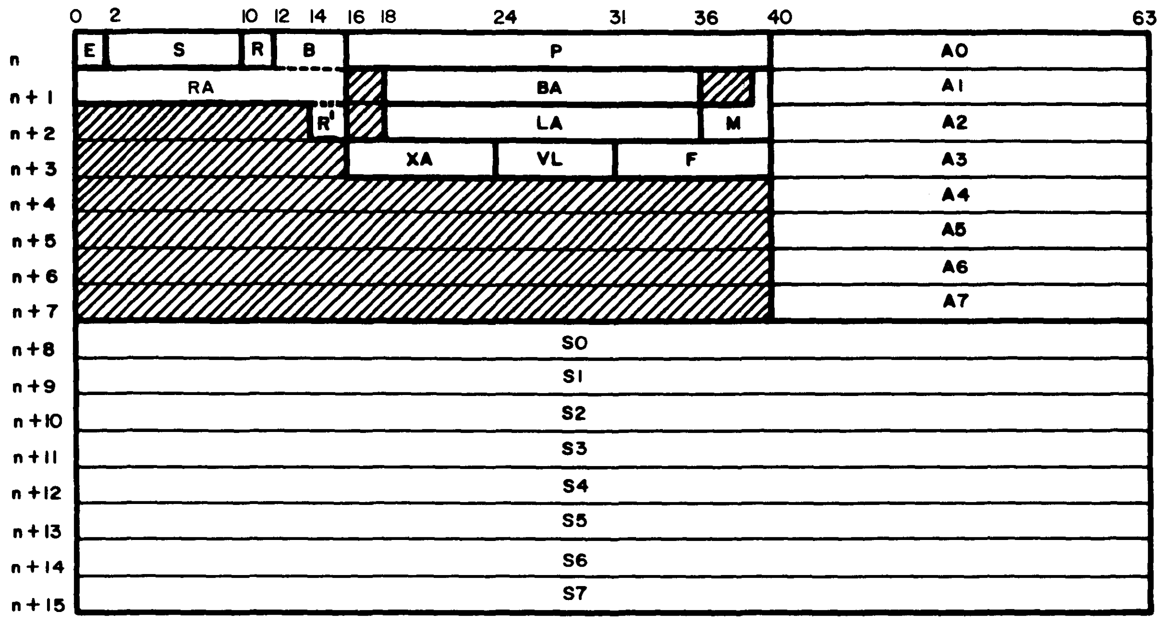
The CPU uses an exchange mechanism for switching instruction execution from program to program. The exchange mechanism uses blocks of program parameters called an Exchange Package and a CPU operation called an exchange sequence.

For the convenience of Cray Assembly Language (CAL) programmers, an alternate bit position representation is used when discussing the Exchange Package. The bits are numbered from left to right with bit 0 assigned to the 2^{63} bit position.

EXCHANGE PACKAGE

The Exchange Package (figure 4-3) is a 16-word block of data in memory associated with a particular computer program. The Exchange Package contains the basic parameters necessary to provide continuity from one program execution interval to the next. These parameters are:

- 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



	Registers
S	Syndrome bits
R'RAB	Read address for error
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
10	Vector
01	I/O
11	Fetch

Word Offset	Bit	M - Modes
n+1	39	Interrupt monitor mode
n+2	36	Interrupt on correctable memory error
n+2	37	Interrupt on floating-point
n+2	38	Interrupt on uncorrectable memory error
n+2	39	Monitor mode
		F - Flags
n+3	31	Programmable Clock Interrupt (PCI) interrupt
n+3	32	MCU interrupt
n+3	33	Floating-point error
n+3	34	Operand range error
n+3	35	Program range error
n+3	36	Memory error
n+3	37	I/O interrupt
n+3	38	Error exit
n+3	39	Normal exit

Figure 4-3. Exchange Package

The exchange sequence swaps data from memory to the operating registers and back to memory. This sequence exchanges data in the currently active Exchange Package residing in the operating registers with an inactive Exchange Package in memory. The Exchange Address (XA) register address of the currently active Exchange Package specifies the address of the inactive Exchange Package to be used in the swap. Data is exchanged and a new program execution interval is initiated by the exchange sequence.

The contents of the B, T, and V operating registers are not swapped in the exchange sequence. Data in these registers must be stored and replaced as required by specific coding in the program supervising the object program execution or by any program that needs this data. (See section 5 for descriptions of operating registers.)

Memory error data

Bit 36 (interrupt on correctable memory error bit) and bit 38 (interrupt on uncorrectable memory error bit) in the Mode (M) register determine if memory error data is included in the Exchange Package. Error data, consisting of four fields of information, appears in the Exchange Package if bit 36 is set and a correctable memory error is encountered or if bit 38 is set and an uncorrectable memory error is detected.

Memory error data fields are described below.

E - Error type	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.
S - Syndrome	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 3 for additional information.
R - Read mode	This field indicates which read mode was 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

R'RAB - Read address	This field contains the address where 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 can 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 14 and 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. These three registers are described below.

Exchange Address register

The 8-bit Exchange Address (XA) 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 specifying 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 the absolute address to 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.

Mode register

The 5-bit Mode (M) register contains part of the Exchange Package for a currently active program. The following bits are assigned in word 1 and word 2 of the Exchange Package.

Word 1

Bit	Description
39	Interrupt Monitor Mode flag; when set, enables all interrupts in monitor mode except PC, MCU, I/O, and normal exit.

Word 2

Bit	Description
36	Correctable Memory Error Mode flag; when set, enables interrupts on correctable memory data errors.
37	Floating-point Error Mode flag; when set, enables interrupts on floating-point errors.
38	Uncorrectable Memory Error Mode flag; when set, enables interrupts on uncorrectable memory data errors.
39	Monitor Mode flag; when set, inhibits all interrupts except memory errors.

The 5 bits are set selectively during an exchange sequence. Word 2, bit 37, the Floating-point Error Mode flag, is set or cleared during the execution interval for a program by using instructions 0021 (enable interrupt on floating-point error) and 0022 (disable interrupt on floating-point error). Remaining bits are not altered during the execution interval for the Exchange Package and are altered only when the Exchange Package is inactive in storage.

Flag register

The 9-bit Flag (F) register contains part of the Exchange Package for the currently active program. This register contains nine flags individually identified within the Exchange Package. Setting any of these flags interrupts program execution. When one or more flags are set, a Request Interrupt signal is sent to initiate an exchange sequence. The contents of the F register are stored with the rest of the Exchange Package. The monitor program can analyze the nine flags for the cause of the interruption. Before the monitor program exchanges back to the package, it must clear the flags in the F register area of the package. If any bit remains set, another exchange occurs immediately.

The F register bits are assigned as follows.

Bit	Description
31	Programmable Clock Interrupt flag; set when the programmable clock generates an interrupt. The programmable clock is explained later in this section.
32	MCU Interrupt flag; set by MCU interrupt; also set during the deadstart sequence to initiate an exchange. The deadstart sequence is explained later in this section.

Bit	Description
33	Floating-point Error flag; set when an overflow condition is detected by a floating-point functional unit. Floating-point functional units are explained in section 5, computation.
34	Operand Range Error flag; set when an out-of-range memory reference for an operand occurs. Operand range error is explained later in this section.
35	Program Range Error flag; set when an instruction fetch memory references an out-of-range address. Program range error is explained later in this section.
36	Memory Error flag; set by a memory error and generates an interrupt.
37	I/O Interrupt flag; when set, indicates the interrupt was generated by a 6 Mbytes per second channel.
38	Error Exit flag; set by an error exit instruction (000).
39	Normal Exit flag; set by a normal exit instruction (004).

Any flag (except the Memory Error flag) can be set in the F register only if the currently active Exchange Package is not in monitor mode. Such flags are set only if the low-order bit of the M register is 0. Except for the Memory Error flag, if the program is in monitor mode and conditions for setting an F register are present, the flag remains cleared and no exchange sequence is initiated.

ACTIVE EXCHANGE PACKAGE

An active Exchange Package resides in the operating registers. The interval of time when the Exchange Package and the program associated with it are active is called an execution interval. An execution interval begins with an exchange sequence where the subject Exchange Package moves from memory to the operating registers. An execution interval ends as the Exchange Package returns to memory in a subsequent exchange sequence.

EXCHANGE SEQUENCE

The exchange sequence is a vehicle for moving an inactive Exchange Package from memory into the operating registers. At the same time, the exchange sequence moves the currently active Exchange Package from the operating registers back into memory. This exchange 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. Location of this block is specified by the content of the XA register and is part of the currently active Exchange Package. The exchange sequence can be initiated by deadstart sequence, interrupt flag set, or program exit.

Exchange initiated by deadstart sequence

The deadstart sequence forces the XA register content to 0 and forces an instruction 000 in the NIP register. These two actions cause execution of a program error exit using memory address 0 as the location of the Exchange Package. The inactive Exchange Package at address 0 then moves into the operating registers and initiates a program using these parameters. The Exchange Package exchanged to address 0 is largely indeterminate because of the deadstart operation. New data entered at these storage addresses discards the Exchange Package.

Exchange initiated by interrupt flag set

An exchange sequence can be initiated by setting any one of the interrupt flags in the F register. Setting one or more flags results in a Request Interrupt signal initiating an exchange sequence.

Exchange initiated by program exit

Two program exit instructions initiate an exchange sequence. Timing of the instruction execution is the same in either case. The difference is determined by which of the two flags is set in the F register. The two instructions are:

000	ERR	Error exit
004	EX	Normal exit

Two exits enable a program to request its own termination. A non-monitor (object) program usually uses the normal exit instruction to exchange back to the monitor program. The error exit allows for abnormal termination of an object program. The exchange address selected is the same as for a normal exit.

Each instruction has a flag in the F register. The appropriate flag is set if 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. Flags are checked for evaluation of the program termination cause.

The monitor program selects an inactive Exchange Package for activation by setting the address of the inactive Exchange Package in the XA register and then executing a normal exit instruction.

Exchange sequence issue conditions

The following are hold issue conditions, execution time, and special cases for an exchange sequence.

HOLD ISSUE CONDITIONS:	Instruction buffer data invalid NIP register not blank Wait Exchange flag not set S, V, or A registers busy
EXECUTION TIME:	58 CPs; consists of an exchange sequence (40 CPs) and a fetch operation (18 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. The defined area must lie within the lower 4096 ($10,000_8$) words of memory. The package at address 0 is the initial monitor program's Exchange Package. 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 base and limit addresses for the programs. Only the monitor program has a field defined to access all of memory, including Exchange Package areas. The defined field 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 exchanges back to the monitor program that exchanged to it. The exchange ensures that program information is always exchanged into its proper Exchange Package.

For example, the monitor program (A) begins an execution interval following deadstart. No interrupts can terminate its execution interval since it is in monitor mode. Program A voluntarily exits by issuing a normal exit instruction (004). However, before doing so, program A sets the contents of the XA register to point to the user program (B) Exchange Package so that program B is the next program to execute. Program A sets the exchange address in program B's Exchange Package to point back to program A.

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

While program B is executing, an interrupt flag sets initiating an exchange sequence. Since program B cannot alter the XA register, the exit is back to program A. Program B's parameters exchange back into its Exchange Package area; program A's parameters held in program B's package during the execution interval exchange into the operating registers.

Program A, upon resuming execution, determines an interrupt caused the exchange and sets the XA register to call the proper interrupt processor into execution. To do this, Program A sets XA to point to the Exchange Package for the interrupt processing program (C). Program A clears the interrupt and initiates execution of program C by executing a normal exit instruction (004). Depending on the design of the operating system, program C can execute in monitor mode or in user mode.

Further information on Exchange Package management is contained in the CRAY-OS Version 1 Reference Manual, publication SR-0011.

MEMORY FIELD PROTECTION

At execution time each object program has a designated field of memory holding instructions and data. Field limits are specified by the monitor program when the object program is loaded and initiated. The field can begin at any word address that is a multiple of 16 and can continue to another address that is one less than a multiple of 16. Field limits are contained in the Base Address (BA) register and the Limit Address (LA) register, described below.

All memory addresses contained in the object program code are relative to the base address beginning 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 read reference beyond the assigned field limits issues and completes, but a zero value is transferred from memory. A memory write reference beyond the assigned field limits is allowed to issue, but no write occurs.

BASE ADDRESS REGISTER

The 18-bit Base Address (BA) register holds the base address of the user field during the execution interval for each Exchange Package. The contents of the BA 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.

LIMIT ADDRESS REGISTER

The 18-bit Limit Address (LA) register holds the limit address of the user field during the execution interval for each Exchange Package. The contents of the LA 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. 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 made for an instruction fetch. An out-of-range memory reference can occur in a non-monitor mode program on a branch or jump instruction calling for a program address above or below the limits. The Program Range Error flag causes an error condition that terminates program execution. The monitor program checks the state of the Program Range Error flag and takes 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 checks the state of the Operand Range Error flag and takes appropriate action, perhaps aborting the user program.

REAL-TIME CLOCK

Programs are timed precisely by using the clock period (CP) counter. The CP counter advances one count each CP. Since the clock advances synchronously with program execution, it can be used to time the program to an exact number of CPs. However, in such an application, the counting can be inaccurate if interrupts occur, exchanging the program.

Instructions used with the real-time clock (RTC) are:

0014*j*0	RT(S*j*)	Enter the RTC register with (S*j*)
072*ixx*	S*i* RT	Transmit (RTC) to S*i*

The 64-bit CP counter can be read by a program by using instruction 072 and can be reset only by the monitor mode instruction 0014*j*0.

PROGRAMMABLE CLOCK

The programmable clock accurately measures the duration of intervals. Intervals selected under monitor program control generate a periodic interrupt. Intervals shorter than 100 microseconds are not practical due to the monitor overhead involved in processing the interrupt.

INSTRUCTIONS

Supporting the programmable clock are the Interrupt Interval (II) register, the Interrupt Countdown (ICD) counter, and four monitor mode instructions:

0014*j*4	RT S*j*	Enter II register with (S*j*)
0014*x*5	CCI	Clear the programmable clock interrupt request
0014*x*6	ECI	Enable the programmable clock interrupt request
0014*x*7	DCI	Disable the programmable clock interrupt request

Interrupt Interval register

The 32-bit Interrupt Interval (II) register is loaded with a binary value equal to the number of CPs that are to elapse between programmable clock interrupt requests. The interrupt interval is transferred from the low-order 32 bits of the S*j* register into the II register and the ICD counter when instruction 0014*j*4 is executed.

This value is held in the II register and is transferred to the ICD counter each time the counter reaches 0 and generates an interrupt request. Content of the II register is changed only by another instruction 0014*j*4.

Interrupt Countdown counter

The 32-bit Interrupt Countdown (ICD) counter is preset to the content of the II register when instruction 0014*j*4 is executed. This counter runs continuously but counts down, decrementing by 1 each CP, until the content of the counter is 0. The ICD sets the programmable clock interrupt request and samples the interval value held in the II register. The ICD repeats the countdown to 0 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 clear programmable clock interrupt request is executed. A programmable clock interrupt request can be set only after the enable programmable clock interrupt request is executed. A programmable clock interrupt request causes an interrupt only 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 can be cleared by executing instruction 0014*x*5.

Following any deadstart, the monitor program should ensure the state of the programmable clock interrupt by issuing instructions 0014*x*5 and 0014*x*7.

DEADSTART SEQUENCE

The deadstart sequence of operations starts a program running in the CRAY-1 M mainframe after power has been turned off and then turned on again or whenever a new operating system is to be re-initialized in the mainframe. All registers in the machine, all control latches, and all words in memory should be considered invalid after turning on the power. The following sequence of operations to begin the program is initiated by 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 I/O Subsystem.

5. Turn off Master Clear signal.

The Master Clear signal halts all internal computation and forces critical control latches to predetermined states. The I/O Clear signal clears the input channel address register and activates the channel. All other input channels remain inactive. The 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 5

INTRODUCTION

The computation section consists of operating registers and functional units associated with three types of processing: address, scalar, and vector. Address processing operates on internal control information such as addresses and indexes and has two levels of 24-bit registers and two integer arithmetic functional units. Scalar and vector processing are performed on data.

A vector is an ordered set of elements. A vector instruction operates on a series of elements repeating the same function 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 eliminating instruction start-up time for all but the first operand. Scalar processing has two levels of 64-bit scalar registers, four functional units dedicated solely to scalar processing, and three floating-point functional units shared with vector operations. Vector processing has 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.

Address information flows from Central Memory or from control registers to address registers. Information in the address registers is 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 transmitted to Central Memory from the address registers.

Data flow in a computation section is generally from Central Memory to registers and from registers to functional units. Results flow from functional units to registers and from registers to Central Memory or back to functional units. Data flows along either the scalar or vector path depending on the processing mode. An exception is that scalar registers can provide one required operand for vector operations performed in the vector functional units.

Integer or floating-point arithmetic operations are performed in the computation section. 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 provide for a floating-point divide operation using a multiple instruction sequence. These instructions produce 64-bit results (1-bit sign, 15-bit exponent, and 48-bit normalized coefficient).

Integer or fixed-point operations are integer addition, integer subtraction, and integer multiplication. Integer addition and subtraction operations produce either 24-bit or 64-bit results. An integer multiply operation produces a 24-bit result. A 64-bit integer multiply operation is done through a software algorithm using the floating-point multiply functional unit to generate multiple partial products. These partial products are then shifted and merged to form the full 64-bit product. 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 and 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 can be positive or negative in either mode. Indexing 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. An additional scalar operation is the leading zero counts.

Characteristics of a CPU computation section are summarized below.

- Integer and floating-point arithmetic
- Twos complement integer arithmetic
- Signed 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

OPERATING REGISTERS

Operating registers, a primary programmable resource of the CPU, enhance the speed of the system by satisfying heavy demands for data made by functional units. A single functional unit requires one to three operands per clock period (CP) to perform the necessary function and delivers results at a rate of one per CP. Multiple functional units can be used concurrently.

The CPU has three primary and two intermediate sets of registers. The 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 scalar and address registers, an intermediate level of registers exists that is not accessible to functional units. These intermediate registers act as buffers for primary registers. Block transfers are possible between these registers and Central Memory so that the number of memory reference instructions required for scalar and address operands is greatly reduced. Intermediate registers supporting scalar registers are referred to as T registers. Intermediate registers supporting the address registers are referred to as B registers.

ADDRESS REGISTERS

Figure 5-1 illustrates registers and functional units used for address processing. The two types of address registers are designated A registers and B registers and are described in the following paragraphs.

A REGISTERS

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

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. Several address adders are devoted exclusively to calculations for memory references and are not available to the program.

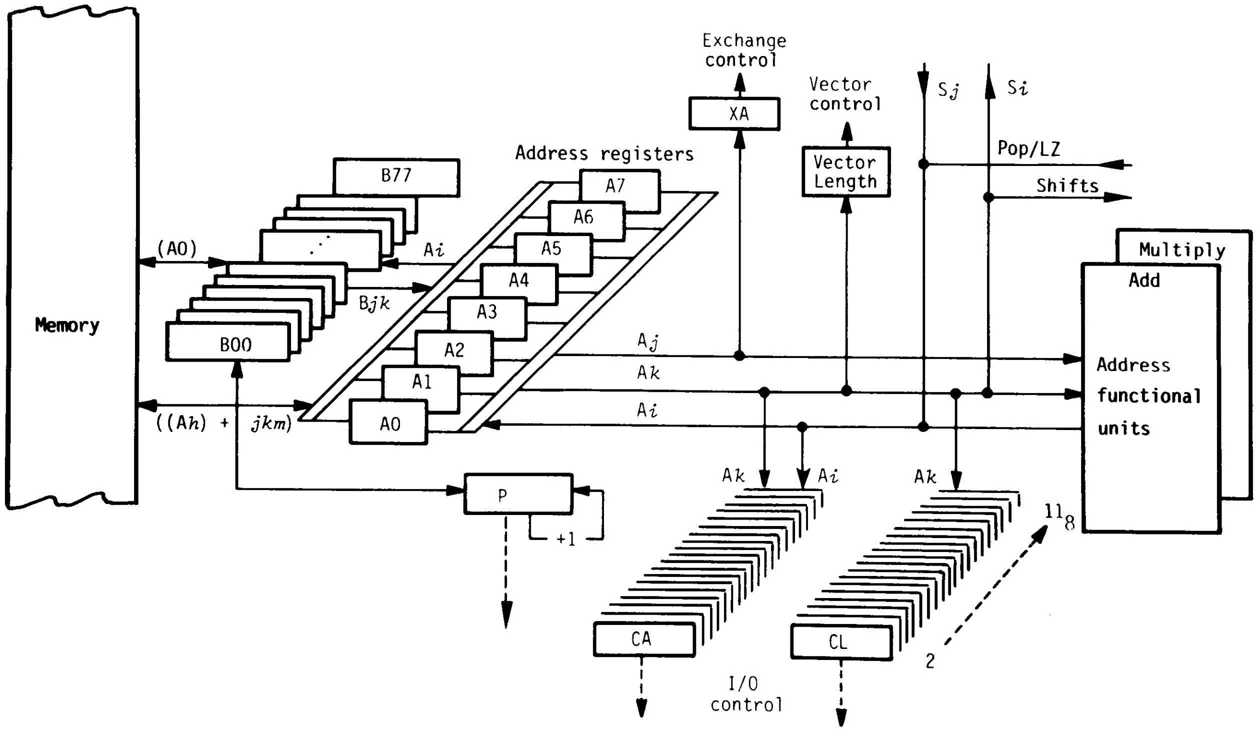


Figure 5-1. Address registers and functional units

Data is moved directly between Central Memory and A registers or is placed in B registers. Placing data in B registers allows buffering of the data between A registers and Central Memory. Data is also transferred between A and S registers.

The Vector Length (VL) register and Exchange Address (XA) register are set by transmitting a value to them from an A register. (The VL register is described under Vector Control Registers later in this section.)

Only one A or B register can be entered with data during each CP. Instruction issue is delayed if it causes data to arrive at the A or B registers concurrently with data already being processed and scheduled to arrive from another source.

When an issued instruction delivers new data to an A register, a reservation is set for that register. The reservation prevents issue of instructions that use the register until new data is delivered.

In this manual, A registers are individually referred to by the letter A followed by a number ranging from 0 through 7. Instructions reference A registers by specifying the register number as the *h*, *i*, *j*, or *k* designator as described in section 7.

The only register implicitly referenced is the A0 register as illustrated in the following instructions:

010*ijkm*	JAZ *exp*	Branch to *ijkm* if (A0)=0
011*ijkm*	JAN *exp*	Branch to *ijkm* if (A0)≠0
012*ijkm*	JAP *exp*	Branch to *ijkm* if (A0) is positive, includes (A0)=0
013*ijkm*	JAM *exp*	Branch to *ijkm* if (A0) is negative
034*ijk*	B*jk*,A*i* ,A0	Read (A*i*) words to B register *jk* from (A0)
035*ijk*	,A0 B*jk*,A*i*	Store (A*i*) words at B register *jk* to (A0)
036*ijk*	T*jk*,A*i* ,A0	Read (A*i*) words to T register *jk* from (A0)
037*ijk*	,A0 T*jk*,A*i*	Store (A*i*) words at T register *jk* to (A0)
176*ixk*	V*i* ,A0,A*k*	Read (VL) words to V*i* from (A0) incremented by (A*k*)
177*xjk*	,A0,A*k* V*j*	Store (VL) words from V*i* from (A0) incremented by (A*k*)

Section 7 of this manual contains additional information on the use of A registers by instructions.

B REGISTERS

The CPU has sixty-four 24-bit B registers used as intermediate storage for A registers. Typically, B registers contain data to be referenced repeatedly over a sufficiently long span making it unnecessary to retain the data in either A registers or in Central Memory. Examples are loop counts, variable array base addresses, and dimensions.

Transfer of a value between an A register and a B register requires only 1 CP. A block of B registers is transferred to or from Central Memory at the maximum rate of one 24-bit value per CP. No reservations are made for B registers and no instructions are issued during block transfers to and from B registers.

Only one B register is entered with data during each CP. Issue of an instruction is delayed if it causes data to arrive at the B registers concurrently with data already being processed and scheduled to arrive from another source.

In this manual, B registers are individually referred to by the letter B followed by a 2-digit octal number ranging from 00 through 77. Instructions reference B registers by specifying the B register number in the *jk* designator as described in section 7.

The only B register implicitly referred to is the B00 register. On execution of the return jump instruction (007), register B00 is set to the next instruction parcel address (P) and a branch to an address specified by *ijkm* occurs. On receiving control, the called routine conventionally saves (B00) so that the B00 register is available 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 to B*jk* and executes instruction 0050*jk*. This instruction, which is a branch to (B*jk*), causes the address currently in B*jk* to be entered into the P register as the address of the next instruction parcel to be executed.

SCALAR REGISTERS

Figure 5-2 illustrates registers and functional units used for scalar processing. The two types of scalar registers are designated S registers and T registers and are described in the following paragraphs.

S REGISTERS

Eight 64-bit S registers are the principal scalar registers for the CPU serving as source and destination for operands executing scalar arithmetic and logical instructions. Scalar functional units perform both integer and floating-point arithmetic operations.

S registers can furnish one operand in vector instructions. Single-word transmissions of data between an S register and an element of a V register are also possible.

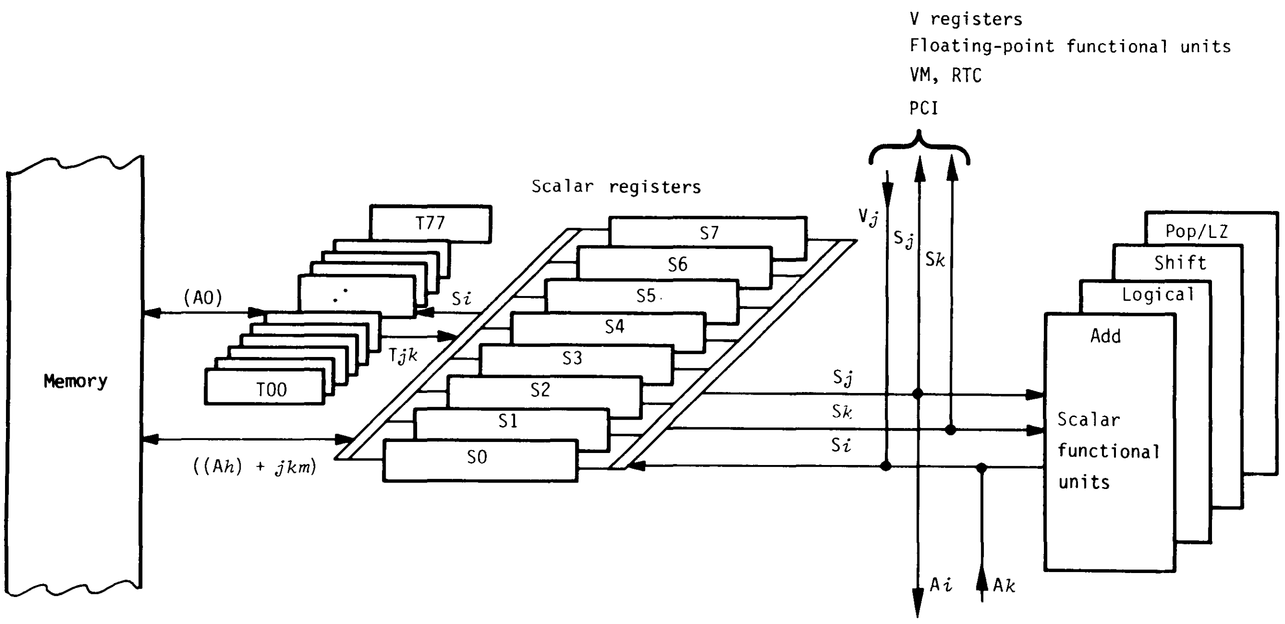


Figure 5-2. Scalar registers and functional units

Data moves directly between Central Memory and S registers or is placed in T registers. This intermediate step allows buffering of scalar operands between S registers and Central Memory. Data is also transferred between A and S registers.

Other uses of S registers include setting or reading of the Vector Mask (VM) register or the Real-time Clock (RTC) register, or setting the Interrupt Interval (II) register. (The VM register is described under Vector Control Registers later in this section.)

Only one S or T register can receive data during each CP. Issue of an instruction is delayed if it causes data to arrive at the S or T registers concurrently with data already being processed and scheduled to arrive from another source.

When an issued instruction delivers new data to an S register, a reservation is set for that register preventing issue of instructions using the register until new data is delivered.

In this manual, S registers are individually referred to by the letter S followed by a number ranging from 0 through 7. Instructions reference S registers by specifying the register number as the *i*, *j*, or *k* designator as described in section 7.

The only register implicitly referred to is the S0 register as illustrated in the following branch instructions.

014*ijkm*	JSZ	*exp*	Branch to *ijkm* if (S0)=0
015*ijkm*	JSN	*exp*	Branch to *ijkm* if (S0)≠0
016*ijkm*	JSP	*exp*	Branch to *ijkm* if (S0) is positive, includes (S0)=0
017*ijkm*	JSM	*exp*	Branch to *ijkm* if (S0) is negative

Section 7 of this manual has additional information on the use of S registers by instructions.

T REGISTERS

The CPU has sixty-four 64-bit T registers used as intermediate storage for S registers. Data is transferred between T and S registers and between T registers and Central Memory. Transfer of a value between a T register and an S register requires only 1 CP. T registers reference Central Memory through block read and block write instructions. Block transfers occur at a maximum rate of one word per CP. No reservations are made for T registers and no instructions are issued during block transfers to and from T registers.

Only one T register receives data during each CP. Issue of an instruction is delayed if it causes data to arrive at the T registers concurrently with data already being processed and scheduled to arrive from another source.

In this manual, T registers are referred to by the letter T followed by a 2-digit octal number ranging from 00 through 77. Instructions reference T registers by specifying the octal number as the *jk* designator as described in section 7.

VECTOR REGISTERS

Figure 5-3 illustrates registers and functional units used for vector operations. Vector registers and vector functional units are described in the following paragraphs.

V REGISTERS

The major computational registers of the CPU are eight V registers, each having 64 elements. Each V register element has 64 bits. When associated data is grouped into successive elements of a V register, the register quantity is 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 V 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 during each CP and as each operation is performed, the result is delivered to successive elements of the result V register. 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.

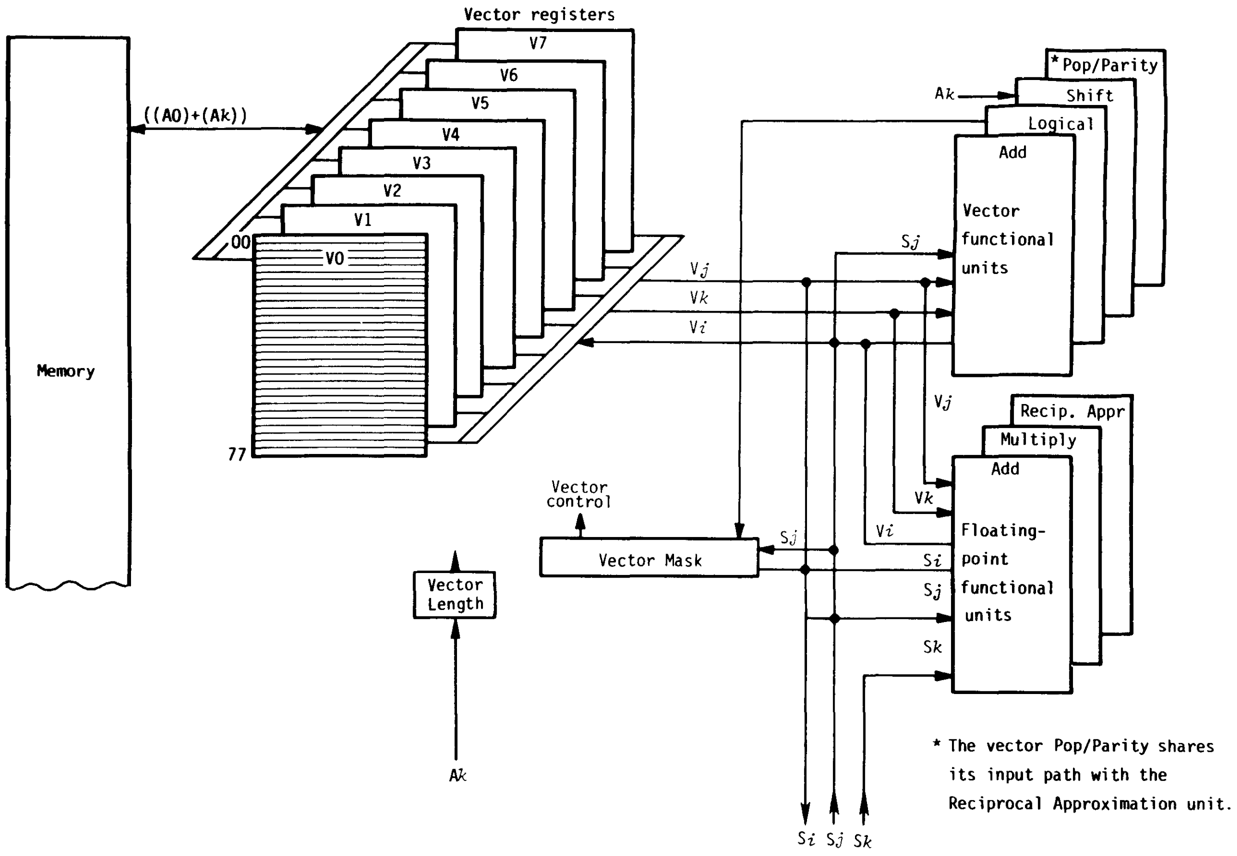


Figure 5-3. Vector registers and functional units

Contents of a V register are transferred to or from Central Memory in a block mode by specifying a first word address in Central Memory, an increment or decrement for the Central Memory address, and a vector length. Transfer then proceeds beginning with the first element of the V register at a maximum rate of one word per CP, depending on bank conflicts.

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

Since many vectors exceed 64 elements, longer vectors are 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. A programmer can choose to construct the vector loop code in a number of ways. The processing of long vectors in FORTRAN is handled by the compiler and is transparent to the programmer.

A V register receiving results can also supply operands to a subsequent operation. Using a register as both a result and operand registsr in two different operations allows for chaining together of two or more vector operations, and two or more results can be produced per CP. Chained operations are detected automatically by the CPU and are not explicitly specified by the programmer. A programmer can reorder certain code segments to enable chained operations.

A conflict can occur between vector and scalar operations involving floating-point operations and memory 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 (VL) elements are processed.

Parallel vector operations are processed by:

- Using different functional units and all different V registers
- Using the result stream from one V register simultaneously as the operand to another operation using a different functional unit (chain mode)

Parallel operations on vectors allow generating two or more results per CP. Most vector operations use two V registers or one S and one V register as operands. Exceptions are vector shifts, vector reciprocal, and load or store instructions.

In this manual, V registers are individually referred to by the letter V followed by a number ranging from 0 through 7. Vector instructions reference V registers by specifying the register number as the *i*, *j*, or *k* designator as described in section 7.

Individual elements of a V register are designated in this manual by decimal numbers ranging from 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

Reservation describes the condition of a register in use. When in use, the register is not available for another operation as a result or as an operand register. During execution of a vector instruction, reservations are placed on 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 V register is lifted during chain slot time. Chain slot time is the CP occurring at functional unit time plus 2 CPs. During this CP, the result is available for use as an operand in another vector operation. Chain slot time does not affect the reservation placed on operand V registers. A V register serves 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. The S register can be modified in the next instruction after vector issue without affecting the vector operation. The vector length and scalar operand (if appropriate) of each vector operation are maintained separately from the VL register and scalar register. Vector operations employing different lengths proceed concurrently; however, vector length should normally not be changed between chain operations because chaining implies operations of the same length.

A0 and A*k* registers in a vector memory reference are available for modification immediately after use.

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

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

VECTOR CONTROL REGISTERS

The Vector Length (VL) register and the Vector Mask (VM) register provide control information needed in the performance of vector operations and are described in the following paragraphs.

Vector Length register

The 7-bit Vector Length (VL) register is set to 1 through 100_8 (VL=0 gives VL=100_8) specifying the length of all vector operations performed by vector instructions and the length of the vectors held by the V registers. The VL register controls the number of operations performed for instructions 140 through 177 and is set to an A register value using instruction 0020.

****

CAUTION

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

For example, during a vector sequence the contents of VL are increased to a larger value and a second operation is initiated to chain to the first operation. The user expects the second operation to 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 wanting to use the system in this manner must take care to avoid chained operations. Although there can be applications of the characteristic produced by chained operations with different contents for VL, Cray Research, Inc., takes no responsibility for its use. Chained operation is not assured since I/O or other interrupts can prevent the chain from occurring.

****

Vector Mask register

The Vector Mask (VM) register has 64 bits, each corresponding to a word element in a V register. Bit 2^{63} corresponds to element 0, bit 2^{0} to element 63. The mask is used with vector merge and test instructions to allow operations to be performed on individual vector elements.

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

FUNCTIONAL UNITS

Instructions other than simple transmits or control operations are performed by hardware organizations called functional units. Each functional unit implements an algorithm or portion of the instruction set. Functional units have independent logic except for Reciprocal Approximation and Vector Population Count units (described later in this section), which share some logic. All functional units can operate at the same time.

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

All functional units perform algorithms in a fixed amount of time; delays are impossible once operands are delivered to the unit. Time required from delivery of operands to the functional unit until completion of the calculation is called functional unit time and is measured in CPs.

Functional units are fully segmented. This means a new set of operands for unrelated computation enters a functional unit during each CP even though the functional unit time is more than 1 CP. This segmentation is possible when information arrives at the functional unit and is held in the functional unit or moves within the functional unit at the end of every CP.

Thirteen functional units are identified in this manual and are arbitrarily described in four groups: address, scalar, vector, and floating-point. Each of the first three groups functions 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 M. The fourth group, floating-point, supports either scalar or vector operations and accepts operands from or delivers results to S or V registers. In addition, Central Memory acts like a fourteenth functional unit for vector operations.

ADDRESS FUNCTIONAL UNITS

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 functional unit

The Address Add functional unit performs 24-bit integer addition and subtraction and executes instructions 030 and 031. Addition and subtraction are performed in a similar manner. The twos complement subtraction for instruction 031 occurs when the ones complement of the A*k* operand is added to the A*j* operand. Then a 1 is added in the low-order bit position of the result. No overflow is detected in the functional unit.

The Address Add functional unit time is 2 CPs.

Address Multiply functional unit

The Address Multiply functional unit executes instruction 032 forming 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 not exceeding its data capabilities. The programmer must be careful when multiplying integers in the functional unit because the functional unit does not detect overflow of the product and the most significant portion of the product could be lost.

The Address Multiply functional unit time is 6 CPs.

SCALAR FUNCTIONAL UNITS

Scalar functional units perform operations on 64-bit operands obtained from S registers and, in most cases, deliver 64-bit results to an S register. The exception is the Population/Leading Zero Count functional unit which delivers its 7-bit result to an A register.

Four functional units are exclusively associated with scalar operations and are described below. Three functional units are used for both scalar and vector operations and are described in the subsection on floating-point functional units.

Scalar Add functional unit

The Scalar Add functional unit performs 64-bit integer addition and subtraction and executes instructions 060 and 061. The addition and subtraction are performed in a similar manner. The twos complement subtraction for instruction 061 occurs when the ones complement of the S*k* operand is added to the S*j* operand. Then a 1 is added in the low-order bit position of the result. No overflow is detected in the Scalar Add functional unit.

The Scalar Add functional unit time is 3 CPs.

Scalar Shift functional unit

The Scalar Shift functional unit shifts the entire 64-bit contents of an S register or the double 128-bit contents of two concatenated S registers. Shift counts are obtained from the *jk* portion of the instruction or from an A register. 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 nonzero. Scalar double shifts use the lower 7 bits of the A register for the shift amount. The 7 bits gives 0 to 127 as the possible range. The most significant of the 7 bits is 2^6 and the least significant bit is 2^0. If any bit 2^{23} through 2^7 is nonzero, the shifter returns a 0. For shifts of 64 through 127 the result is also 0 if the *j* designator is 0; that is, S0 is the second register used. All A register shift counts are considered positive, unsigned integers.

The Scalar Shift functional unit executes instructions 052 through 057. Single-shift instructions, 052 through 055, have a functional unit time of 2 CPs. Double-shift instructions, 056 and 057, have a functional unit time of 3 CPs.

Scalar Logical functional unit

The Scalar Logical functional unit manipulates bit-by-bit the 64-bit quantities obtained from S registers. It executes instructions 042 through 051, the mask, and Boolean instructions. Instructions 042 through 051 have a functional unit time of 1 CP.

Scalar Population/Parity/Leading Zero functional unit

This functional unit executes instructions 026 and 027. Instruction 026*ij*0 counts the number of bits in an S register having a value of 1 in the operand and has a functional unit time of 4 CPs. Instruction 026*ij*1 returns a 1-bit population parity count (even parity) of the S*j* register's contents. Instruction 027 counts the number of bits of 0 preceding a 1 bit in the operand and has a functional unit time of 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.

VECTOR FUNCTIONAL UNITS

Most vector functional units perform operations on operands obtained from two V registers or from a V register and an S register. The Reciprocal Approximation, Shift, and Population/Parity functional units, which require only one operand, are exceptions. Results from a vector functional unit are delivered to a V register.

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

Four functional units described in this section are exclusively associated with vector operations. Three functional units are associated with both vector operations and scalar operations and are described in the subsection on floating-point functional units. Also, the recursive characteristic of vector functional units is described in the subsection on floating-point functional units since it is used primarily with the floating-point functional units. When a floating-point functional unit is used for a vector operation, the general description of vector functional units given in the subsection applies.

Vector functional unit reservation

A functional unit engaged in a vector operation remains busy during each CP and cannot participate in other operations. In this state, the functional unit is reserved. Other instructions requiring 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 functional unit

The Vector Add functional 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. Addition and subtraction are performed in a similar manner. For subtraction operations (156 and 157), the Vk operand is complemented prior to addition and a 1 is added into the low-order bit position of the result. No overflow is detected by the unit.

The Vector Add functional unit time is 3 CPs; chain slot time is 5 CPs.

Vector Shift functional unit

The Vector Shift functional 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 and are end off with zero fill.

Vector single and double shifts use the lower 7 bits of the A register for the shift amount. The 7 bits gives 0 to 127 as the possible range. The most significant of the 7 bits is 2^6 and the least significant bit is 2^0. If any bit 2^{23} through 2^7 is nonzero, the shifter returns a 0. For shifts of 64 through 127 the result is also 0 if the *j* designator is 0. All shift counts are considered positive unsigned integers.

The Vector Shift functional unit executes instructions 150 through 153. The functional unit time is 4 CPs; chain slot time is 6 CPs.

Vector Logical functional unit

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

The Vector Logical functional unit time is 2 CPs; chain slot time is 4 CPs.

Vector Population/Parity functional unit

The Vector Population/Parity functional unit counts the 1 bits in each element of the source V register. The total number of 1 bits is the population count. This population count can be an odd or an even number; only the low-order bit is significant if calculating parity.

Instructions 174*ij*1 (vector population count) and 174*ij*2 (vector population count parity) use the same operation code as the vector reciprocal approximation instruction. Some restrictions for the Reciprocal Approximation functional unit also apply for vector population instructions (see subsection on Reciprocal Approximation). 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 Vector Population/Parity functional unit time is 6 CPs; chain slot time is 8 CPs.

FLOATING-POINT FUNCTIONAL UNITS

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

Information on floating-point out-of-range conditions is contained in the subsection on floating-point arithmetic.

Floating-point Add functional unit

The Floating-point Add functional unit performs addition or subtraction of 64-bit operands in floating-point format and executes instructions 062, 063, and 170 through 173. A result is normalized even when operands are unnormalized. (Normalized floating-point numbers are described in the subsection on floating-point arithmetic.) Out-of-range exponents are detected as described in the subsection on floating-point arithmetic.

Floating-point Add functional unit time is 6 CPs; chain slot time is 8 CPs.

Floating-point Multiply functional unit

The Floating-point Multiply functional unit executes instructions 064 through 067 and 160 through 167. These instructions provide for full-precision 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 can be rounded or not rounded.

Input operands are assumed to be normalized. The Floating-point Multiply functional unit delivers a normalized result only if both input operands are normalized.

Out-of-range exponents are detected as described in the subsection on floating-point arithmetic. However, if both operands have zero exponents, the result is considered as an integer product, is not normalized, and is not considered out-of-range. This case provides a fast method of computing a 48-bit integer product, although the operands in this case must be shifted before the multiply operation.

Floating-point Multiply functional unit time is 7 CPs; chain slot time is 9 CPs.

Reciprocal Approximation functional unit

The Reciprocal Approximation functional unit finds the approximate reciprocal of a 64-bit operand in floating-point format. The unit executes instructions 070 and 174*ij*0. Since the Vector Population/Parity functional 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 correct. 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. Out-of-range exponents are detected as described under Floating-point Arithmetic.

The Reciprocal Approximation functional unit time is 14 CPs; chain slot time is 16 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* and/or *k* is desirable under certain circumstances. Such action facilitates reducing 64 elements to only a few. The number of terms generated by the partial reduction is determined by the number of values that are in process in a functional unit at one time and equal to the functional unit time (in CPs)+2.

****

CAUTION

Cray Research cautions against using a vector register as both a result and an operand because of the recursive characteristic of vector processing. Also, where upward compatibility is an issue, note that vector recursion is not available on all Cray Research, Inc., computers.

****

The recursive characteristic is introduced into the vector processing because of the handling of element counters. At the beginning of a vector operation, *i*, *j*, *k* element counters are set to 0. In a nonrecursive operation, the operand register element counter begins incrementing immediately while the element counter for the result register is held at 0 until functional unit time + 2 CPs. In a recursive operation (when an operand register is the same as the result register), the element counter for the operand/result register is held at 0. The element counter does not begin incrementing until the first result arrives from the functional unit at functional unit time + 2 CPs. This counter then begins to advance by 1 each CP.

Note that until functional unit time + 2, 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 (if not the same) immediately begins advancing by 1 on each successive CP, sending the contents of elements 0, 1, 2, ... on successive CPs.

Thus, the first functional unit time + 2 elements of the operand/result register contain results based on 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 operand used in calculating results for the next functional unit time + 2 elements (second group). The third group contains results based on the results delivered to the second group and so on until the final group of elements is generated as determined by the vector length.

This recursive characteristic of vector processing applies to any vector operation, arithmetic or logical. The value initially placed in element 0 of the operand/result register depends on the operation being performed. For example, when using the Floating-point Add functional unit recursively, element 0 of the operand/result register is usually set to an initial value of 0.0; when using the Floating-point Multiply functional unit recursively, element 0 of the operand/result register is usually set to an initial value of 1.0. In a recursive operation (except for shifts), only element 0 is used of the V register used in the operation. All other elements are replaced before they are used as an operand.

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 (VL) register contains 64.

A floating-point add instruction (171212 or 171221) 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 with 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) = (V200)                                                                + (V100)
V201 = (V200) + (V101) = (V200)                                                                + (V101)
V202 = (V200) + (V102) = (V200)                                                                + (V102)
V203 = (V200) + (V103) = (V200)                                                                + (V103)
V204 = (V200) + (V104) = (V200)                                                                + (V104)
V205 = (V200) + (V105) = (V200)                                                                + (V105)
V206 = (V200) + (V106) = (V200)                                                                + (V106)
V207 = (V200) + (V107) = (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)
       \____/                   \_________________________________________________________________/
          ^                                                 ^
          |_____________________ EQUALS ____________________|
```

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 represented in the registers as illustrated in figure 5-4. The Address Add and Address Multiply functional units perform 24-bit arithmetic. The Scalar Add and the Vector Add functional units perform 64-bit arithmetic.

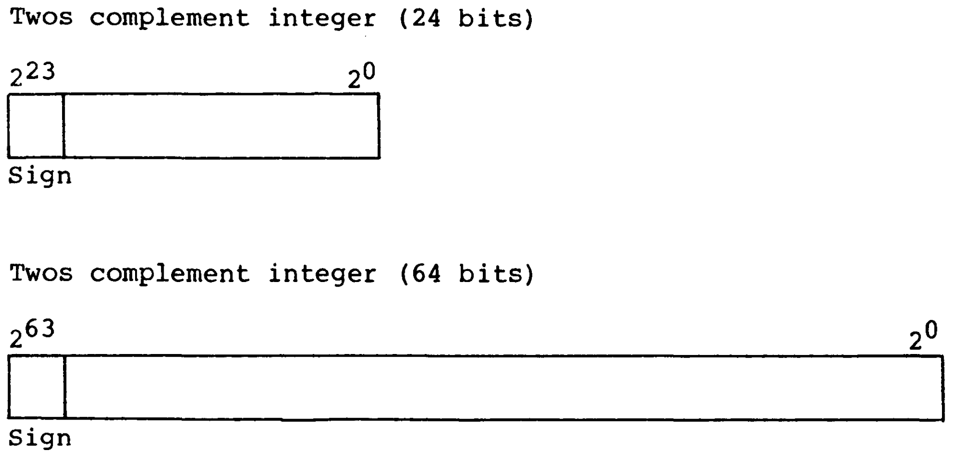

Twos complement integer (24 bits)
2^{23}
2^0
Sign
Twos complement integer (64 bits)
2^{63}
2^0
Sign

Figure 5-4. Integer data formats

Multiplication of two scalar (64-bit) integer operands is accomplished by using the floating-point multiply instruction and one of the two methods that follows. The method used depends on the magnitude of the operands and the number of bits to contain the product.

If the operand bits are nonzero only in the 24 least significant bits, the two integer operands can be multiplied by shifting them each left 24 bits before the multiply operation. The Floating-point Multiply functional unit recognizes the conditions where both operands have zero exponents as a special case. The Floating-point Multiply functional unit 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 5-6.) If the operand coefficients are generated other than by shifting, so the low-order 24 bits would be nonzero, the low-order 48 bits of the product could have been nonzero, and the high-order 48 bits (the return part) could be one larger than expected because a truncation compensation constant is added during the multiply.

If the operands are greater than 24 bits, multiplication is done by forming multiple partial products and then shifting and adding the partial products.

Division is done by algorithm; the particular algorithm used depends on the number of bits in the quotient. The quickest and most frequently used method is to convert the numbers to floating-point format and then use the floating-point functional 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 (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 5-5. Since the coefficient is signed magnitude, it is not complemented for negative values.

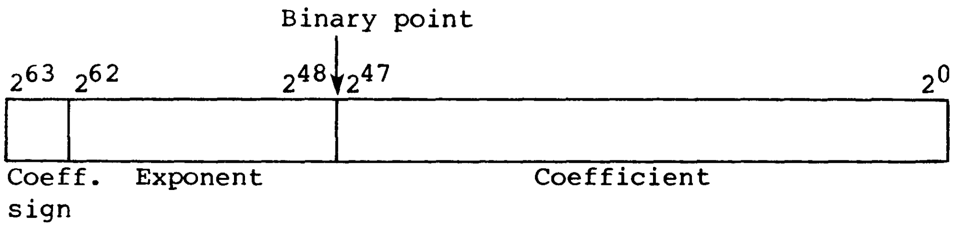


Figure 5-5. Floating-point data format

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 (note the negative range is one larger):

2^{-20000_8} through 2^{+17777_8}

In terms of decimal values, the floating-point format of the CPU allows the accurate expression of numbers 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 where a negative 0 is one operand going to the Floating-point Multiply functional unit.

Normalized floating-point numbers, floating-point range errors, double precision numbers, and the addition, multiplication, and division algorithms are described in this subsection.

Normalized floating-point numbers

A nonzero floating-point number is normalized if the most significant bit of the coefficient is nonzero. This condition implies the coefficient has been shifted as far left as possible and the exponent adjusted accordingly. Therefore, the floating-point number has no leading zeros in the coefficient. The exception is that a normalized floating-point zero is all zeros.

When a floating-point number is created by inserting an exponent of 40060_8 into a 48-bit integer word, 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 S*j* field of an instruction, an operand in S*k* is normalized using the 062*i*0*k* instruction. S*i*, which can be S*k*, contains the normalized result.

The 170*i*0*k* instruction normalizes V*k* into V*i*.

Floating-point range errors

Overflow of the floating-point range is indicated by an exponent value of 60000_8 or greater 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.

The Cray Operating System (COS) keeps a bit in a table to indicate the condition of the mode bit. System software manipulates the mode bit and uses the table bit to indicate how the mode should be left for the user. Therefore, the user usually needs to request that COS set the appropriate value in the table if the user changes the mode.

Floating-point range error conditions are detected by the floating-point functional units as described in the following paragraphs.

Floating-point Add functional 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. This condition sets the Floating-point Error flag with an exponent of 60000_8 being sent to the result register along with the computed coefficient, as in the following example:

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

NOTE

If the result of an add or subtract operation is less than the machine minimum, the error is suppressed (even though both operands have exponents greater than or equal to 60000_8) because the machine minimum takes precedence in error detection.

Floating-point Multiply functional unit - Out-of-range conditions are tested before normalizing. In the Floating-point Multiply functional unit, if the exponent of either operand is greater than or equal to 60000_8 or if the biased sum minus 1 of the two unbiased 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 operand is less than the machine minimum, the error is suppressed (even though the other operand can be out of range) because the operand that is less than the machine minimum takes precedence in error detection.

If both incoming exponents are equal to 0, the operation is treated as an integer multiply. 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, the operands should be considered as 24-bit integers in bits 2^{47} through 2^{24}. In figure 5-6, operand 1 is 4 and operand 2 is 6, producing a 48-bit result of 30_8. Bit 2^{63} obeys the usual rules for multiplying signs and the result is a sign and magnitude integer. Note the form of integers (see figure 5-4) accepted by the integer add and subtract and expected by the software is twos complement not sign and magnitude. Therefore, negative products must be converted.

If bits 2^0 through 2^{23} in operands 1 and 2 of figure 5-6 have any 1 bits, the product might be one too large (2^0) because a truncation compensation constant is added during the multiply process. (The following paragraphs discuss the truncation constant and its use.) The size of the shaded area in operands 1 and 2 (see figure 5-6) does not need to be the same for both operands. To get a correct product, the only requirement is that the sum of the number of bits in the shaded area is 48 bits or more. If the sum is more than 48 bits, the binary point in the product is the number of places to the left that the sum is in excess of 48 (that is, assuming the operand binary points are at the left boundary of the shaded area).

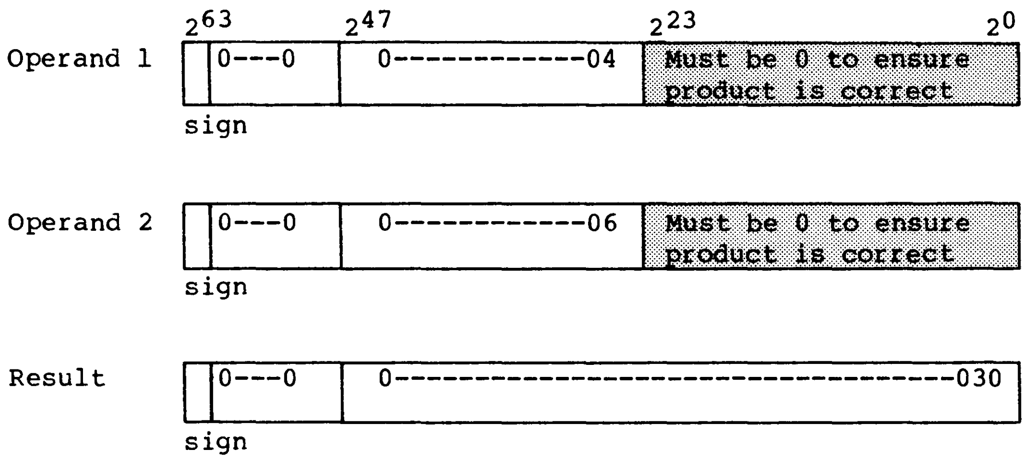


Figure 5-6. Integer multiply in Floating-point Multiply functional unit

Floating-point Reciprocal Approximation functional unit - For the Floating-point Reciprocal Approximation functional 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 and the computed coefficient are sent to the result register.

Double-precision numbers

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

Addition algorithm

Floating-point addition or subtraction is performed in a 49-bit register (see figure 5-7). Trial subtraction of the exponents selects 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 the 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 the result is less than the machine minimum, the error is suppressed.

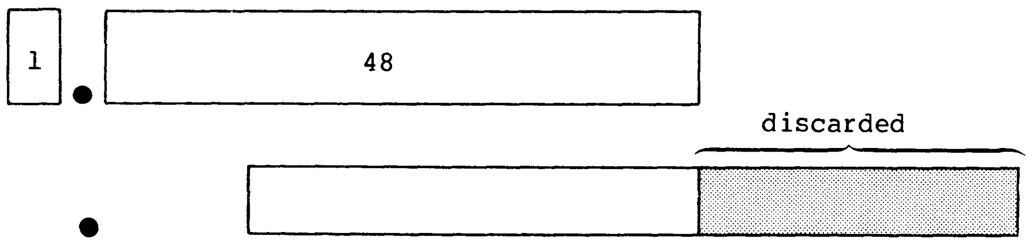


Figure 5-7. 49-bit floating-point addition

The Floating-point Add functional unit normalizes any floating-point number within the format of the CRAY-1 M floating-point number system. The functional unit right shifts 1 or left shifts up to 48 per result to normalize the result.

One zero operand and one valid operand can be sent to the Floating-point Add functional unit, and the valid operand is sent through the unit normalized. Concurrently, the functional unit checks for overflow and/or underflow; underflow results are not flagged as errors.

Multiplication algorithm

The Floating-point Multiply functional unit has the two 48-bit coefficients as input into a multiply pyramid (see figure 5-8). If the coefficients are both normalized, then a full product is either 95 bits or 96 bits, depending on the value of the coefficients. A 96-bit product is normalized as generated. A 95-bit product requires a left shift of one to generate the final coefficient. If the shift is done, the final exponent is reduced by one to reflect the shift. The following discussion and the power of two designators used assumes that the product generated is in its final form; that is, no shift was required. On the CRAY-1 M, 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 average value of this truncation is

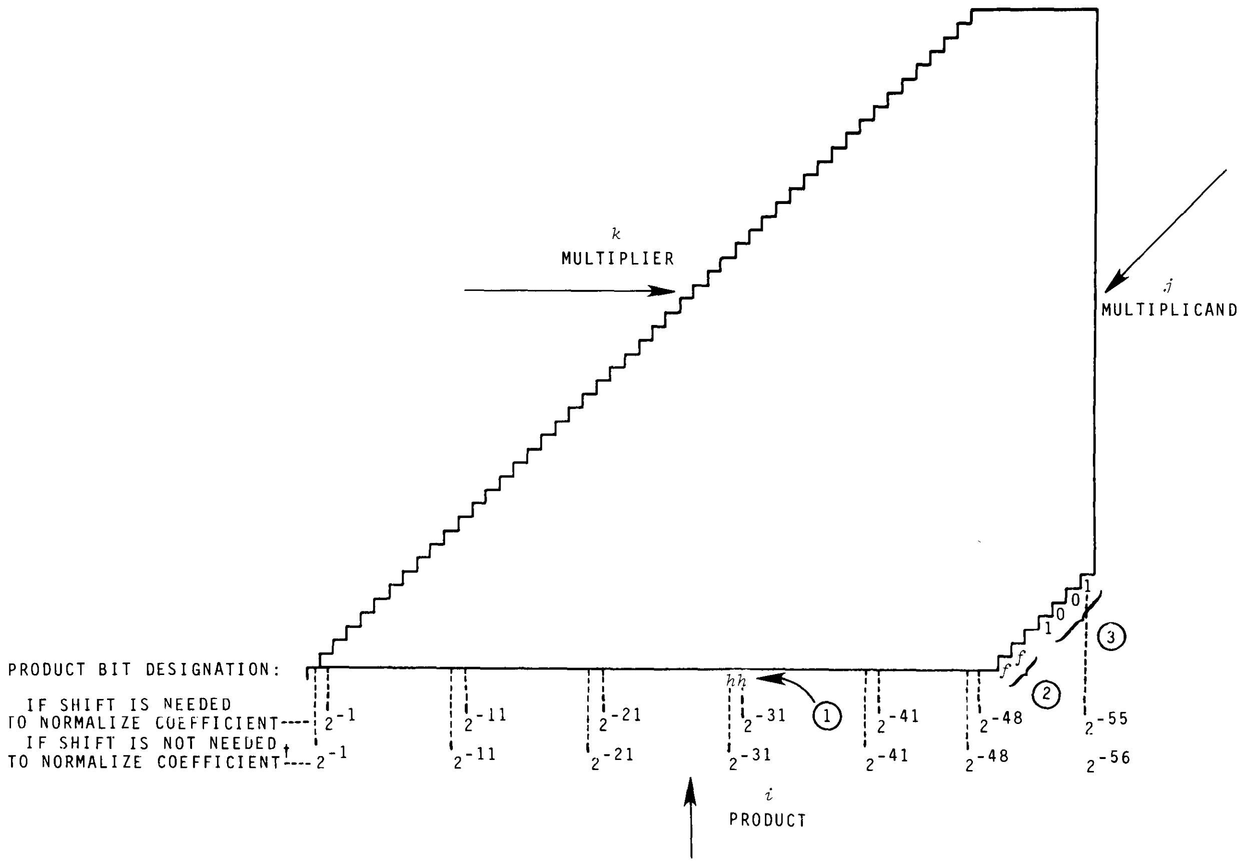


Figure 5-8. Floating-point multiply partial-product sums pyramid

① $hh = 11_2$ for half-precision rounded multiply, 00_2 for full-precision rounded or full-precision unrounded multiply
② $ff = 11_2$ for full-precision rounded multiply, 00_2 for half-precision rounded or full-precision unrounded multiply
③ Truncation compensation constant, 1001_2 used for all multiplies

† Bit designations are used in the explanation of the Floating-point Multiply functional unit operation.

9.25×2^{-56}, which was determined by adding all carries produced by all possible combinations that could be truncated and dividing the sum by the number of possible combinations. Nine carries are injected at the 2^{-56} position to compensate for the truncated bits. The effect of the truncation without compensation is at most a result coefficient one smaller than expected. With compensation, the results range from one too large to one too small in the 2^{-48} bit position with approximately 99 percent of the values having zero deviation from what would have been generated had a full 96-bit pyramid been present. The multiplication is commutative; that is, A times B equals B times A.

Rounding is optional where truncation compensation is not. The rounding method used adds a constant so that it is 50 percent high ($.25 \times 2^{-48}$; high) 38 percent of the time and 25 percent low ($.125 \times 2^{-48}$; low) 62 percent of the time resulting in near zero average rounding error. 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 the 29 most significant bits of the normalized result are transmitted back.

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

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

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

With a full 96-bit pyramid and rounding equal to one-half the least significant bit, the variation would be expected to be:

$$-0.5 \times 2^{-48} \text{ to } +0.5 \times 2^{-48}$$

Division algorithm

The CRAY-1 M performs floating-point division through reciprocal approximation, facilitating hardware implementation of a fully segmented functional unit. Because of this segmentation, operands enter the reciprocal unit during each CP. In vector mode, results are produced at a 1-CP rate and are used in other vector operations during chaining because all functional units in the CRAY-1 M 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 5-9). 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 can be repeated using x_1 to find x_2, etc.

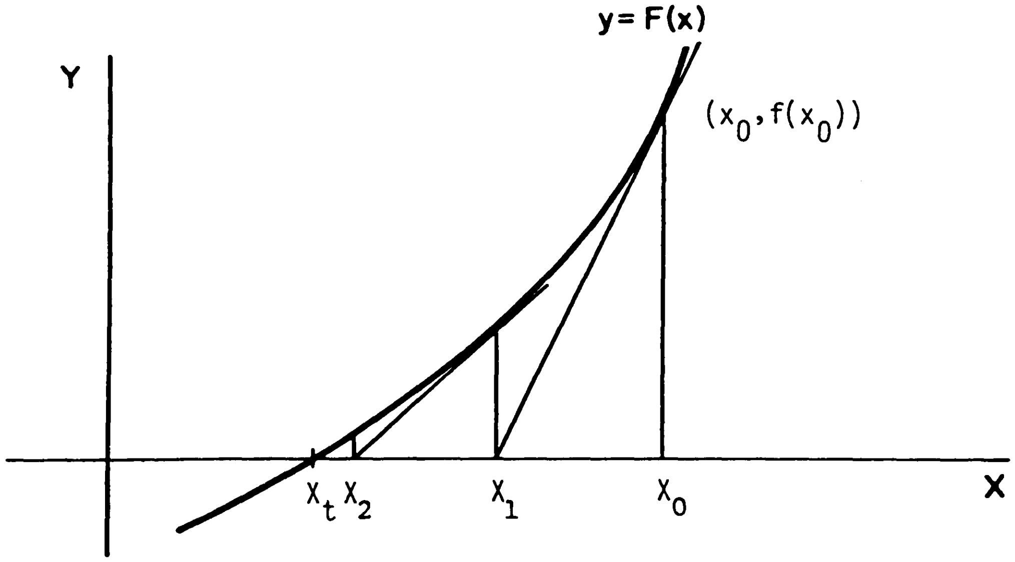


Figure 5-9. Newton's method

Derivation of the division algorithm

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

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

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 then the value of x that makes F(x) equal 0. This is the case when $x = x_t$, therefore $F(x_t)$ in the preceding equation can be replaced by 0, giving the following approximation:

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

Notice that $x_t - x_i$ is the correction 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:

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

that is, $- \frac{F(x_i)}{F'(x_t)}$ is the approximate correction.

If this quantity is substituted into the approximation, then:

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

This gives, the following equation:

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

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 is repeated until the answer becomes sufficiently close for practical use. (On the CRAY-1 M, if the correction term is exact, the operation must not be repeated.)

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):

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

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

and applying equation (1),

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

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

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

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

This approximation technique using Newton's method is implemented in the CRAY-1 M. 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 M 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 Approximation functional unit, and its result is accurate to 30 bits.

A fourth iteration uses a special instruction within the Floating-point Multiply functional unit to calculate the correction term. This iteration is used to increase accuracy of the reciprocal unit's answer to full precision. A fifth iteration should not be done.

The division algorithm that computes S1/S2 to full-precision requires the following operations:

S3 = 1/S2	Performed by the Reciprocal Approximation functional unit
S4 = (2 - (S3 * S2))	Performed by the Floating-point Multiply functional unit in iteration mode

S5 = S4 * S3	Performed by the Floating-point Multiply functional unit using full-precision. S5 now equals 1/S2 to 48-bit accuracy.
S6 = S5 * S1	Performed by the Floating-point Multiply functional unit using full-precision rounded

The reciprocal approximation at step 1 is correct to 30 bits. An additional Newton iteration (fourth iteration) at operations 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 are possible.

****

CAUTION

The reciprocal iteration is designed for use once with each half-precision reciprocal generated. If the fourth iteration (the programmed iteration) results in an exact reciprocal or if an exact reciprocal is generated by some other method, performing another iteration results in an incorrect final reciprocal.

****

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

S3 = 1/S2	Performed by the Reciprocal Approximation functional unit
S6 = S1 * S3	Performed by the Floating-point Multiply functional unit in half-precision

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

Another method of computing divisions is as follows:

S3 = 1/S2	Performed by the Reciprocal Approximation functional unit
S5 = S1 * S3	Performed by the Floating-point Multiply functional unit

S4 = (2 - (S3 * S2))	Performed by the Floating-point Multiply functional unit
S6 = S4 * S5	Performed by the Floating-point Multiply functional unit

A scalar quotient is computed in 29 CPs since operations 2 and 3 issue in successive CPs. With this method the correction to reach a full-precision reciprocal is applied after the numerator is multiplied times the half-precision reciprocal rather than before.

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.)

****

CAUTION

The coefficient of the reciprocal produced by the alternate method can be as much as 2×2^{-48} different from the first method described for generating full-precision reciprocals. This difference can occur because one method can round up as much as twice while the other method may not round at all. One round can occur while the correction is generated and the second round can occur when producing the final quotient.

Therefore, if the reciprocals are to be compared, the same method should be used each time the reciprocals are generated. Cray FORTRAN (CFT) uses a consistent method and ensures the reciprocals of numbers are always the same.

****

For example, two 64-element vectors are divided in 3 * 64 CPs plus overhead. (The overhead associated with the functional units for this case is 38 CPs).

LOGICAL OPERATIONS

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

An operation similar to the AND function produces the following results:

Operand 1	1 0 1 0
Operand 2	1 1 0 0
Result	0 1 0 0

The logical product (AND) operation is used for masking operations where the ones specify the bits to be saved. In this variant of the AND function, the zeros specify the bits to be saved (Operand 1 is the mask).

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 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 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 6

INTRODUCTION

The Input/Output section of the CRAY-1 M Series mainframe contains one or two 100 Mbytes per second channels and four control channels, each with a maximum transfer rate of 6 Mbytes per second. A 100 Mbytes per second channel is 64 bits wide and has one input channel and one output channel. Each 6 Mbytes per second channel pair is 16 bits wide and has an input channel and an output channel. This section describes a 100 Mbytes per second channel, the 6 Mbytes per second channels, Master Clear sequences, and memory accessing, lockouts, conflicts, request conditions, and addressing.

DATA TRANSFER FOR I/O SUBSYSTEM

The standard 100 Mbytes per second channel transfers data between Central Memory of the CRAY-1 M mainframe and the Buffer I/O Processor (BIOP) of the I/O Subsystem. If present, a second 100 Mbytes per second channel transfers data between Central Memory and the Disk I/O Processor (DIOP) or the Auxiliary I/O Processor (XIOP); however, software is currently not available to transfer data between Central Memory and the XIOP. A 100 Mbytes per second channel 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 in Central Memory.

The CPU side of a 100 Mbytes per second channel uses a pair of 16-word buffers to stream the data out of Central Memory and another pair to stream data into Central Memory. On output, as one buffer block is being sent to the I/O Processors (IOPs), the other buffer is filling from Central Memory. Similarly, on input, one buffer block is filling from an IOP while the other is transmitting to Central Memory.

At the IOP side of the 100 Mbytes per second channel, data passing into Local Memory (an I/O Processor's memory) is double-buffered and disassembled into 16-bit parcels. The channel side passing data from Local Memory simply assembles the 16-bit parcels into 64-bit words for transmission to the CPU.

The instruction fetch, exchange sequence, and normal 6 Mbytes per second channel memory requests take precedence over a 100 Mbytes per second channel/Central Memory request. Data is sent in blocks, with 16 words as the normal block length (16 banks). Each block transfer keeps Central Memory busy for 11 clock periods (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.

An IOP controls the 100 Mbytes per second channel linking it with Central Memory. The IOP initiates all data transfers on the channel and performs all error processing required for the channel. There are no CPU instructions for the 100 Mbytes per second channel. Programming details for the 100 Mbytes per second channel are described in the CRAY I/O Subsystem Reference Manual, publication HR-0030.

DATA TRANSFER FOR THE SOLID-STATE STORAGE DEVICE

The Solid-state Storage Device (SSD) requires a 100 Mbytes per second channel, a standard 6 Mbytes per second channel, and a special controller to connect to the CRAY-1 M mainframe. Port 2 of the SSD connects with the CRAY-1 M CPU. The CPU controls the SSD by communicating transfer commands to the controller using a standard 6 Mbytes per second channel adapted for use with the SSD. The controller sends the appropriate control signals, which start transfer, to each end of the 100 Mbytes per second channel link. Programming details for the SSD are described in the Solid-state Storage Device (SSD) Reference Manual, CRI publication HR-0031.

6 MBYTES PER SECOND CHANNELS

The 6 Mbytes per second channels have three basic types of control logic:

- 16-bit asynchronous; used for front-end interfaces; the standard CRAY-1 mainframe 6 Mbytes per second channel
- 16-bit high-speed asynchronous
- 16-bit SSD control

Each type of 6 Mbytes per second channel has the same electrical interface to the I/O cable but differs in timing, protocol, and data rates.

CHANNEL GROUPS

6 Mbytes per second channels are numbered octally 2 through 11 and are divided into four groups as follows.

Group 1 input channels:	2, 6
Group 2 output channels:	3, 7
Group 3 input channels:	4, 10
Group 4 output channels:	5, 11

I/O INSTRUCTIONS

The instructions used with 6 Mbytes per second channels are:

0010*jk*	Set the Current Address (CA) register for the channel indicated by (A*j*) to (A*k*) and activate the channel
0011*jk*	Set the Limit Address (CL) register for the channel indicated by (A*j*) to (A*k*)
0012*jx*	Clear the interrupt flag and error flag for the channel indicated by (A*j*)
033*i*0*x*	Transmit channel number to A*i*
033*ij*0	Transmit address of channel (A*j*) to A*i*
033*ij*1	Transmit error flag of channel (A*j*) to A*i*

The SSD redefines several of these instructions. For additional information, refer to the Solid-state Storage Device (SSD) Reference Manual, CRI publication HR-0031.

6 MBYTES PER SECOND CHANNEL OPERATION

Each input or output channel directly accesses 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 6-1. In both input and output operations, parcel 0 is always transferred first.

Each input or output 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 (CA) register, and a channel Limit Address (CL) register.

Three control signals (Ready, Resume, and Disconnect) coordinate transfer of parcels over the channels. The method of coordination varies among the channel types. In addition to the three control signals, either the input or output channel of a pair has a Master Clear line. Appendix E describes the signal sequence of a 6 Mbytes per second channel.

Table 6-1. 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 are caused by the following:

- On all output channels, if (CA) becomes equal to (CL), then for each channel type 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 asynchronous	Disconnect received and channel active, or CA is equal to CL and channel active
16-bit high-speed asynchronous	Disconnect received and channel active

- External device disconnect is received on any input channel and channel is active

- Channel error condition occurs (described later in this section)

The number of the channel causing an interrupt can be determined by using instruction 033, which reads into A*i* 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 when an interrupt from the next highest priority channel, if present, is sensed.

INPUT CHANNEL PROGRAMMING

To start an input operation, the CPU program:

1. Sets the channel limit address to the last word address + 1 (LWA+1). (See figure 6-1.)

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

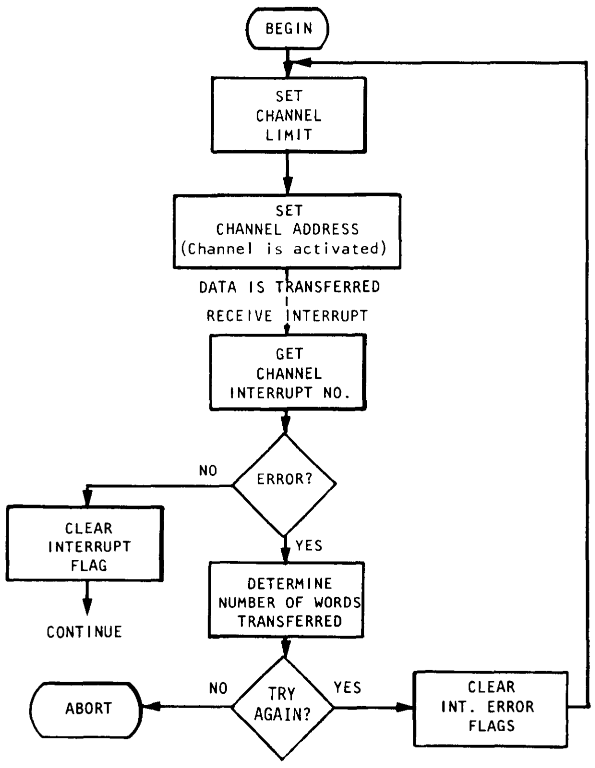


Figure 6-1. Basic I/O program flowchart

Setting the current address causes the Channel Active flag to 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 CA register. When the word is accepted by memory, the current address is advanced by 1.

The external transmitting device sends a Disconnect signal to indicate end of the transfer. When the Disconnect signal 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 signal is received or when an error condition is detected. Setting the interrupt flag deactivates the input channel.

INPUT CHANNEL ERROR CONDITIONS

Input channel error conditions can occur at a parcel level (parity error) or channel level (unexpected Ready signal). These error conditions are described below.

Parity error

When a parcel in error occurs on either a 16-bit asynchronous channel or a 16-bit high-speed asynchronous channel, the Parity Fault flag sets immediately. The Parity Fault flag does not generate an interrupt but is saved and sets the error flag when a disconnect occurs. Therefore, the program checks the state of the error flag when an interrupt is honored.

Unexpected Ready signal

On a 16-bit asynchronous channel if a Ready signal is received when the channel is not active, the Ready condition is saved until the channel is activated, then one of three conditions occurs depending on the module type: an error condition occurs; and interrupt is generated; or the Ready condition is saved. If the third condition occurs, a Resume signal is sent; no error flag is set and no interrupt request is generated.

If a Ready signal 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 is equal to CL (an extra Ready), the error flag is set. An interrupt request is generated, but no Resume signal 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) is executed on the interrupting channel to clear the external device.

If an unexpected Ready signal is received during a memory reference on a 16-bit high-speed asynchronous channel, the normal burst of four pulses of the Resume signal 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 is equal to CL when the unexpected Ready signal arrives, no Resume pulses are sent; the data is not sampled; and the error flag is set to generate an interrupt.

A Ready signal is not expected when the 16-bit synchronous channel is inactive, or when CA is equal to CL, or after the first Ready signal 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:

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

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

Setting the current address causes the Channel Active flag to set. The channel reads the first word from memory addressed by the contents of the CA 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, comparing the contents of the CA register and the CL register. If they are equal, the operation is complete as soon as the last parcel transfer is finished.

OUTPUT CHANNEL ERROR CONDITIONS

The interrupt flag also sets if an error is detected. The only error that an output channel detects is a Resume signal received when the channel is inactive. No external response is generated.

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:

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

SEQUENCE FOR ASYNCHRONOUS CHANNELS

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

1.	0012*jk*	CI,A*j*		Clear <u>output</u> channel to ensure CPU activity on the channel pair has stopped.
2.	0012*jk*	CI,A*j*		Clear <u>input</u> channel to ensure external activity on the channel pair has stopped.
3.	0011*jk*	CL,A*j*	A*k*	Set <u>input</u> channel limit to an arbitrary value.
4.	0010*jk*	CA,A*j*	A*k*	Set <u>input</u> channel current address equal to the same value. Instruction 0010*jk* initiates the Master Clear signal.
5.	0012*jk*	CI,A*j*		Clear <u>input</u> channel. Instruction 0012*jk* stops the input channel activity just initiated.
6.	Delay 1			Device dependent. Delay 1 determines the duration of the Master Clear signal.
7.	0011*jk*	CL,A*j*	A*k*	Set <u>input</u> channel limit. This value can be the same value as used in steps 3 and 4 and turns off the Master Clear signal.
8.	Delay 2			Device dependent. Delay 2 allows time for initialization activities in the attached device to complete.

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

SEQUENCE FOR HIGH-SPEED ASYNCHRONOUS CHANNELS

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

1.	0012*jk*	CI,A*j*		Clear output channel interrupt to ensure CPU activity on the channel pair has stopped.
2.	0012*jk*	CI,A*j*		Clear input channel interrupt to ensure external activity on the channel pair has stopped.
3.	0011*jk*	CL,A*j*	A*k*	Set output channel limit to an arbitrary value.
4.	0010*jk*	CA,A*j*	A*k*	Set output channel current address equal to the same value. Instruction 0010*jk* initiates the Master Clear signal.
5.	0012*jk*	CI,A*j*		Clear output channel. Instruction 0012*jk* stops the output channel activity just initiated.
6.	Delay 1			Device dependent. Delay 1 determines the duration of the Master Clear signal.
7.	0011*jk*	CL,A*j*	A*k*	Set output channel limit. Instruction 0011*jk* can be the same value as used in steps 3 and 4, ands turns off the Master Clear signal.
8.	Delay 2			Device dependent. Delay 2 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 should be a minimum of 20 CPs.

MEMORY ACCESS

Each of the four channel groups is assigned a time slot (see figure 6-2) that is scanned once every 4 CPs for a memory request. The lowest numbered channel in the group has the highest priority. A memory request, accepted or rejected, causes the requesting channel to miss the next two time slots. Therefore, any given channel requests a memory reference only once every 12 CPs. However, another channel in the same group as a channel that has just made a memory request causes 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 CP.

I/O LOCKOUT

An I/O memory request is 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 (maximum of 64 words).

Exchange sequences and instruction fetch sequences can also cause lockouts.

MEMORY BANK CONFLICTS

Memory bank conflicts are tested for CPU scalar references. All other memory references (block transfers, I/O memory references, 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. When four I/O reference requests are made but none are honored, CPU scalar references are held off for one I/O memory reference.

Each memory bank can accept a new request every 7 CPs (for scalar references). To test for a memory bank conflict, the 4 low-order bits[†] of the memory address move through six registers staying 1 CP in each register. The first register is rank A, the second is rank B, the third is rank C, the fourth is rank D, the fifth is rank E, and the sixth is rank F. In the seventh CP, the address is placed in the memory address register.

† 3 bits for 8-bank phasing; refer to section 3 of this publication.

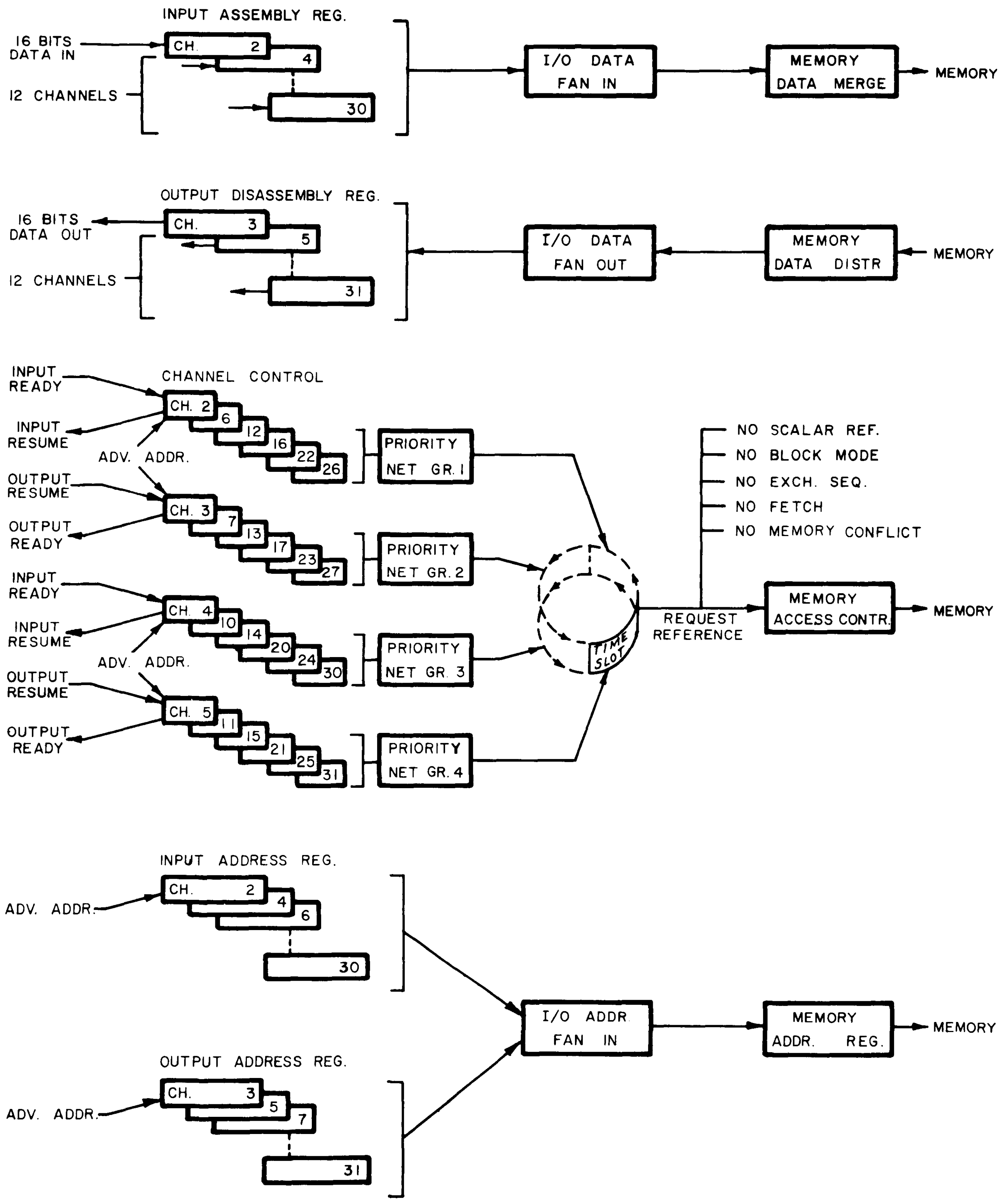


Figure 6-2. Channel I/O control

I/O MEMORY CONFLICTS

Before allowing an I/O request to reference memory, a check is made to ensure no block transfer, exchange sequence, or instruction fetch sequence is in progress and no address or scalar instruction requiring a memory reference is in execution. If any of these conditions exists, the I/O request is blocked and is resubmitted 12 CPs later. The fourth time an I/O request is resubmitted without being honored, scalar references (address or scalar instructions requiring memory) will be held in CIP to allow the I/O request to reference memory.

I/O MEMORY REQUEST CONDITIONS

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

- I/O request
- Memory quiet or three previous I/O requests with none being honored
- No fetch request
- No block transfer instructions 034 through 037 (between memory and B or T registers) or block transfer instructions 176, or 177 (between memory and V registers) in process
- No exchange sequence
- No instruction 033 request for channel status information (not a memory conflict)

I/O MEMORY ADDRESSING

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

CPU INSTRUCTIONS 7

INSTRUCTION FORMAT

Each instruction used in a CRAY-1 M computer 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 0 through 3 from left to right and can be addressed in branch instructions. A 2-parcel instruction begins in any parcel of a word and can span a word boundary. For example, a 2-parcel instruction beginning in the fourth parcel of a word ends in the first parcel of the next word. No padding to word boundaries is required. Figure 7-1 illustrates the general form of instructions.

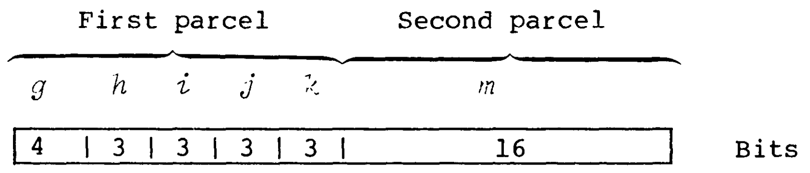


Figure 7-1. General form for instructions

Four variations of this general format use the fields differently; two forms are 1-parcel formats and two are 2-parcel formats. The formats of these four variations are described below.

1-PARCEL INSTRUCTION FORMAT WITH DISCRETE *j* AND *k* FIELDS

The most common of the 1-parcel instruction formats uses the *i*, *j*, and *k* fields as individual designators for operand and result registers (see figure 7-2). The *g* and *h* fields define the operation code. The *i* field designates a result register and the *j* and *k* fields designate operand registers. Some instructions ignore one or more of the *i*, *j*, and *k* fields. The following types of instructions use this format.

- Arithmetic
- Logical
- Double shift
- Floating-point constant

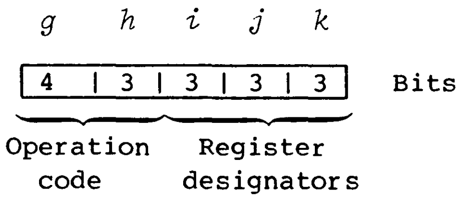


Figure 7-2. 1-parcel instruction format with discrete *j* and *k* fields

1-PARCEL INSTRUCTION FORMAT WITH COMBINED *j* AND *k* FIELDS

Some 1-parcel instructions use the *j* and *k* fields as a combined 6-bit field (see figure 7-3). The *g* and *h* fields contain the operation code, and the *i* field is generally a destination register identifier. The combined *j* and *k* fields generally contain a constant or a B or T register designator. The branch instruction 005 and the following types of instructions use the 1-parcel instruction format with combined *j* and *k* fields.

- Constant
- B and T register block memory transfer
- B and T register data transfer
- Single shift
- Mask

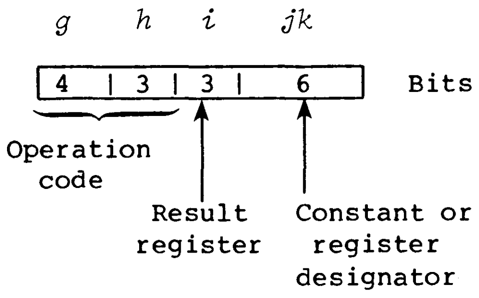


Figure 7-3. 1-parcel instruction format with combined *j* and *k* fields

2-PARCEL INSTRUCTION FORMAT WITH COMBINED *j*, *k*, AND *m* FIELDS

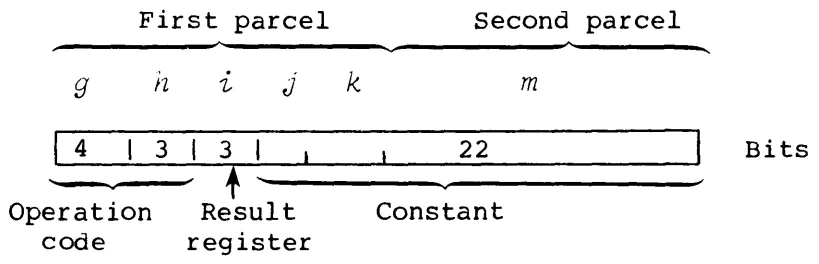
The instruction type for a 22-bit immediate constant uses the combined *j*, *k*, and *m* fields to hold the constant. The 7-bit *gh* field contains an operation code, and the 3-bit *i* field designates a result register. The instruction type using this format transfers the 22-bit *jkm* constant to an A or S register.

NOTE

When using an immediate constant having a parcel value, and that is relocatable, the result of the relocation will be incorrect if the loader-determined actual address within the user's field length is greater than 1,048,575 because the resulting relocated value will have more than 22 significant bits.

The instruction type used for scalar memory transfers also requires a 22-bit *jkm* field for an address displacement. This instruction type uses the 4-bit *g* field for an operation code, the 3-bit *h* field to designate an address index register, and the 3-bit *i* field to designate a source or result register. (See subsection on special register values.)

Figure 7-4 shows the two general applications for the 2-parcel instruction format with combined *j*, *k*, and *m* fields.



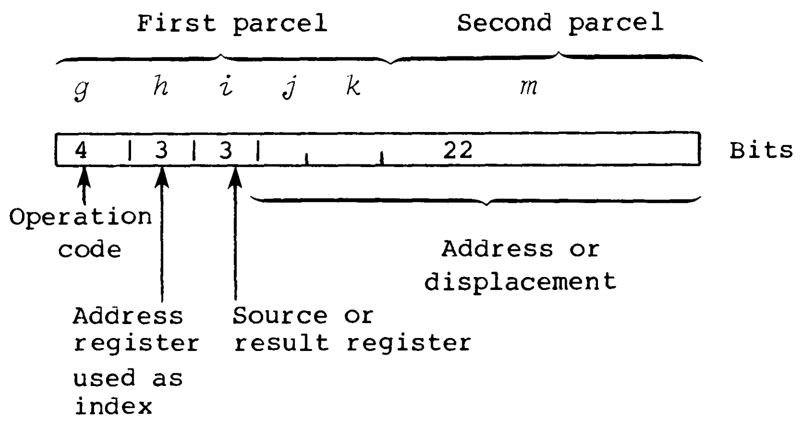


Figure 7-4. 2-parcel instruction format with combined *j*, *k*, and *m* fields

2-PARCEL INSTRUCTION FORMAT WITH COMBINED *i*, *j*, *k*, AND *m* FIELDS

The 2-parcel branch instruction type uses the combined *i*, *j*, *k*, and *m* fields to contain a 24-bit address that allows branching to an instruction parcel (see figure 7-5). A 7-bit operation code (*gh*) is followed by an *ijkm* field. The high-order bit of the *i* field is unused.

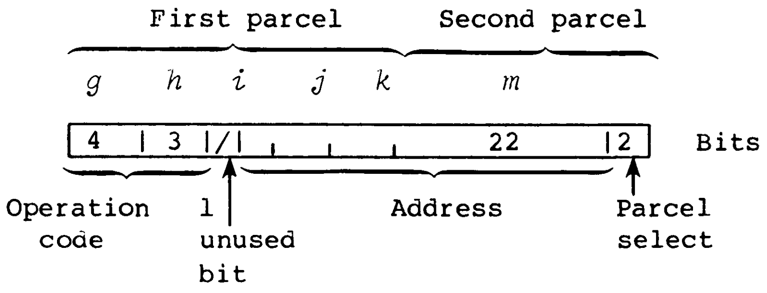


Figure 7-5. 2-parcel instruction format with combined *i*, *j*, *k*, and *m* fields

SPECIAL REGISTER VALUES

If the S0 and A0 registers are referenced in the *j* or *k* fields of an instruction, the contents of the respective register are not used; instead, a special operand is generated. The special value is available regardless of existing A0 or S0 reservations (and in this case are not checked). This use does not alter the actual value of the S0 or A0 register. If S0 or A0 is used in the *i* field as the operand, the actual value of the register is provided. The table below shows the special register values.

Field	Operand value
A*h*, *h*=0	0
A*i*, *i*=0	(A0)
A*j*, *j*=0	0
A*k*, *k*=0	1
S*i*, *i*=0	(S0)
S*j*, *j*=0	0
S*k*, *k*=0	2^{63}

INSTRUCTION ISSUE

Instructions are read from the instruction buffers one parcel at a time 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 the CIP register issues when conditions in the functional units and registers are such that functions required for execution can be performed without conflicting with a previously issued instruction. Instruction parcels issue out of the CIP register at a maximum rate of one per clock period. Once an instruction is delivered to the CIP register, that instruction 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 (10*h* and 12*h*).

Entry to the NIP register is blocked for the second parcel of a 2-parcel instruction, leaving NIP zero. Instead, the parcel is delivered to the Lower Instruction Parcel (LIP) register. The zeros in NIP (the pseudo second parcel) are transferred to CIP and issued as a do-nothing instruction.

When special register values (A0 or S0) are selected by an instruction for A*h*, A*j*, A*k*, S*j*, or S*k*, 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. Each instruction begins with boxed information consisting of the Cray Assembler Language (CAL) syntax format, a brief description of each instruction, and the octal code sequence defined by the *gh* fields. The appearance of an *m* in a format designates an instruction consisting of two parcels. An *x* in the format signifies the field containing the *x* is ignored during instruction execution on the CRAY-1 M Series of Computer Systems.

Following the boxed 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 delaying issue of an instruction until conditions are met.

Instruction issue time assumes that if an instruction issues at clock period *n* (CP *n*), the next instruction issues at CP *n* + issue time (preceding instruction issued) if its own issue conditions have been met.

The following special characters can appear in the operand field description of the symbolic machine instructions and are used by the assembler in determining the operation to be performed.

+	Arithmetic sum of adjoining registers
-	Arithmetic difference of adjoining registers
*	Arithmetic product of adjoining registers
/	Division or reciprocal
#	Use ones complement
>	Shift value or form mask from left to right
<	Shift value or form mask from right to left
&	Logical product of adjoining registers
!	Logical sum of adjoining registers
\	Logical difference of adjoining registers

In some instructions, register designators are prefixed by the following letters, which have special meaning to the assembler.

F	Floating-point operation
H	Half-precision operation
R	Rounded operation
I	Reciprocal iteration
P	Population count
Q	Population count parity
Z	Leading zero count

INSTRUCTION 000

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

Instruction 000 is treated as an error condition and an exchange sequence occurs. Content of the instruction buffers is voided by the exchange sequence. Instruction 000 halts execution of an incorrectly coded program branching into an unused area of memory (if memory was backgrounded with zeros) or into a data area (if the data is positive integers, right-justified ASCII, or floating-point zero). If monitor mode is not in effect, the Error Exit flag in the F register is set. All instructions issued before this instruction run to completion. When results of previously issued instructions arrive at the operating registers, an exchange occurs to the Exchange Package designated by contents of the XA register. The program address stored during the exchange sequence is the contents of the P register advanced by one count, that is, the address of the instruction following the error exit instruction.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process
EXECUTION TIME:	Instruction issue, 58 CPs for 16 banks or 66 CPs for 8 banks; this time includes an exchange sequence (40 CPs) and an instruction fetch operation (18 CPs for 16-bank phasing and 26 CPs for 8-bank phasing).
SPECIAL CASES:	Inhibit instruction issue Begin exchange sequence

† Special CAL syntax form

CAL Syntax	Description	Octal Code
CA,A*j* A*k*	Set the Current Address (CA) register for the channel indicated by (A*j*) to (A*k*) and activate the channel	0010*jk*
CL,A*j* A*k*	Set the Limit Address (CL) register for the channel indicated by (A*j*) to (A*k*)	0011*jk*
CI,A*j*	Clear the interrupt flag and error flag for the channel indicated by (A*j*)	0012*jx*
XA A*j*	Enter the XA register with (A*j*)	0013*jx*

Instructions 0010 through 0013 are privileged to monitor mode and provide operations useful to the operating system. Functions are selected through the *i* designator. 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 operation of the 6 Mbytes per second channels. Each channel has two registers directing 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 CA sets, activating the channel. As transfer continues, CA is incremented toward CL. When (CA) is equal to (CL), transfer is complete for words at initial (CA) through (CL)-1. When the *j* designator is 0 or when the content of A*j* is less than 2 or greater than 25, 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 (A*j*) to the XA register. When the *j* designator is 0, the XA register is cleared.

HOLD ISSUE CONDITIONS: Instructions 034 through 037 in process
Exchange in process
S*i*, A*j*, or A*k* reserved (except S0 and A0)

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 in effect.

For instructions 0010, 0011, and 0012:
If *j*=0, instruction is a no-op even in monitor mode.
If (A*j*) is less than 2 or (A*j*) 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 instruction 0013:
If *j*=0, XA register is cleared.

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

INSTRUCTION 0014

CAL Syntax	Description	Octal Code
RT S*j*	Enter the Real-time Clock register with (S*j*)	0014*j*0
PCI S*j*	Enter Interrupt Interval (II) register with (S*j*)	0014*j*4
CCI	Clear the programmable clock interrupt request	0014*x*5
ECI	Enable programmable clock interrupt request	0014*x*6
DCI	Disable programmable clock interrupt request	0014*x*7

Instruction 0014 performs specialized functions for managing real-time and programmable clocks and is privileged to monitor mode. The instruction is treated as a pass instruction if the monitor mode bit is not set.

When the *k* designator is 0, the instruction loads the contents of the S*j* register into the RTC register. When the *j* designator is 0, the RTC register is cleared.

When the *k* designator is 4, the instruction loads the low-order 32 bits from the S*j* register into both the II register and ICD counter. When the *j* designator is 0, the II register and the ICD counter are cleared.

When the *k* designator is 5, the 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, the instruction enables repeated programmable clock interrupt requests at a repetition rate determined by the value stored in the II register.

When the *k* designator is 7, the instruction disables repeated programmable clock interrupt requests until an instruction 0014*x*6 is executed to enable requests.

HOLD ISSUE CONDITIONS: Instructions 034 through 037 in process
Exchange in process
S*j*, A*j*, or A*k* 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 in effect.
	For instructions 0014*j*0 and 0014*j*4, if *j*=0, (S*j*)=0.
	Instructions 0015, 0016, 0017 are not implemented in the CRAY-1 M hardware but they act as no-op instructions. There is no CAL syntax for them.

INSTRUCTION 0020

CAL Syntax	Description	Octal Code
VL A*k*	Transmit (A*k*) to VL register	0020*xk*
VL 1[†]	Transmit 1 to VL register	0020*x*0

Instruction 0020 enters the VL register with a value determined by the contents of A*k*. The low-order 7 bits of (A*k*) 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 the VL register 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 the VL register is 0 or 64.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A*k* reserved (except A0)
EXECUTION TIME:	Instruction issue, 1 CP VL register ready, 1 CP
SPECIAL CASES:	Maximum vector length is 64. (A*k*)=1 if *k*=0. When (VL) modulo 64 is 0, then the number of operations performed is 64.

† Special CAL syntax form

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

Instruction 0021 sets the Floating-point Mode flag and instruction 0022 clears the Floating-point Mode flag in the M register.

When set, the Floating-point Mode flag enables interrupts on floating-point range errors as described in section 5.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A*k* reserved (except A0)
EXECUTION TIME:	Instruction issue, 1 CP
SPECIAL CASES:	Instructions 0023, 0024, 0025, 0026, and 0027 are not implemented but act as no-ops. There is no CAL syntax for them.

****

CAUTION

The operating system has status bits reflecting whether interrupts on floating-point range errors are enabled or disabled. Such software status bits need to be modified to agree with the Floating-point Mode flag.

****

INSTRUCTION 003

CAL Syntax	Description	Octal Code
VM S*j*	Transmit (S*j*) to VM register	003*xjx*
VM 0[†]	Clear VM register	003*x*0*x*

Instruction 003 enters the VM register with the contents of S*j*. The VM register is cleared if the *j* designator is 0. Instruction 003 is used in conjunction with the vector merge instructions (146 and 147) where an operation is performed depending on the contents of the VM register.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S*j* reserved (except S0) Instruction 003 in process, unit busy 3 CPs Instruction 14*x* in process, unit busy (VL)+4 CPs Instruction 175 in process, unit busy (VL)+4 CPs
EXECUTION TIME:	Instruction issue, 1 CP VM ready in 3 CPs except for use in instruction 073 VM ready in 6 CPs for instruction 073
SPECIAL CASE:	(S*j*)=0 if *j*=0.

† Special CAL syntax form

INSTRUCTION 004

CAL Syntax	Description	Octal Code
EX	Normal exit	004*xxx*
EX *exp*†	Normal exit, programmer encoded	004*ijk*

Instruction 004 causes an exchange sequence which voids the contents of the instruction buffers. If monitor mode is not in effect, the normal exit flag in the F register is set. All instructions issued before this instruction are run to completion; that is, when all results arrive at the operating registers because 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. Instruction 004 is used to issue a monitor request from a user program.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process
EXECUTION TIME:	Instruction issue, 58 CPs for 16 banks or 66 CPs for 8 banks; this time includes an exchange sequence (40 CPs) and an instruction fetch operation (18 CPs for 16 banks, 26 CPs for 8 banks).
SPECIAL CASES:	Inhibit instruction issue Begin exchange sequence

† Special CAL syntax form

INSTRUCTION 005

CAL Syntax	Description	Octal Code
J B*jk*	Branch to (B*jk*)	005*xjk*

Instruction 005 sets the P register to the 24-bit parcel address specified by the contents of B*jk* causing execution to continue at that address. Instruction 005 can be used to return from a subroutine.

HOLD ISSUE CONDITIONS: Instructions 034 through 037 in process
Exchange in process
Memory busy (hold memory can extend hold issue) if parcel 2 or branch destination is out of buffer or out of range

EXECUTION TIME: Instruction issue:

Instruction parcel and following parcel both in same buffer and branch address in a buffer; 7 CPs for 16 banks and 8 banks.

Instruction parcel and following parcel both in same buffer and branch address not in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Instruction parcel and following parcel in different buffers and branch address in a buffer; 7 CPs for 16 banks and 8 banks.

Instruction parcel and following parcel in different buffers and branch address not in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Parcel following instruction parcel not in a buffer and branch address in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Parcel following instruction parcel not in a buffer and branch address not in buffer; 33 CPs for 16 banks, 49 CPs for 8 banks.

SPECIAL CASES: This instruction executes as if it were a 2-parcel instruction. Even though the parcel following the first parcel of instruction 005 is not used, it can cause a delay of instruction 005 if it is out of buffer. See execution times.

INSTRUCTION 006

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

The 2-parcel instruction 006 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: Instructions 034 through 037 in process
Exchange in process
Memory busy (hold memory can extend hold issue) if parcel 2 or branch destination is out of buffer or out of range

EXECUTION TIME: Instruction issue:

Both parcels of instruction in the same buffer and branch address in a buffer; 5 CPs for 16 banks and 8 banks.

Both parcels of instruction in the same buffer and branch address not in a buffer; 18 CPs for 16 banks, 26 CPs for 8 banks.

Both parcels of instruction in different buffers and branch address in a buffer; 7 CPs for 16 banks and 8 banks.

Both parcels of instruction in different buffers and branch address not in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Second parcel of instruction not in a buffer and branch address in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Second parcel of instruction not in a buffer and branch address not in a buffer; 33 CPs for 16 banks, 49 CPs for 8 banks.

SPECIAL CASES: None

INSTRUCTION 007

CAL Syntax	Description	Octal Code
R *exp*	Return jump to *ijkm*; set B00 to (P)+2	007*ijkm*

The 2-parcel instruction 007 sets register B00 to the address of the parcel following the second parcel of the instruction. 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. This instruction provides a return linkage for subroutine calls. The subroutine is entered via a return jump. The subroutine can return to the caller at the instruction following the call by executing a branch to the contents of the B00 register, assuming that the called program saved and restored the contents of B00 or that B00 was not changed during execution of the called program.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process Memory busy (hold memory can extend hold issue) if parcel 2 or branch destination is out of buffer or out of range
EXECUTION TIME:	Instruction issue: Both parcels of instruction in the same buffer and branch address in a buffer; 5 CPs for 16 banks and 8 banks. Both parcels of instruction in the same buffer and branch address not in a buffer; 18 CPs for 16 banks, 26 CPs for 8 banks. Both parcels of instruction in different buffers and branch address in a buffer; 7 CPs for 16 banks and 8 banks. Both parcels of instruction in different buffers and branch address not in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks. Second parcel of instruction not in a buffer and branch address in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks. Second parcel of instruction not in a buffer and branch address not in a buffer; 33 CPs for 16 banks, 49 CPs for 8 banks.
SPECIAL CASES:	None

CAL Syntax		Description	Octal Code
JAZ	*exp*	Branch to *ijkm* if (A0)=0	010*ijkm*
JAN	*exp*	Branch to *ijkm* if (A0)≠0	011*ijkm*
JAP	*exp*	Branch to *ijkm* if (A0) positive, includes (A0)=0	012*ijkm*
JAM	*exp*	Branch to *ijkm* if (A0) negative	013*ijkm*

The 2-parcel instructions 010 through 013 test the contents of A0 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:
Instructions 034 through 037 in process
Exchange in process
A0 busy in previous 2 CPs
Memory busy (hold memory can extend hold issue) if parcel 2 or branch destination is out of range

EXECUTION TIME:
Instruction issue for branch taken:

Both parcels of instruction in the same buffer, branch taken, and branch address in a buffer; 5 CPs for 16 banks and 8 banks.

Both parcels of instruction in the same buffer, branch taken, and branch address not in a buffer; 18 CPs for 16 banks, 26 CPs for 8 banks.

Both parcels of instruction in different buffers, branch taken, and branch address in a buffer; 7 CPs for 16 and 8 banks.

EXECUTION TIME: (continued)

Both parcels of instruction in different buffers, branch taken, and branch address not in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Second parcel of instruction not in a buffer, branch taken, and branch address in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Second parcel of instruction not in a buffer, branch taken, and branch address not in buffer; 33 CPs for 16 banks, 49 CPs for 8 banks.

Instruction issue for branch not taken:

Both parcels of instruction in the same buffer, branch not taken, and next instruction in same instruction buffer; 2 CPs for 16 banks and 8 banks.

Both parcels of instruction in the same buffer and branch not taken with next instruction in different instruction buffer; 4 CPs for 16 banks and 8 banks.

Both parcels of instruction in the same buffer and branch not taken with next instruction in memory; 18 CPs for 16 banks, 26 CPs for 8 banks.

Both parcels of instruction in different buffers and branch not taken; 4 CPs for 16 banks and 8 banks.

Second parcel of instruction not in a buffer and branch not taken; 17 CPs for 16 banks, 25 CPs for 8 banks.

SPECIAL CASE: (A0)=0 is considered a positive condition.

CAL Syntax		Description	Octal Code
JSZ	*exp*	Branch to *ijkm* if (S0)=0	014*ijkm*
JSN	*exp*	Branch to *ijkm* if (S0)≠0	015*ijkm*
JSP	*exp*	Branch to *ijkm* if (S0) positive, includes (S0)=0	016*ijkm*
JSM	*exp*	Branch to *ijkm* if (S0) negative	017*ijkm*

The 2-parcel instructions 014 through 017 test the contents of S0 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:	Instructions 034 through 037 in process Exchange in process S0 busy in previous 2 CPs Memory busy (hold memory can extend hold issue) if parcel 2 or branch destination is out of buffer or out of range
EXECUTION TIME:	Instruction issue for branch taken: Both parcels of instruction in the same buffer, branch taken, and branch address in a buffer; 5 CPs for 16 banks and 8 banks. Both parcels of instruction in the same buffer, branch taken, and branch address not in a buffer; 18 CPs for 16 banks, 26 CPs for 8 banks. Both parcels of instruction in different buffers, branch taken, and branch address in a buffer; 7 CPs for 16 banks and 8 banks.

EXECUTION TIME: (continued)

Both parcels of instruction in different buffers, branch taken, and branch address not in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Second parcel of instruction not in a buffer, branch taken, and branch address in a buffer; 20 CPs for 16 banks, 28 CPs for 8 banks.

Second parcel of instruction not in a buffer, branch taken, and branch address not in a buffer; 33 CPs for 16 banks, 49 CPs for 8 banks.

Instruction issue for branch not taken:

Both parcels of instruction in the same buffer, branch not taken, and next instruction in same instruction buffer; 2 CPs for 16 banks and 8 banks.

Both parcels of instruction in the same buffer and branch not taken with next instruction in different instruction buffer; 4 CPs for 16 banks and 8 banks.

Both parcels of instruction in the same buffer and branch not taken with next instruction in memory; 18 CPs for 16 banks, 26 CPs for 8 banks.

Both parcels of instruction in different buffers and branch not taken; 4 CPs for 16 banks and 8 banks.

Second parcel of instruction not in a buffer and branch not taken; 17 CPs for 16 banks, 25 CPs for 8 banks.

SPECIAL CASE: (S0)=0 is considered a positive condition.

CAL Syntax	Description	Octal Code
A*i* *exp*	Transmit *jkm* to A*i*	020*ijkm*
A*i* *exp*	Transmit ones complement of *jkm* to A*i*	021*ijkm*

The 2-parcel instruction 020 enters a 24-bit value into A*i* composed of the 22-bit *jkm* field and 2 high-order bits of 0.

The 2-parcel instruction 021 enters 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 into A*i*. The complement is formed by changing all 1 bits to 0 and all 0 bits to 1. Thus, for instruction 021, the high-order 2 bits of A*i* are set to 1. The instruction provides a means of entering a negative value into A*i*. However, if the instruction is used to enter a negative number, the positive number used in the *jkm* field must be one smaller than the absolute value of the expected final negative number.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict A*i* 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, 17 CPs A*i* ready, 1 CP
SPECIAL CASES:	None

INSTRUCTION 022

CAL Syntax	Description	Octal Code
A*i* *exp*	Transmit *jk* to A*i*	022*ijk*

Instruction 022 enters the 6-bit quantity from the *jk* field into the low-order 6 bits of A*i*. The high-order 18 bits of A*i* are zeroed. No sign extension occurs.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict A*i* reserved
EXECUTION TIME:	Instruction issue, 1 CP A*i* ready, 1 CP
SPECIAL CASES:	None

INSTRUCTION 023

CAL Syntax	Description	Octal Code
A*i* S*j*	Transmit (S*j*) to A*i*	023*ijx*

Instruction 023 enters the low-order 24 bits of (S*j*) into A*i*. The high-order bits of (S*j*) are ignored.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict A*i* reserved S*j* reserved (except S0)
EXECUTION TIME:	Instruction issue, 1 CP A*i* ready, 1 CP
SPECIAL CASE:	(S*j*)=0 if *j*=0.

CAL Syntax	Description	Octal Code
A*i* B*jk*	Transmit (B*jk*) to A*i*	024*ijk*
B*jk* A*i*	Transmit (A*i*) to B*jk*	025*ijk*

Instruction 024 enters the contents of B*jk* into A*i*.

Instruction 025 enters the contents of A*i* into B*jk*.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict (instruction 024 only) A*i* reserved
EXECUTION TIME:	For instruction 024, A*i* ready, 1 CP Instruction issue for instruction 024 or 025, 1 CP
SPECIAL CASES:	None

INSTRUCTION 026

CAL Syntax	Description	Octal Code
A*i* PS*j*	Population count of (S*j*) to A*i*	026*ij*0
A*i* QS*j*	Population count parity of (S*j*) to A*i*	026*ij*1

Instruction 026 is executed in the Population/Leading Zero functional unit.

Instruction 026*ij*0 counts the number of bits set to 1 in (S*j*) and enters the result into the low-order 7 bits of A*i*. The high-order 17 bits of A*i* are zeroed.

Instruction 026*ij*1 counts the number of bits set to 1 in (S*j*). Then, the low-order bit, showing the odd/even state of the result is transferred to the low-order bit position of the A*i* register. The high-order 23 bits are cleared. The actual population count is not transferred.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict A*i* reserved S*j* reserved (except S0)
EXECUTION TIME:	Instruction issue, 1 CP A*i* ready, 4 CPs
SPECIAL CASE:	(A*i*)=0 if *j*=0.

CAL Syntax	Description	Octal Code
A*i* ZS*j*	Leading zero count of (S*j*) to A*i*	027*ijx*

Instruction 027 is executed in the Population/Leading Zero functional unit.

Instruction 027 counts the number of leading zeros in S*j* and enters the result into the low-order 7 bits of A*i*. The high-order 17 bits of A*i* are zeroed.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict A*i* reserved S*j* reserved (except S0)
EXECUTION TIME:	Instruction issue, 1 CP A*i* ready, 3 CPs
SPECIAL CASE:	(A*i*)=64 if *j*=0.

INSTRUCTIONS 030 - 031

CAL Syntax		Description	Octal Code
A*i*	A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*	030*ijk*
A*i*	A*k*†	Transmit (A*k*) to A*i*	030*i*0*k*
A*i*	A*j*+1†	Integer sum of (A*j*) and 1 to A*i*	030*ij*0
A*i*	A*j*-A*k*	Integer difference (A*j*) less (A*k*) to A*i*	031*ijk*
A*i*	-1†	Transmit -1 to A*i*	031*i*00
A*i*	-A*k*†	Transmit the negative of (A*k*) to A*i*	031*i*0*k*
A*i*	A*j*-1†	Integer difference (A*j*) less 1 to A*i*	031*ij*0

Instructions 030 and 031 are executed in the Address Add functional unit.

Instruction 030 forms the integer sum of (A*j*) and (A*k*) and enters the result into A*i*. No overflow is detected.

Instruction 031 forms the integer difference of (A*j*) and (A*k*) and enters the result into A*i*. No overflow is detected.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict A*i* reserved A*j* or A*k* reserved (except A0)
EXECUTION TIME:	Instruction issue, 1 CP A*i* ready, 2 CPs
SPECIAL CASES:	For instruction 030: (A*i*)=(A*k*) if *j*=0 and *k*≠0. (A*i*)=1 if *j*=0 and *k*=0. (A*i*)=(A*j*)+1 if *j*≠0 and *k*=0. For instruction 031: (A*i*)= -(A*k*) if *j*=0 and *k*≠0. (A*i*)= -1 if *j*=0 and *k*=0. (A*i*)=(A*j*)-1 if *j*≠0 and *k*=0.

† Special CAL syntax form

INSTRUCTION 032

CAL Syntax	Description	Octal Code
A*i* A*j**A*k*	Integer product of (A*j*) and (A*k*) to A*i*	032*ijk*

Instruction 032 is executed in the Address Multiply functional unit.

Instruction 032 forms the integer product of (A*j*) and (A*k*) and enters the low-order 24 bits of the result into A*i*. No overflow is detected.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict A*i* reserved A*j* or A*k* reserved (except A0)
EXECUTION TIME:	Instruction issue, 1 CP A*i* ready, 6 CPs
SPECIAL CASES:	(A*i*)=0 if *j*=0. (A*k*)=1 if *k*=0. Thus (A*i*)=(A*j*) if *j*≠0 and *k*=0.

INSTRUCTION 033

CAL Syntax		Description	Octal Code
A*i*	CI	Channel number of highest priority interrupt request to A*i*	033*i*0*x*
A*i*	CA,A*j*	Current address of channel (A*j*) to A*i*	033*ij*0
A*i*	CE,A*j*	Error flag of channel (A*j*) to A*i*	033*ij*1

Instruction 033 enters channel status information into A*i*. The *j* and *k* designators and contents of A*j* define the desired information.

The channel number of the highest priority interrupt request is entered into A*i* when the *j* designator is 0. The contents of A*j* specify a channel number when the *j* designator is nonzero. The value of the Current Address (CA) register for the channel is entered into A*i* when the *k* designator is 0. The error flag for the channel is entered into the low-order bit of A*i* when the *k* designator is 1. High-order bits of A*i* are cleared. The error flag is cleared only in monitor mode using instruction 0012.

Instruction 033 does not interfere with channel operation.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A register access conflict A*i* reserved A*j* reserved (except A0)
EXECUTION TIME:	Instruction issue, 1 CP A*i* ready, 4 CPs
SPECIAL CASES:	(A*i*)=highest priority channel causing interrupt if (A*j*)=0. (A*i*)=current address of channel (A*j*) if (A*j*)≠0 and *k*=0. (A*i*)=I/O error flag of channel (A*j*) if (A*j*)≠0 and *k*=1. (A*i*)=0 if (A*j*)=1. 2 CPs must elapse after an instruction 0012*jx* issues before issuing an instruction 033*i*0*x*.

CAL Syntax	Description	Octal Code
B*jk*,A*i* ,A0	Block transfer (A*i*) words from memory starting at address (A0) to B registers starting at register *jk*	034*ijk*
B*jk*,A*i* 0,A0†	Block transfer (A*i*) words from memory starting at address (A0) to B registers starting at register *jk*	034*ijk*
,A0 B*jk*,A*i*	Block transfer (A*i*) words from B registers starting at register *jk* to memory starting at address (A0)	035*ijk*
0,A0 B*jk*,A*i*†	Block transfer (A*i*) words from B registers starting at register *jk* to memory starting at address (A0)	035*ijk*
T*jk*,A*i* ,A0	Block transfer (A*i*) words from memory starting at address (A0) to T registers starting at register *jk*	036*ijk*
T*jk*,A*i* 0,A0†	Block transfer (A*i*) words from memory starting at address (A0) to T registers starting at register *jk*	036*ijk*
,A0 T*jk*,A*i*	Block transfer (A*i*) words from T registers starting at register *jk* to memory starting at address (A0)	037*ijk*
0,A0 T*jk*,A*i*†	Block transfer (A*i*) words from T registers starting at register *jk* to memory starting at address (A0)	037*ijk*

Instructions 034 through 037 perform block transfers between memory and B or T registers.

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

† Special CAL syntax form

The first register involved in the transfer is specified by *jk*. Successive transfers involve successive B or T registers until B77 or T77 is reached. Since processing of the registers is circular, B00 is processed after B77 and T00 is processed after T77 if the count in (A*i*) is not exhausted.

The first memory location referenced by the transfer instruction is specified by (A0). The A0 register contents are not altered by execution of the instruction. Once the instruction issues, A0 can be changed immediately without affecting 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, high-order 40 bits are zeroed. When transferring from memory to B registers, only low-order 24 bits are transmitted; high-order 40 bits are ignored.

HOLD ISSUE CONDITIONS:	A0 through A7 reserved (instructions 034 and 036) A0 A*i*, or S0 through S7 reserved (instructions 035 and 037) Block sequence flag set (instructions 034 through 037, 176, and 177) Instructions 034 through 037 in process Exchange in process Scalar reference in CP 2 Rank B data valid Fetch request in previous CP I/O memory request
EXECUTION TIME:	For instructions 034 and 036: Instruction issue, 16 CPs+(A*i*) CPS if (A*i*)≠0; 5 CPs if (A*i*)=0. For instructions 035 and 037: Instruction issue, 10 CPs+(A*i*) CPS if (A*i*)≠0; 7 CPs if(A*i*)=0.
SPECIAL CASES:	Block all issues when in process. Block all I/O references. (A*i*)=0 causes a transfer of no data. (A*i*) in the range greater than 100_8 and less than 200_8 causes a wrap-around condition. If (A*i*) 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.

CAL Syntax	Description	Octal Code
S*i* *exp*	Transmit *jkm* to S*i*	040*ijkm*
S*i* *exp*	Transmit complement of *jkm* to S*i*	041*ijkm*

The 2-parcel instructions 040 and 041 enter immediate values into an S register.

Instruction 040 enters a 64-bit value composed of the 22-bit *jkm* field and 42 high-order bits of 0 into S*i*.

Instruction 041 enters 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 into S*i*. The complement is formed by changing all 1 bits to 0 and all 0 bits to 1. Thus, for instruction 041, the high-order 42 bits of S*i* are set to 1's. The instruction provides for entering a negative value into S*i*. Since the register value is the ones complement of *jkm*, to get the twos complement *jkm* should be 0 to get -1, 1 to get -2, 3 to get -4, etc.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S register access conflict S*i* 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, 17 CPs S*i* ready, 1 CP
SPECIAL CASES:	None

CAL Syntax	Description	Octal Code
S*i* <*exp*	Form *exp* bits of ones mask in S*i* from right; *jk* field gets 64-*exp*.	042*ijk*
S*i* #>*exp*†	Form *exp* bits of zeros mask in S*i* from left; *jk* field gets *exp*.	042*ijk*
S*i* 1†	Enter 1 into S*i*	042*i*77
S*i* -1†	Enter -1 into S*i*	042*i*00
S*i* >*exp*	Form *exp* bits of ones mask in S*i* from left; *jk* field gets *exp*.	043*ijk*
S*i* #<*exp*†	Form *exp* bits of zeros mask in S*i* from right; *jk* field gets 64-*exp*.	043*ijk*
S*i* 0†	Clear S*i*	043*i*00

Instructions 042 and 043 are executed in the Scalar Logical functional unit.

Instruction 042 generates a mask of 64-*jk* ones from right to left in S*i*. For example, if *jk*=0, S*i* contains all 1 bits (integer value= -1) and if *jk*=77_8, S*i* contains zeros in all but the low-order bit (integer value=1).

Instruction 043 generates a mask of *jk* ones from left to right in S*i*. For example, if *jk*=0, S*i* contains all 0 bits (integer value=0) and if *jk*=77_8, S*i* contains ones in all but the low-order bit (integer value= -2).

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S register access conflict S*i* reserved
EXECUTION TIME:	Instruction issue, 1 CP S*i* ready, 1 CP
SPECIAL CASES:	None

† Special CAL syntax form

CAL Syntax	Description	Octal Code
S*i* S*j*&S*k*	Logical product of (S*j*) and (S*k*) to S*i*	044*ijk*
S*i* S*j*&SB†	Sign bit of (S*j*) to S*i*	044*ij*0
S*i* SB&S*j*†	Sign bit of (S*j*) to S*i* (*j*≠0)	044*ij*0
S*i* #S*k*&S*j*	Logical product of (S*j*) and complement of (S*k*) to S*i*	045*ijk*
S*i* #SB&S*j*†	(S*j*) with sign bit cleared to S*i*	045*ij*0
S*i* S*j*\S*k*	Logical difference of (S*j*) and (S*k*) to S*i*	046*ijk*
S*i* S*j*\SB†	Toggle sign bit of (S*j*), then enter into S*i*	046*ij*0
S*i* SB\S*j*†	Toggle sign bit of (S*j*); then enter into S*i* (*j*≠0)	046*ij*0
S*i* #S*j*\S*k*	Logical equivalence of (S*k*) and (S*j*) to S*i*	047*ijk*
S*i* #S*k*†	Transmit ones complement of (S*k*) to S*i*	047*i*0*k*
S*i* #S*j*\SB†	Logical equivalence of (S*j*) and sign bit to S*i*	047*ij*0
S*i* #SB\S*j*†	Logical equivalence of (S*j*) and sign bit to S*i* (*j*≠0)	047*ij*0
S*i* #SB†	Enter ones complement of sign bit into S*i*	047*i*00
S*i* S*j*!S*i*&S*k*	Scalar merge	050*ijk*
S*i* S*j*!S*i*&SB†	Scalar merge of (S*i*) and sign bit of (S*j*) to S*i*	050*ij*0
S*i* S*j*!S*k*	Logical sum of (S*j*) and (S*k*) to S*i*	051*ijk*
S*i* S*k*†	Transmit (S*k*) to S*i*	051*i*0*k*
S*i* S*j*!SB†	Logical sum of (S*j*) and sign bit to S*i*	051*ij*0
S*i* SB!S*j*†	Logical sum of (S*j*) and sign bit to S*i* (*j*≠0)	051*ij*0
S*i* SB†	Enter sign bit into S*i*	051*i*00

† Special CAL syntax

Instructions 044 through 051 are executed in the Scalar Logical functional unit.

Instruction 044 forms the logical product (AND) of (S*j*) and (S*k*) and enters the result into S*i*. Bits of S*i* are set to 1 when corresponding bits of (S*j*) and (S*k*) are 1 as in the following example:

$$\begin{array}{ll} (Sj) & = 1\ 1\ 0\ 0 \\ (Sk) & = \underline{1\ 0\ 1\ 0} \\ (Si) & = 1\ 0\ 0\ 0 \end{array}$$

(S*j*) is transmitted to S*i* if the *j* and *k* designators have the same nonzero value. S*i* is cleared if the *j* designator is 0. The sign bit of (S*j*) is transmitted to S*i* if the *j* designator is nonzero and the *k* designator is 0.

Instruction 045 forms the logical product (AND) of (S*j*) and the complement of (S*k*) and enters the result into S*i*. Bits of S*i* are set to 1 when corresponding bits of (S*j*) and the complement of (S*k*) are 1 as in the following example where (S*k*') = complement of (S*k*):

if (S*k*) = 1 0 1 0

$$\begin{array}{ll} (Sj) & = 1\ 1\ 0\ 0 \\ (Sk') & = \underline{0\ 1\ 0\ 1} \\ (Si) & = 0\ 1\ 0\ 0 \end{array}$$

S*i* is cleared if the *j* and *k* designators have the same value or if the *j* designator is 0. (S*j*) with the sign bit cleared is transmitted to S*i* if the *j* designator is nonzero and the *k* designator is 0.

Instruction 046 forms the logical difference (exclusive OR) of (S*j*) and (S*k*) and enters the result into S*i*. Bits of S*i* are set to 1 when corresponding bits of (S*j*) and (S*k*) are different as in the following example:

$$\begin{array}{ll} (Sj) & = 1\ 1\ 0\ 0 \\ (Sk) & = \underline{1\ 0\ 1\ 0} \\ (Si) & = 0\ 1\ 1\ 0 \end{array}$$

S*i* is cleared if the *j* and *k* designators have the same nonzero value. (S*k*) is transmitted to S*i* if the *j* designator is 0 and the *k* designator is nonzero. The sign bit of (S*j*) is complemented and the result is transmitted to S*i* if the *j* designator is nonzero and the *k* designator is 0.

Instruction 047 forms the logical equivalence of (S*j*) and (S*k*) and enters the result into S*i*. Bits of S*i* are set to 1 when corresponding bits of (S*j*) and (S*k*) are the same as in the following example:

$$\begin{array}{lc} (Sj) = & 1\ 1\ 0\ 0 \\ (Sk) = & 1\ 0\ 1\ 0 \\ \hline (Si) = & 1\ 0\ 0\ 1 \end{array}$$

S*i* is set to all ones if the *j* and *k* designators have the same nonzero value. The complement of (S*k*) is transmitted to S*i* if the *j* designator is 0 and the *k* designator is nonzero. All bits except the sign bit of (S*j*) are complemented and the result is transmitted to S*i* if the *j* designator is nonzero and the *k* designator is 0. The result is the complement produced by instruction 046.

Instruction 050 merges the contents of (S*j*) with (S*i*) depending on the ones mask in S*k*. The result is defined by the following Boolean equation where S*k*' is the complement of S*k* as illustrated:

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

if (S*k*) = 1 1 1 1 0 0 0 0

$$\begin{array}{lc} (Sk') = & 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \\ (Si) = & 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0 \\ (Sj) = & 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\ \hline (Si) = & 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0 \end{array}$$

Instruction 050 is intended for merging portions of 64-bit words into a composite word. Bits of S*i* are cleared when the corresponding bits of S*k* are 1 if the *j* designator is 0 and the *k* designator is nonzero. The sign bit of (S*j*) replaces the sign bit of S*i* if the *j* designator is nonzero and the *k* designator is 0. The sign bit of S*i* is cleared if the *j* and *k* designators are both 0.

Instruction 051 forms the logical sum (inclusive OR) of (S*j*) and (S*k*) and enters the result into S*i*. Bits of S*i* are set when 1 of the corresponding bits of (S*j*) and (S*k*) is set as in the following example:

$$\begin{array}{lc} (Sj) = & 1\ 1\ 0\ 0 \\ (Sk) = & 1\ 0\ 1\ 0 \\ \hline (Si) = & 1\ 1\ 1\ 0 \end{array}$$

(S*j*) is transmitted to S*i* if the *j* and *k* designators have the same nonzero value. (S*k*) is transmitted to S*i* if the *j* designator is 0 and the *k* designator is nonzero. (S*j*) with the sign bit set to 1 is transmitted to S*i* if the *j* designator is nonzero and the *k* designator is 0. A ones mask consisting of only the sign bit is entered into S*i* if the *j* and *k* designators are both 0.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S register access conflict S*i* reserved S*j* or S*k* reserved (except S0)
EXECUTION TIME:	Instruction issue, 1 CP S*i* ready, 1 CP
SPECIAL CASES:	(S*j*)=0 if *j*=0. (S*k*)=2^{63} if *k*=0.

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

Instructions 052 through 055 are executed in the Scalar Shift functional unit. They shift values in an S register by an amount specified by *jk*. All shifts are end off with zero fill.

Instruction 052 shifts (S*i*) left *jk* places and enters the result into S0. Shift range is 0 through 63 left.

Instruction 053 shifts (S*i*) right by 64-*jk* places and enters the result into S0. Shift range is 1 through 64 right.

Instruction 054 shifts (S*i*) left *jk* places and enters the result into S*i*. Shift range is 0 through 63 left.

Instruction 055 shifts (S*i*) right by 64-*jk* places and enters the result into S*i*. Shift range is 1 through 64 right.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S register access conflict S*i* reserved S0 reserved (instructions 052 and 053 only)
EXECUTION TIME:	Instruction issue, 1 CP For instructions 052 and 053, S0 ready, 2 CPs For instructions 054 and 055, S*i* ready, 2 CPs
SPECIAL CASES:	None

CAL Syntax	Description	Octal Code
S*i* S*i*,S*j*<A*k*	Shift (S*i*) and (S*j*) left by (A*k*) places to S*i*	056*ijk*
S*i* S*i*,S*j*<1†	Shift (S*i*) and (S*j*) left one place to S*i*	056*ij*0
S*i* S*i*<A*k*†	Shift (S*i*) left (A*k*) places to S*i*	056*i*0*k*
S*i* S*j*,S*i*>A*k*	Shift (S*j*) and (S*i*) right by (A*k*) places to S*i*	057*ijk*
S*i* S*j*,S*i*>1†	Shift (S*j*) and (S*i*) right one place to S*i*	057*ij*0
S*i* S*i*>A*k*†	Shift (S*i*) right (A*k*) places to S*i*	057*i*0*k*

Instructions 056 and 057 are executed in the Scalar Shift functional unit. They shift 128-bit values formed by logically joining two S registers. Shift counts are obtained from register A*k*. All shift counts, (A*k*), are considered positive. All 24 bits of (A*k*) are used for the shift count. A shift of one place occurs if the *k* designator is 0. If *j*=0, the shifts function as if the shifted value was 64 bits rather than 128 bits since the S*j* value used 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 instructions 056 and 057, (S*j*) is unchanged, provided *i*≠*j*. For shifts greater than 64, the shift is end off with zero fill. If *i*=*j* and the shift is greater than 64, the shift is the same as if the respective instruction 054 or 055 was used with a shift count 64 less.

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

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

† Special CAL syntax form

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S register access conflict S*i* reserved S*j* or A*k* reserved (except S0 and A0)
EXECUTION TIME:	Instruction issue, 1 CP S*i* ready, 3 CPs
SPECIAL CASES:	(S*j*)=0 if *j*=0. (A*k*)=1 if *k*=0. Circular shift if $i=j\neq 0$ and (A*k*) is less than 64.

CAL Syntax		Description	Octal Code
S*i*	S*j*+S*k*	Integer sum of (S*j*) and (S*k*) to S*i*	060*ijk*
S*i*	S*j*-S*k*	Integer difference of (S*j*) and (S*k*) to S*i*	061*ijk*
S*i*	-S*k*†	Transmit negative of (S*k*) to S*i*	061*i*0*k*

Instructions 060 and 061 are executed in the Scalar Add functional unit.

Instruction 060 forms the integer sums of (S*j*) and (S*k*) and enters the result into S*i*. No overflow is detected.

Instruction 061 forms the integer difference of (S*j*) and (S*k*) and enters the result into S*i*. No overflow is detected.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S register access conflict S*i* reserved S*j* or S*k* reserved (except S0)
EXECUTION TIME:	S*i* ready, 3 CPs Instruction issue, 1 CP
SPECIAL CASES:	(S*i*)=2^{63} if *j*=0 and *k*=0. For instruction 060: (S*i*)=(S*k*) if *j*=0 and *k*≠0. (S*i*)=(S*j*) with 2^{63} complemented if *j*≠0 and *k*=0. For instruction 061: (S*i*)= -(S*k*) if *j*=0 and *k*≠0. (S*i*)=(S*j*) with 2^{63} complemented if *j*≠0 and *k*=0.

† Special CAL syntax form

CAL Syntax		Description	Octal Code
S*i*	S*j*+FS*k*	Floating sum of (S*j*) and (S*k*) to S*i*	062*ijk*
S*i*	+FS*k*†	Normalize (S*k*) to S*i*	062*i*0*k*
S*i*	S*j*-FS*k*	Floating difference of (S*j*) and (S*k*) to S*i*	063*ijk*
S*i*	-FS*k*†	Transmit normalized negative of (S*k*) to S*i*	063*i*0*k*

Instructions 062 and 063 are performed in the Floating-point Add functional unit. Operands are assumed to be in floating-point format. The result is normalized even if the operands are not normalized.

Instruction 062 forms the sum of the floating-point quantities in S*j* and S*k* and enters the normalized result into S*i*.

Instruction 063 forms the difference of the floating-point quantities in S*j* and S*k* and enters the normalized result into S*i*.

Overflow conditions are described in section 5. For floating-point operands with the sign set (bit=1), zero exponent and zero coefficient are treated as 0 (that is, all 64 bits=0).††

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S*i* register access conflict S*i* reserved S*j* or S*k* reserved (except S0) Instructions 170 through 173 in process, unit busy (VL)+4 CPs
EXECUTION TIME:	Instruction issue, 1 CP S*i* ready, 6 CPs

† Special CAL syntax form

†† Considered -0. No floating-point unit generates a -0 except the Floating-point Multiply functional unit if one of the operands was a -0. Normally, -0 occurs in logical manipulations when a sign is attached to a number; that number can be 0.

SPECIAL CASES:

For instruction 062:
(S*i*)=(S*k*) normalized if (S*k*) exponent is valid, *j*=0, and *k*≠0.
(S*i*)=(S*j*) normalized if (S*j*) exponent is valid, *j*≠0, and *k*=0.

For instruction 063:
(S*i*)= -(S*k*) normalized if (S*k*) exponent is valid, *j*=0, and *k*≠0. Sign of (S*i*) is opposite that of (S*k*) if (S*k*)≠0.
(S*i*)=(S*j*) normalized if (S*j*) exponent is valid, *j*≠0, and *k*=0.

CAL Syntax		Description	Octal Code
S*i*	S*j**FS*k*	Floating-point product of (S*j*) and (S*k*) to S*i*	064*ijk*
S*i*	S*j**HS*k*	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*	065*ijk*
S*i*	S*j**RS*k*	Rounded floating-point product of (S*j*) and (S*k*) to S*i*	066*ijk*
S*i*	S*j**IS*k*	Reciprocal iteration; 2-(S*j*)*(S*k*) to S*i*	067*ijk*

Instructions 064 through 067 are executed in the Floating-point Multiply functional unit. Operands are assumed to be in floating-point format. The result is not guaranteed to be normalized if the operands are not normalized.

Instruction 064 forms the product of the floating-point quantities in S*j* and S*k* and enters the result into S*i*.

Instruction 065 forms the half-precision rounded product of the floating-point quantities in S*j* and S*k* and enters the result into S*i*. The low-order 19 bits of the result are cleared.

Instruction 066 forms the rounded product of the floating-point quantities in S*j* and S*k* and enters the result into S*i*.

Instruction 067 forms two minus the product of the floating-point quantities in S*j* and S*k* and enters the result into S*i*. This instruction is used in the divide sequence as described in section 5 under Floating-point Arithmetic.

In the evaluation C=2-B*A, B must be a reciprocal of A of less than 47 significant bits and not the exact reciprocal; otherwise, C will be in error. The reciprocal produced by the reciprocal approximation instruction meets this criterion.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process
	Exchange in process
	S register access conflict
	S*i* reserved
	S*j* or S*k* reserved (except S0)
	Instructions 160 through 167 in process, unit busy (VL)+4 CPs

EXECUTION TIME:	Instruction issue, 1 CP S*i* ready, 7 CPs
SPECIAL CASES:	(S*j*)=0 if *j*=0.

(S*k*)=2^{63} if *k*=0.

If both exponent fields are 0, an integer multiply is performed. Correct integer multiply results are produced if the following conditions are met:

- Both operand sign bits are 0.
- The sum of the 0 bits to the right of the least significant 1 bit in the two operands is greater than or equal to 48.

The integer result obtained is the high-order 48 bits of the 96-bit product of the two operands.

INSTRUCTION 070

CAL Syntax	Description	Octal Code
S*i* /HS*j*	Floating-point reciprocal approximation of (S*j*) to S*i*	070*ijx*

Instruction 070 is executed in the Reciprocal Approximation functional unit.

Instruction 070 forms an approximation to the reciprocal of the normalized floating-point quantity in S*j* and enters the result into S*i*. This instruction occurs in the divide sequence to compute the quotient of two floating-point quantities as described in section 5 under Floating-point Arithmetic.

The reciprocal approximation instruction produces a result of 30 significant bits. The low-order 18 bits are zeros. The number of significant bits can be extended to 48 using the reciprocal iteration instruction and a multiply.

Instruction 070 can delay a scalar memory reference instruction for 1 CP with the hold memory function.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S*i* reserved S*j* reserved (except S0) Instruction 174 in process, unit busy (VL)+4 CPs
EXECUTION TIME:	S*i* ready, 14 CPs Instruction issue, 1 CP
SPECIAL CASES:	(S*i*) is meaningless if (S*j*) is not normalized. The unit assumes that bit 2^{47} of (S*j*)=1; no test is made of this bit. (S*j*)=0 produces a range error; the result is meaningless. (S*j*)=0 if *j*=0.

INSTRUCTION 071

CAL Syntax	Description	Octal Code
S*i* A*k*	Transmit (A*k*) to S*i* with no sign extension	071*i*0*k*
S*i* +A*k*	Transmit (A*k*) to S*i* with sign extension	071*i*1*k*
S*i* +FA*k*	Transmit (A*k*) to S*i* as unnormalized floating-point number	071*i*2*k*
S*i* 0.6	Transmit constant 0.75 x 2^{48} to S*i*	071*i*3*x*
S*i* 0.4	Transmit constant 0.5 to S*i*	071*i*4*x*
S*i* 1.	Transmit constant 1.0 to S*i*	071*i*5*x*
S*i* 2.	Transmit constant 2.0 to S*i*	071*i*6*x*
S*i* 4.	Transmit constant 4.0 to S*i*	071*i*7*x*

Instruction 071 performs 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 A*k* is transmitted to S*i*. The value is treated as an unsigned integer. The high-order bits of S*i* are zeros.

When the *j* designator is 1, the 24-bit value in A*k* is transmitted to S*i*. The value is treated as a signed integer. The sign bit of A*k* is extended through the high-order bit of S*i*.

When the *j* designator is 2, the 24-bit value in A*k* is transmitted to S*i* as an unnormalized floating-point quantity. The result is then 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 A*k*. If the sign bit of A*k* is set, the twos complement of A*k* is entered into S*i* as the magnitude of the coefficient and bit 2^{63} of S*i* is set for the sign of the coefficient.

A sequence of instructions is used to convert to floating-point format of an integer whose absolute value is less than 24 bits:

```
CAL code:  A1   S1
           S1   +FA1
           S1   +FS1      9 CPs required
```

When the *j* designator is 3, the floating-point constant of 0.75 x 2^{48} is entered into S*i* (4006060000000000000000_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 is used for conversion of an integer in S1:

```
CAL code:   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 S*i*.

When the *j* designator is 5, the floating-point constant 1.0 (=0 40001 4000 0000 0000 0000_8) is entered into S*i*.

When the *j* designator is 6, the floating-point constant 2.0 (=0 40002 4000 0000 0000 0000_8) is entered into S*i*.

When the *j* designator is 7, the floating-point constant 4.0 (=0 40003 4000 0000 0000 0000_8) is entered into S*i*.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process S*i* register access conflict S*i* reserved A*k* reserved (except A0); applies to all forms of the instruction, that is, *j* designators 0 through 7.
EXECUTION TIME:	Instruction issue, 1 CP S*i* ready, 2 CPs
SPECIAL CASES:	(A*k*)=1 if *k*=0. (S*i*)=(A*k*) if *j*=0. (S*i*)=(A*k*) sign extended if *j*=1. (S*i*)=(A*k*) unnormalized if *j*=2. (S*i*)=0.6 x 2^{60} (octal) if *j*=3. (S*i*)=0.4 x 2^{0} (octal) if *j*=4. (S*i*)=0.4 x 2^{1} (octal) if *j*=5. (S*i*)=0.4 x 2^{2} (octal) if *j*=6. (S*i*)=0.4 x 2^{3} (octal) if *j*=7.

CAL Syntax	Description	Octal Code
S*i* RT	Transmit (RTC) to S*i*	072*ixx*
S*i* VM	Transmit (VM) to S*i*	073*ixx*
S*i* T*jk*	Transmit (T*jk*) to S*i*	074*ijk*
T*jk* S*i*	Transmit (S*i*) to T*jk*	075*ijk*

Instructions 072 through 075 transmit register values to S*i* except for instruction 075 which transmits (S*i*) to T*jk*.

Instruction 072 enters the 64-bit value of the real-time clock (RTC) into S*i*. The clock is incremented by 1 each CP. The RTC is set only by the monitor through use of instruction 0014.

Instruction 073 enters the 64-bit value of the VM register into S*i*. The VM register is usually read after having been set by instruction 175.

Instruction 074 enters the contents of T*jk* into S*i*.

Instruction 075 enters the contents of S*i* into T*jk*.

HOLD ISSUE CONDITIONS: Instructions 034 through 037 in process
Exchange in process
S*i* register access conflict (instructions 072, 073, and 074)
S*i* reserved
For instruction 073:
  Instruction 175 in process, VM busy (VL)+6 CPs
  Instruction 003 in process, VM not available until 6 CPs after instruction 003 issues

EXECUTION TIME: Instruction issue, 1 CP
For instructions 072 through 074, S*i* ready, 1 CP
For instruction 075, T*jk* ready, 1 CP

SPECIAL CASES: None

CAL Syntax	Description	Octal Code
S*i* V*j*,A*k*	Transmit (V*j* element (A*k*)) to S*i*	076*ijk*
V*i*,A*k* S*j*	Transmit (S*j*) to V*i* element (A*k*)	077*ijk*
V*i*,A*k* 0†	Clear V*i* element (A*k*)	077*i*0*k*

Instructions 076 and 077 transmit a 64-bit quantity between a V register element and an S register.

Instruction 076 transmits the contents of an element of register V*j* to S*i*.

Instruction 077 transmits the contents of register S*j* to an element of register V*i*.

The low-order 6 bits of (A*k*) determine the vector element for either instruction.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A*k* reserved (except A0) S*i* register access conflict (instruction 076 only) For instruction 076, S*i* and V*j* reserved For instruction 077, V*i* and S*j* reserved
EXECUTION TIME:	Instruction issue, 1 CP For instruction 076, S*i* ready, 5 CPs For instruction 077, V*i* ready, 1 CP
SPECIAL CASES:	(S*j*)=0 if *j*=0. (A*k*)=1 if *k*=0.

† Special CAL syntax form

CAL Syntax	Description	Octal Code
A*i* *exp*,A*h*	Read from ((A*h*)+*jkm*) to A*i*	10*hijkm*
A*i* *exp*,0†	Read from (*jkm*) to A*i*	100*ijkm*
A*i* *exp*,†	Read from (*jkm*) to A*i*	100*ijkm*
A*i* ,A*h*†	Read from (A*h*) to A*i*	10*hi*00 0
exp,A*h* A*i*	Store (A*i*) to (A*h*)+*jkm*	11*hijkm*
exp,0 A*i*†	Store (A*i*) to *jkm*	110*ijkm*
exp, A*i*†	Store (A*i*) to *exp*	110*ijkm*
,A*h* A*i*†	Store (A*i*) to (A*h*)	11*hi*00 0
S*i* *exp*,A*h*	Read from ((A*h*)+*jkm*) to S*i*	12*hijkm*
S*i* *exp*,0†	Read from (*exp*) to S*i*	120*ijkm*
S*i* *exp*,†	Read from (*exp*) to S*i*	120*ijkm*
S*i* ,A*h*†	Read from (A*h*) to S*i*	12*hi*00 0
exp,A*h* S*i*	Store (S*i*) to (A*h*)+*jkm*	13*hijkm*
exp,0 S*i*†	Store (S*i*) to *exp*	130*ijkm*
exp, S*i*†	Store (S*i*) to *exp*	130*ijkm*
,A*h* S*i*†	Store (S*i*) to (A*h*)	13*hi*00 0

The 2-parcel instructions 10*h* through 13*h* transmit data between memory and an A register or an S register. The content of A*h* (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, (A*h*) 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 functional unit.

† Special CAL syntax form

Instructions 10*h* and 11*h* 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 A*i* into memory, the high-order 40 bits of the memory word are zeroed. Instructions 12*h* and 13*h* transmit 64-bit quantities to or from register S*i*.

HOLD ISSUE CONDITIONS:

Instructions 034 through 037 in process
Exchange in process
Four I/O memory reference requests with none honored
Rank B, C, D, E, or F bank conflict
Storage hold continuation
A*h* reserved
For instruction 10*h*, A*i* register access conflict
For instructions 10*h* and 11*h*, A*i* reserved
For instructions 12*h* and 13*h*, S*i* reserved
For instruction 12*h*, S*i* register access conflict
Fetch request in previous CP
Instruction 176 in process, unit busy (VL)+8 CPs
Instruction 177 in process, unit busy (VL)+9 CPs

EXECUTION TIME:

Instruction issue:
  Both parcels in same buffer, 2 CPs
  Parcels in different buffers, 4 CPs
  Second parcel not in a buffer, 17 CPs
  For instruction 10*h*, A*i* ready, 13 CPs
  For instruction 12*h*, S*i* ready, 13 CPs
Memory ready for next scalar read or store, 7 CPs

SPECIAL CASES:

For instructions 10*h* and 12*h*:
  Rank B conflict, 5 CPs delay before A*i* or S*i* ready
  Rank C conflict, 4 CPs delay before A*i* or S*i* ready
  Rank D conflict, 3 CP delay before A*i* or S*i* ready
  Rank E conflict, 2 CP delay before A*i* or S*i* ready
  Rank F conflict, 1 CP delay before A*i* or S*i* ready

For instruction 12*h*:
  Hold storage, 1 CP delay if 070 register access conflict occurs (when the result entering coincides with a reciprocal approximation result entering S*i*).

INSTRUCTIONS 140 - 147

CAL Syntax	Description	Octal Code
V*i* S*j*&V*k*	Logical products of (S*j*) and (V*k* elements) to V*i* elements	140*ijk*
V*i* V*j*&V*k*	Logical products of (V*j* elements) and (V*k* elements) to V*i* elements	141*ijk*
V*i* S*j*!V*k*	Logical sums of (S*j*) and (V*k* elements) to V*i* elements	142*ijk*
V*i* V*k*†	Transmit (V*k* elements) to V*i* elements	142*i*0*k*
V*i* V*j*!V*k*	Logical sums of (V*j* elements) and (V*k* elements) to V*i* elements	143*ijk*
V*i* S*j*\V*k*	Logical differences of (S*j*) and (V*k* elements) to V*i* elements	144*ijk*
V*i* V*j*\V*k*	Logical differences of (V*j* elements) and (V*k* elements) to V*i* elements	145*ijk*
V*i* 0†	Clear V*i* elements	145*iii*
V*i* S*j*!V*k*&VM	If VM bit=1, transmit (S*j*) to the corresponding element in V*i*. If VM bit=0, transmit the (corresponding V*k* element) to the (corresponding V*i* element).	146*ijk*
V*i* #VM&V*k*†	If VM bit=1, transmit (0) to the corresponding element in V*i*. If VM bit=0, transmit the (corresponding V*k* element) to the (corresponding V*i* element).	146*i*0*k*
V*i* V*j*!V*k*&VM	If VM bit=1, transmit the (corresponding V*j* element) to the (corresponding V*i* element). If VM bit=0, transmit the (corresponding V*k* element) to the (corresponding V*i* element).	147*ijk*

Instructions 140 through 147 are executed in the Vector Logical functional unit. The number of operations performed is determined by the contents of the VL register. All operations start with element 0 of the V*i*, V*j*, or V*k* register and increment the element number by 1 for each operation performed. All results are delivered to V*i*.

† Special CAL syntax form

For instructions 140, 142, 144, and 146, a copy of the content of S*j* is delivered to the functional unit. The copy of the content is held as one of the operands until completion of the operation. Therefore, S*j* can be changed immediately without affecting the vector 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 operand pairs and enter the result into V*i*. Bits of an element of V*i* are set to 1 when the corresponding bits of (S*j*) or (V*j* element) and (V*k* 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
```

Instructions 142 and 143 form the logical sums (inclusive OR) of operand pairs and deliver the results to V*i*. Bits of an element of V*i* are set to 1 when one of the corresponding bits of (S*j*) or (V*j* element) and (V*k* 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
```

Instructions 144 and 145 form the logical differences (exclusive OR) of operand pairs and deliver the results of V*i*. Bits of an element are set to 1 when the corresponding bit of (S*j*) or (V*j* element) is different from (V*k* 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
```

Instructions 146 and 147 transmit operands to V*i* depending on the contents of the VM register. 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 instruction 146, the first operand is always (S*j*), the second operand is (V*k* element). For instruction 147, the first operand is (V*j* element) and the second operand is (V*k* 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, (V*k* element), is selected.

EXAMPLES:

1. If instruction 146 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. If instruction 147 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.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process V*i* or V*k* reserved Instruction 14*x* in process, unit busy (VL)+4 CPs Instruction 175 in process, unit busy (VL)+4 CPs Instruction 003 in process, unit busy 3 CPs For instructions 140, 142, 144, and 146, S*j* reserved For instructions 141, 143, 145, and 147, V*j* reserved
EXECUTION TIME:	Instruction issue, 1 CP V*i* ready, 9 CPs if (VL) is less than or equal to 5 V*i* ready, (VL)+4 CPs if (VL) greater than 5 V*j* or V*k* ready, 5 CPs if (VL) is less than or equal to 5 V*j* or V*k* ready, (VL) CPs if (VL) greater than 5 Unit ready, (VL) + 4 CPs Chain slot ready, 4 CPs
SPECIAL CASES:	(S*j*)=0 if *j*=0. For instruction 145, if *i*=*j*=*k*, (V*i*)=0.

CAL Syntax		Description	Octal Code
V*i*	V*j*<A*k*	Shift (V*j*) elements left by (A*k*) places to V*i* elements	150*ijk*
V*i*	V*j*<1†	Shift (V*j*) elements left one place to V*i* elements	150*ij*0
V*i*	V*j*>A*k*	Shift (V*j*) elements right by (A*k*) places to V*i* elements	151*ijk*
V*i*	V*j*>1†	Shift (V*j*) elements right one place to V*i* elements	151*ij*0

Instructions 150 and 151 are executed in the Vector Shift functional unit. The number of operations performed is determined by the contents of the VL register. Operations start with element 0 of the V*i* and V*j* registers and end with elements specified by (VL)-1.

All shifts are end off with zero fill. The shift count is obtained from (A*k*) and elements of V*i* are cleared if the shift count exceeds 63. All shift counts (A*k*) are considered positive. All 24 bits of A*k* are used for the shift count.

Unlike shift instructions 052 through 055, these instructions receive the shift count from A*k*, rather than the *jk* fields.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process V*i* or V*j* reserved A*k* reserved (except A0) Instructions 150 through 153 in process, unit busy (VL)+4 CPs
EXECUTION TIME:	Instruction issue, 1 CP V*i* ready, 11 CPs if (VL) is less than or equal to 5 V*i* ready, (VL)+6 CPs if (VL) greater than 5 V*j* ready, 5 CPs if (VL) is less than or equal to 5 V*j* ready, (VL) CPs if (VL) greater than 5 Unit ready, (VL)+4 CPs Chain slot ready, 6 CPs
SPECIAL CASE:	(A*k*)=1 if *k*=0.

† Special CAL syntax form

CAL Syntax	Description	Octal Code
V*i* V*j*,V*j*<A*k*	Double shifts of (V*j* elements) left (A*k*) places to V*i* elements	152*ijk*
V*i* V*j*,V*j*<1†	Double shifts of (V*j* elements) left one place to V*i* elements	152*ij*0
V*i* V*j*,V*j*>A*k*	Double shifts of (V*j* elements) right (A*k*) places to V*i* elements	153*ijk*
V*i* V*j*,V*j*>1†	Double shifts of (V*j* elements) right one place to V*i* elements	153*ij*0

Instructions 152 and 153 are executed in the Vector Shift functional unit. The instructions shift 128-bit values formed by logically joining the contents of two elements of the V*j* register. The direction of the shift determines whether the high-order bits or the low-order bits of the result are sent to V*i*. Shift counts are obtained from register A*k*.

All shifts are end off with zero fill.

The number of operations is determined by the contents of the VL register.

Instruction 152 performs left shifts. The operation starts with element 0 of V*j*. If (VL) is 1, element 0 is joined with 64 bits of 0, and the resulting 128-bit quantity is then shifted left by the amount specified by (A*k*). Only the one operation is performed. The 64 high-order bits remaining are transmitted to element 0 of V*i*.

If (VL) is 2, the operation starts with element 0 of V*j* being joined with element 1, and the resulting 128-bit quantity is then shifted left by the amount specified by (A*k*). The high-order 64 bits remaining are transmitted to element 0 of V*i*. Figure 7-6 illustrates this operation.

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 V*i*. Figure 7-7 illustrates this operation.

If (VL) is 2, element 1 is joined with 64 bits of 0 and only two operations are performed. In general, the last element of V*j* as determined by (VL) is joined with 64 bits of zeros. Figure 7-8 illustrates this operation.

† Special CAL syntax form

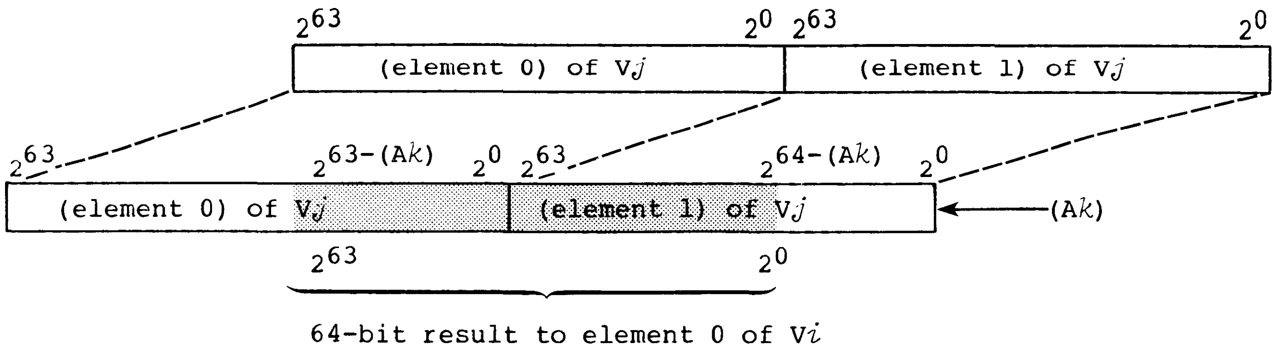


Figure 7-6. Vector left double shift, first element, VL greater than 1

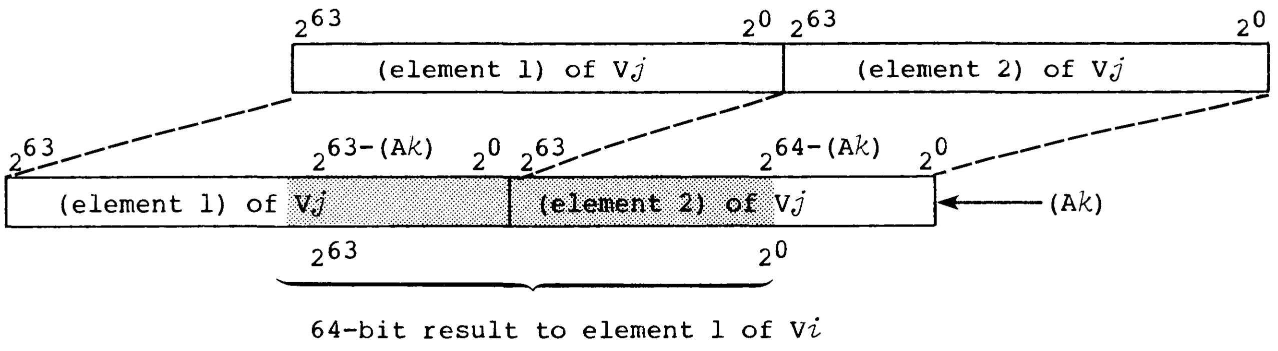


Figure 7-7. Vector left double shift, second element, VL greater than 2

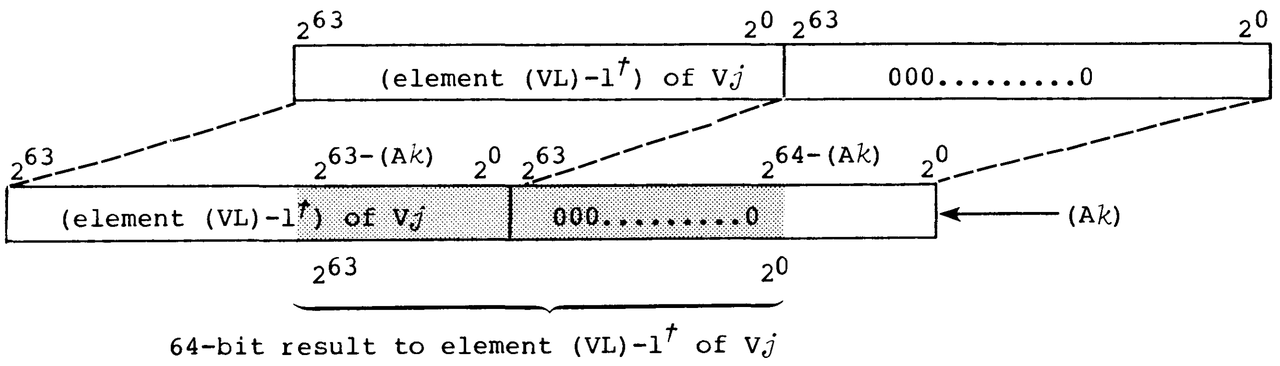


Figure 7-8. Vector left double shift, last element

† Elements are numbered 0 through 63 in the V registers; therefore, element (VL)-1 refers to the VLth element.

If (A*k*) is greater than 127, the result is all zeros. If (A*k*) is greater than 64 and less than 128, the result register contains at least (A*k*)-64 zeros.

EXAMPLES: If instruction 152 is to be executed and the following register conditions exist:

```
(VL)   = 4
(A1)   = 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 V5 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
```

Instruction 153 performs right shifts. Element 0 of V*j* is joined with 64 high-order bits of 0 and the 128-bit quantity is shifted right by the amount specified by (A*k*). The 64 low-order bits of the result are transmitted to element 0 of V*i*. Figure 7-9 illustrates this operation.

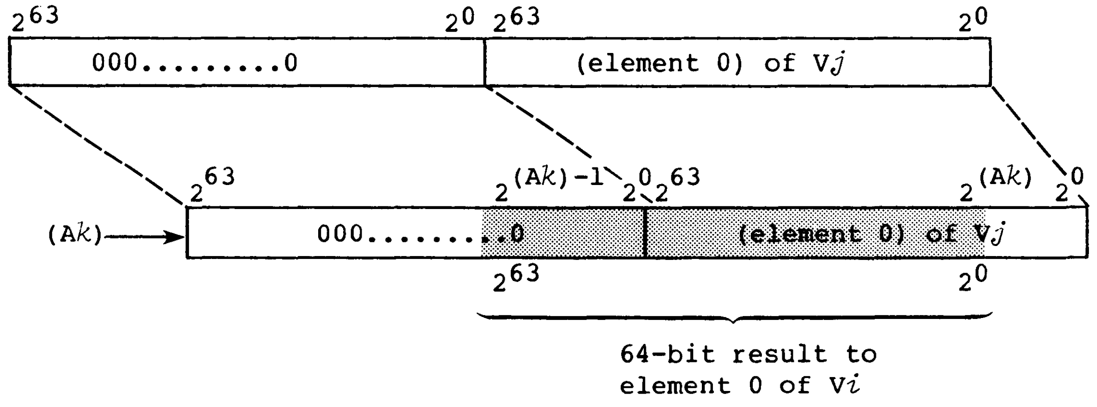


Figure 7-9. Vector right double shift, first element

EXAMPLES: (continued)

If (VL)=1, only one operation is performed. In general, however, instruction execution continues by joining element 0 with element 1, shifting the 128-bit quantity by the amount specified by (A*k*), and transmitting the result to element 1 of V*i*. This operation is shown in figure 7-10.

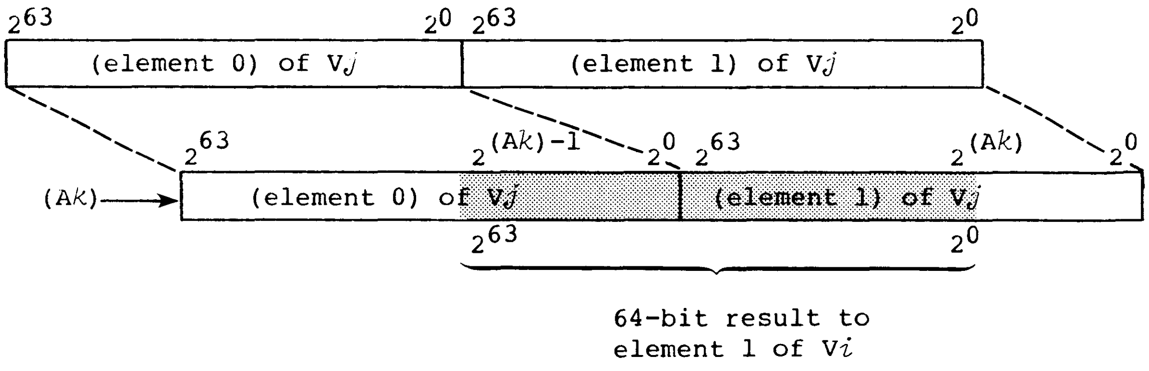


Figure 7-10. Vector right double shift, second element, VL greater than 1

The last operation performed by the instruction joins the last element of V*j* as determined by (VL) with the preceding element. Figure 7-11 illustrates this operation.

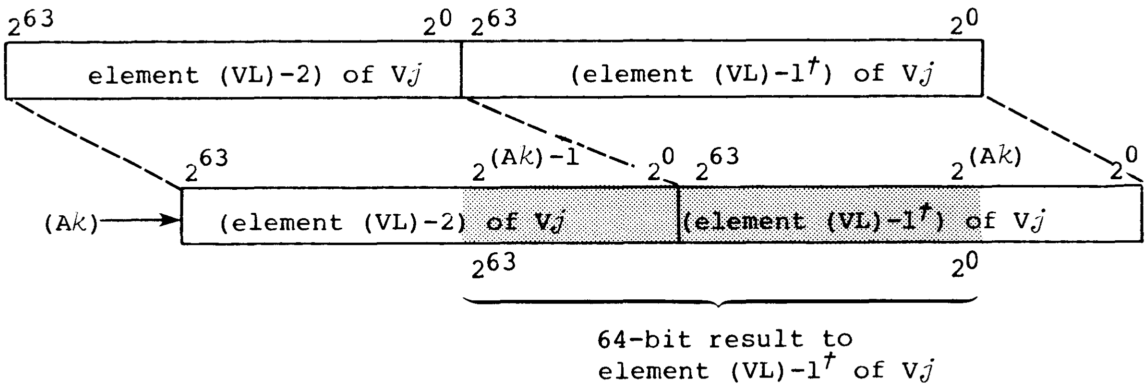


Figure 7-11. Vector right double shift, last operation

† Elements are numbered 0 through 63 in the V registers; therefore, element (VL)-1 refers to the VLth element.

EXAMPLES: (continued)	If an instruction 153 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:

Instructions 034 through 037 in process
Exchange in process
V*i* or V*j* reserved
A*k* reserved (except A0)
Instructions 150 through 153 in process, unit busy (VL)+4 CPs

EXECUTION TIME:

Instruction issue, 1 CP
V*i* ready, 11 CPs if (VL) is less than or equal to 5
V*i* ready, (VL)+6 CPs if (VL) is greater than 5
V*j* ready, 5 CPs if (VL) is less than or equal to 5
V*j* ready, (VL) CPs if (VL) is greater than 5
Unit ready, (VL)+4 CPs
Chain slot ready, 6 CPs

SPECIAL CASE: (A*k*)=1 if *k*=0.

CAL Syntax		Description	Octal Code
V*i*	S*j*+V*k*	Integer sums of (S*j*) and (V*k* elements) to V*i* elements	154*ijk*
V*i*	V*j*+V*k*	Integer sums of (V*j* elements) and (V*k* elements) to V*i* elements	155*ijk*
V*i*	S*j*-V*k*	Integer differences of (S*j*) and (V*k* elements) to V*i* elements	156*ijk*
V*i*	-V*k*†	Transmit negative of (V*k* elements) to V*i* elements	156*i*0*k*
V*i*	V*j*-V*k*	Integer differences of (V*j* elements) and (V*k* elements) to V*i* elements	157*ijk*

Instructions 154 through 157 are executed in the Vector Add functional unit.

Instructions 154 and 155 perform integer addition. Instructions 156 and 157 perform integer subtraction. The number of additions or subtractions 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 V*i*. No overflow is detected.

Instructions 154 and 156 deliver a copy of (S*j*) 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 V*k*. For instructions 155 and 157, both operands are obtained from V registers.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process
	Exchange in process
	V*i* or V*k* reserved
	Instructions 154 through 157 in process, unit busy (VL)+4 CPs
	For instructions 154 and 156, S*j* reserved (except S0)
	For instructions 155 and 157, V*j* reserved

† Special CAL syntax form

EXECUTION TIME:	Instruction issue, 1 CP V*i* ready, 10 CPs if (VL) is less than or equal to 5 V*i* ready, (VL)+5 CPs if (VL) is greater than 5 V*j* or V*k* ready, 5 CPs if (VL) is less than or equal to 5 V*j* or V*k* ready, (VL) CPs if (VL) is greater than 5 Unit ready, (VL)+4 CPs Chain slot ready, 5 CPs
SPECIAL CASES:	For instruction 154, if *j*=0, then (S*j*)=0 and (V*i* element)=(V*k* element). For instruction 156, if *j*=0, then (S*j*)=0 and (V*i* element)= -(V*k* element).

CAL Syntax		Description	Octal Code
V*i*	S*j**FV*k*	Floating-point products of (S*j*) and (V*k* elements) to V*i* elements	160*ijk*
V*i*	V*j**FV*k*	Floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements	161*ijk*
V*i*	S*j**HV*k*	Half-precision rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements	162*ijk*
V*i*	V*j**HV*k*	Half-precision rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements	163*ijk*
V*i*	S*j**RV*k*	Rounded floating-point products of (S*j*) and (V*k* elements) to V*i* elements	164*ijk*
V*i*	V*j**RV*k*	Rounded floating-point products of (V*j* elements) and (V*k* elements) to V*i* elements	165*ijk*
V*i*	S*j**IV*k*	Reciprocal iterations; 2-(S*j*) * (V*k* elements) to V*i* elements	166*ijk*
V*i*	V*j**IV*k*	Reciprocal iterations; 2-(V*j* elements) * (V*k* elements) to V*i* elements	167*ijk*

Instructions 160 through 167 are executed in the Floating-point Multiply functional 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. Instructions 160, 162, 164, and 166 deliver a copy of (S*j*) to the functional unit where the copy is retained as one of the operands until the completion of the operation. Therefore, S*j* can be changed immediately without affecting the vector operation. The other operand is an element of V*k*. For instructions 161, 163, 165, and 167, both operands are obtained from V registers.

All results are delivered to elements of V*i*. If neither operand is normalized, there is no guarantee that the products will be normalized.

Out-of-range conditions are described in section 5.

Instruction 160 forms the products of the floating-point quantity in S*j* and the floating-point quantities in elements of V*k* and enters the results into V*i*.

Instruction 161 forms the products of the floating-point quantities in elements of V*j* and V*k* and enters the results into V*i*.

Instruction 162 forms the half-precision rounded products of the floating-point quantity in S*j* and the floating-point quantities in elements of V*k* and enters the results into V*i*. The low-order 19 bits of the result elements are zeroed.

Instruction 163 forms the half-precision rounded products of the floating-point quantities in elements of V*j* and V*k* and enters the results into V*i*. The low-order 19 bits of the result elements are zeroed.

Instruction 164 forms the rounded products of the floating-point quantity in S*j* and the floating-point quantities in elements of V*k* and enters the results into V*i*.

Instruction 165 forms the rounded products of the floating-point quantities in elements of V*j* and V*k* and enters the results into V*i*.

Instruction 166 forms for each element, two minus the product of the floating-point quantity in S*j* and the floating-point quantity in elements of V*k*. It then enters the results into V*i*. See the description of instruction 067 for more details.

Instruction 167 forms for each element pair, two minus the product of the floating-point quantities in elements of V*j* and V*k* and enters the results into V*i*. See the description of instruction 067 for more details.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process
	Exchange in process
	V*i* or V*k* reserved
	Instruction 16*x* in process, unit busy (VL)+4 CPs
	For instructions 160, 162, 164, and 166, S*j* reserved
	For instructions 161, 163, 165, and 167, V*j* reserved

EXECUTION TIME:	Instruction issue, 1 CP V*i* ready, 14 CPs if (VL) is less than or equal to 5 V*i* ready, (VL)+9 CPS if (VL) is greater than 5 V*j* or V*k* ready, 5 CPS if (VL) is less than or equal to 5 V*j* or V*k* ready, (VL) CPs if (VL) is greater than 5 Unit ready, (VL)+4 CPs Chain slot ready, 9 CPs
SPECIAL CASE:	(S*j*)=0 if *j*=0.

CAL Syntax		Description	Octal Code
V*i*	S*j*+FV*k*	Floating-point sums of (S*j*) and (V*k* elements) to V*i* element	170*ijk*
V*i*	+FV*k*†	Transmit normalized (V*k* elements) to V*i* elements	170*i*0*k*
V*i*	V*j*+FV*k*	Floating-point sums of (V*j* elements) and (V*k* elements) to V*i* elements	171*ijk*
V*i*	S*j*-FV*k*	Floating-point differences of (S*j*) and (V*k* elements) to V*i* elements	172*ijk*
V*i*	-FV*k*†	Transmit normalized negatives of (V*k* elements) to V*i* elements	172*i*0*k*
V*i*	V*j*-FV*k*	Floating-point differences of (V*j* elements) and (V*k* elements) to V*i* elements	173*ijk*

Instructions 170 through 173 are executed in the Floating-point Add functional 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 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 V*i* normalized and results are normalized even if the operands are not normalized.

Instructions 170 and 172 deliver a copy of (S*j*) to the functional unit where it remains as one of the operands until the completion of the operation. The other operand is an element of V*k*. For instructions 171 and 173, both operands are obtained from V registers. Out-of-range conditions are described in section 5.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process
	Exchange in process
	V*i* or V*k* reserved
	Instructions 170 through 173 in process, unit busy (VL) + 4 CPs
	For instructions 170 and 172, S*j* reserved (except S0)
	For instructions 171 and 173, V*j* reserved

† Special CAL syntax form

EXECUTION TIME:	Instruction issue, 1 CP V*i* ready, 13 CPs if (VL) is less than or equal to 5 V*i* ready, (VL)+8 CPs if (VL) is greater than 5 V*j* and V*k* ready, 5 CPs if (VL) is less than or equal to 5 V*j* and V*k* ready, (VL) CPs if (VL) is greater than 5 Unit ready, (VL)+4 CPs Chain slot ready, 8 CPs
SPECIAL CASE:	(S*j*)=0 if *j*=0.

INSTRUCTION 174

CAL Syntax	Description	Octal Code
V*i* /HV*j*	Floating-point reciprocal approximation of (V*j* elements) to V*i* elements	174*ij*0

Instruction 174 is executed in the Reciprocal Approximation functional unit. The instruction forms an approximate value of the reciprocal of the normalized floating-point quantity in each element of V*j* and enters the result into elements of V*i*. The number of elements for which approximations are found is determined by the contents of the VL register.

Instruction 174 occurs in the divide sequence to compute the quotients of floating-point quantities as described in section 5 under floating-point arithmetic.

The reciprocal approximation instruction produces a result of 30 significant bits. The low-order 18 bits are zeros. The number of significant bits can be extended to 48 using the reciprocal iteration instruction and a multiply.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process V*i* or V*k* reserved Instruction 174 in process, unit busy for (VL) + 4 CPs
EXECUTION TIME:	Instruction issue, 1 CP V*i* ready, 21 CPs if (VL) is less than or equal to 5 V*i* ready, (VL)+16 CPs if (VL) is greater than 5 V*j* ready, 5 CPs if (VL) is less than or equal to 5 V*j* ready, (VL) CPs if (VL) is greater than 5 Unit ready, (VL)+4 CPs Chain slot ready, 16 CPs
SPECIAL CASE:	(V*i* element) is meaningless if (V*j* element) is not normalized; the unit assumes that bit 2^{47} of (V*j* element) is 1; no test of this bit is made.

INSTRUCTIONS 174*ij*1 - 174*ij*2

CAL Syntax	Description	Octal Code
V*i* PV*j*	Population count of (V*j* elements) to V*i* elements	174*ij*1
V*i* QV*j*	Population count parity of (V*j* elements) to V*i* elements	174*ij*2

Instructions 174*ij*1 and 174*ij*2 are executed in the Vector Population/Parity functional unit, sharing some logic with the Reciprocal Approximation functional unit.

Instruction 174*ij*1 counts the number of bits set to 1 in each element of V*j* and enters the results into corresponding elements of V*i*. The results are entered into the low-order 7 bits of each V*i* element; the remaining high-order bits of each V*i* element are zeroed.

Instruction 174*ij*2 counts the number of bits set to 1 in each element of V*j*. The least significant bit of each element result shows whether the result is an odd or even number. Only the least significant bit of each element is transferred to the least significant bit position of the corresponding element of register V*i*. The remainder of the element is set to zeros. The actual population count results are not transferred.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process V*i* reserved V*k* reserved Instruction 174 in process; unit busy for (VL)+4 CPs
EXECUTION TIME:	Instruction issue, 1 CP V*i* ready, 13 CPs if (VL) is less than or equal to 5 V*i* ready, (VL)+8 CPs if (VL) is greater than 5 V*j* ready, 5 CPs if (VL) is less than or equal to 5 V*j* ready, (VL) CPs if (VL) is greater than 5 Unit ready, (VL)+4 CPs Chain slot ready, 8 CPs
SPECIAL CASES:	None

CAL Syntax	Description	Octal Code
VM V*j*,Z	VM=1 when (V*j* element)=0	175*xj*0
VM V*j*,N	VM=1 when (V*j* element)≠0	175*xj*1
VM V*j*,P	VM=1 when (V*j* element) positive, (bit 2^{63}=0), includes (V*j* element)=0	175*xj*2
VM V*j*,M	VM=1 when (V*j* element) negative, (bit 2^{63}=1)	175*xj*3

Vector mask instruction 175 is executed in the Vector Logical functional unit.

Instruction 175*xjk* creates a vector mask in VM based on the results of testing the contents of the elements of register V*j*. Each bit of VM corresponds to an element of V*j*. 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 (V*j* element) is 0 and is set to 0 when (V*j* element) is nonzero.

If the *k* designator is 1, the VM bit is set to 1 when (V*j* element) is nonzero and is set to 0 when (V*j* element) is 0.

If the *k* designator is 2, the VM bit is set to 1 when (V*j* element) is positive and is set to 0 when (V*j* element) is negative. A zero value is considered positive.

If the *k* designator is 3, the VM bit is set to 1 when (V*j* element) is negative and is set to 0 when (V*j* 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 V*j* are zeroed.

Vector mask instruction 175 provides a vector counterpart to the scalar conditional branch instructions.

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process V*j* reserved Instruction 14*x* in process, unit busy (VL)+4 CPs Instruction 003 in process, VM busy 3 CPs Instruction 175 in process, unit busy (VL)+4 CPs
EXECUTION TIME:	Instruction issue, 1 CP V*j* ready, 5 CPs if (VL) is less than or equal to 5 V*j* ready, (VL) CPs if (VL) is greater than 5 Except for instruction 073, VM ready (VL)+4 CPs For instruction 073, VM ready (VL)+6 CPs
SPECIAL CASES:	*k*=0 or 4, VM bit *xx*=1 if (V*j* element *xx*)=0. *k*=1 or 5, VM bit *xx*=1 if (V*j* element *xx*)≠0. *k*=2 or 6, VM bit *xx*=1 if (V*j* element *xx*) is positive. *k*=3 or 7, VM bit *xx*=1 if (V*j* element *xx*) is negative.

CAL Syntax	Description	Octal Code
V*i* ,A0,A*k*	Transmit (VL) words from memory to V*i* elements starting at memory address (A0) and incrementing by (A*k*) for successive addresses	176*ixk*
V*i* ,A0,1†	Transmit (VL) words from memory to V*i* elements starting at memory address (A0) and incrementing by 1 for successive addresses	176*ix*0
,A0,A*k* V*j*	Transmit (VL) words from V*j* elements to memory starting at memory address (A0) and incrementing by (A*k*) for successive addresses	177*xjk*
,A0,1 V*j*†	Transmit (VL) words from V*j* elements to memory starting at memory address (A0) and incrementing by 1 for successive addresses	177*xj*0

Instructions 176 and 177 transfer blocks of data between V registers and memory.

Instruction 176 transfers data from memory to elements of register V*i*.

Instruction 177 transfers data from elements of register V*j* to memory.

Register elements begin with 0 and are incremented by 1 for each transfer. Memory addresses begin with (A0) and are incremented by the contents of A*k*. A*k* contains a signed 22-bit integer which is added to the address of the current word to obtain the address of the next word. A*k* can specify either a positive or negative increment allowing both forward and backward streams of reference. The 2 high-order bits of (A*k*) are ignored.

The number of words transferred is determined by the contents of the VL register.

† Special CAL syntax form

HOLD ISSUE CONDITIONS:	Instructions 034 through 037 in process Exchange in process A0 reserved A*k* reserved where *k*=1 through 7 Block sequence flag set (instructions 034 through 037, 176, and 177) Scalar reference (3 CPs maximum) Rank B data valid Fetch request in last CP For instruction 176, V*i* reserved For instruction 177, V*j* reserved I/O memory request
EXECUTION TIME:	For instruction 176 (assuming no bank conflicts): Except for instructions 034 through 037, 100 through 137, 176, and 177, instruction issue 1 CP Instruction issue for instructions 034 through 037, 100 through 137, 176, and 177, (VL)+8 CPs V*i* ready, 16 CPs if (VL) is less than or equal to 5 V*i* ready, (VL)+9 CPs if (VL) is greater than 5 For instruction 177 (assuming no bank conflicts): Except for instructions 034 through 037, 100 through 137, 176, and 177, instruction issue 1 CP Instruction issue for instructions 034 through 037, 100 through 137, 176, and 177, (VL)+9 CPs V*j* ready, 5 CPs if (VL) is less than or equal to 5 V*j* ready, (VL) CPs if (VL) is greater than 5
SPECIAL CASES:	Increment, (A*k*), =1 if *k*=0. Chain slot issue is 11 CPs if full speed for instruction 176, blocked for instruction 177 Inhibit I/O references. Inhibit instructions 034 through 037, 100 through 137, 176, and 177.

SPECIAL CASES: (continued)

(Ak) determines speed control. Successive addresses are located in successive banks. References to the same bank can be made every 8 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 8 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 4 CPs. If the address is incremented by 4 (16 bank) or 2 (8 bank), every fourth reference is to the same bank and words can transfer every 2 CPs. With any address incrementing that allows 8 CPs before addressing the same bank, one word can transfer each CP.

APPENDIX SECTION

SUMMARY OF CPU TIMING INFORMATION — A

When issue conditions are satisfied, an instruction completes in a fixed amount of time (scalar memory references are exceptions). Instruction issue can cause reservations to be placed on a functional unit or registers. Knowledge of the issue conditions, instruction execution times, and reservations permits 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 available. No conflicts can arise with other scalar instructions; however, vector floating-point instructions reserve the floating-point units. Scalar memory references can be delayed due to conflicts.

2. The result register must be available.

3. The operand register must be available.

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 available at the time the results are stored. A previous instruction with a longer execution time could still be occupying the input path.

5. At least one of the last four I/O memory reference requests must have been honored.

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 (CPs) 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:

$A \leftarrow M$	13 CPs	$A \leftarrow C$	4 CPs
$M \leftarrow A$	1†,†† CP	$A \leftarrow A+A$	2 CPs
$A \leftarrow B$	1 CP	$A \leftarrow A*A$	6 CPs
$B \leftarrow A$	1 CP	$A \leftarrow$ pop(S)	4 CPs
$A \leftarrow S$	1 CP	$A \leftarrow$ lzc(S)	3 CPs
$A \leftarrow I$	1 CP	$VL \leftarrow A$	1 CP

64-bit results:

$S \leftarrow M$	13 CPs	$S \leftarrow S+S$	3 CPs
$M \leftarrow S$	1†,†† CPs	$S \leftarrow$ S(f add)S	6† CPs
$S \leftarrow T$	1 CP	$S \leftarrow$ S(f mult)S	7† CPs
$T \leftarrow S$	1 CP	$S \leftarrow$ S(ra)	14† CPs
$S \leftarrow I$	1 CP	$S \leftarrow V$	5 CPs
$S \leftarrow$ S(logical)S	1 CP	$V \leftarrow S$	3 CPs
$S \leftarrow$ S(shift)I	2 CPs	$S \leftarrow VM$	1 CP
$S \leftarrow$ S(shift)A	3 CPs	$S \leftarrow RTC$	1 CP
$S \leftarrow$ S(mask)I	1 CP	$S \leftarrow A$	2 CPs
$RTC \leftarrow S$	1 CP	$VM \leftarrow S$	3 CPs

The following is an example of the use of this chart of execution times to optimize timing.

† Issue can be delayed because of a functional unit reservation by a vector instruction. Memory can be considered a functional unit for timing considerations.

†† A*i* to memory or S*i* to memory instructions free the source register in 1 CP. However, the instructions are 2-parcel instructions and take 2 CPs to cycle through the CIP register before another instruction can issue.

	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 available. (Conflicts can occur with vector operations.)

2. The result register must be available. (Conflicts can occur with vector operations.)

3. The operand registers must be available 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) + 9 CPs on a write operation, (VL) + 8 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 can issue, that is, has met all other issue conditions. This process is called chaining. Several vector instructions using different functional units can 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 instruction (177) is reserved for (VL) clock periods. Scalar operand registers are not reserved.

Vector instructions produce one result per CP. The functional unit times are given below. The vector read and write instructions (176 and 177) produce results more slowly if bank conflicts arise due to the increment value (A*k*) being a multiple of 2 (4 for 16-bank phasing). Chaining cannot occur for the vector read operation in this case.

If (A*k*) is an odd multiple of 2 (4 for 16-bank phasing), results are produced every 2 CPs.

If (A*k*) is an odd multiple of 4 (8 for 16-bank phasing), results are produced every 4 CPs.

If (A*k*) is an even multiple of 4 (8 for 16-bank phasing), results are produced every 8 CPs.

Functional unit	Time (CPs)
Vector Logical	2
Vector Shift	4
Vector Integer Add	3
Floating-point Add	6
Floating-point Multiply	7
Reciprocal Approximation	14
Memory	8, 9, or 10
Vector Population/Parity	6

A transmit vector mask to S*i* instruction (073) is delayed by (VL) + 6 CPs from the issue of a previous vector mask instruction (175) and is delayed by 6 CPs from the issue of the preceding transmit (S*j*) to VM instruction (003).

HOLD ISSUE

A delay of issue results if an instruction 100 through 137 is in the CIP register and a hold memory condition exists (see following subsection on hold memory). The delay depends on the hold memory delay.

Memory must be quiet before issue of the B and T register block copy instructions (034-037). The low-order 7 bits (A*i*) affect the timing. Subsequent instructions cannot issue for 16 + (A*i*) CPs if (A*i*)≠0 and 5 CPs if (A*i*)=0 when reading data to the B and T registers (instructions 034 and 036). The subsequent instructions cannot issue for 10 + (A*i*) CPs when storing data (instructions 035 and 037).

The B and T register block read instructions (034 and 036) 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 available 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 18 CPs (26 CPs for 8-bank phasing).

A 2-parcel instruction takes a minimum of 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 16 CPs (24 CPs for 8-bank phasing) if the next instruction parcel is not in an instruction buffer.

HOLD MEMORY

A delay of 5, 4, 3, 2, or 1 CPs is added to an A or S register memory read if a bank conflict occurs with rank B, C, D, E, or F respectively. A conflict occurs if the address is in the same bank as the address in rank B, C, D, E, or F. An additional 1 CP delay is added to a hold memory condition if an instruction 070 destination register conflict is sensed.

Conflicts can occur only with scalar references. The scalar instruction senses the conflict condition at issue time + 2 CPs. The scalar instruction address enters rank B at issue + 2 CPs. The scalar instruction address enters rank C at issue + 3 CPs. The scalar instruction address enters rank D at issue + 4 CPs. The scalar instruction address enters rank E at issue + 5 CPs. The scalar instruction address enters rank F at issue + 6 CPs.

INTERRUPT TIMING

After a sensed interrupt condition, a minimum of 3 CPs + two parcel issues must occur before the interrupt is generated. During the first 3 CPs, if no hold issue conditions exist, instruction parcels can issue. At the end of the 3 CPs, the NIP register parcel is examined. If the NIP instruction is a 2-parcel instruction, three parcel issues occur before the interrupt. If the NIP instruction is a 1-parcel instruction, only two parcel issues occur before the interrupt.

PHYSICAL ORGANIZATION OF THE MAINFRAME

B

MAINFRAME

The CRAY-1 M mainframe is 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 6-column mainframe extends 135° around the arc. The columns are approximately 6.5 feet tall. At the base of the columns are cabinets for power supplies and cooling distribution systems. These cabinets are 1.5 feet high and extend outward approximately 2.5 feet.

Viewing the mainframe from the top, the upper chassis are labeled D through I proceeding counterclockwise. In the same manner, the lower chassis are named P through U. The general chassis layout is shown in figure B-2. In an 8-bank machine, the I and U chassis are not used.

Physical characteristics of the of the CRAY-1 M are summarized below.

- Dimensions
  Base - approximately 8 feet by 3.5 feet by 1.5 feet high
  Columns - approximately 5 feet by 2 feet by 6.5 feet high
  including height of base

- 12 logic chassis arranged two per column in 6 columns

- Approximately 750 modules (maximum memory size)

- Approximately 130 standard module types

- Up to 576 IC packages per module

- Power consumption approximately 50 kW input for maximum memory size

- Refrigerant-22 cooled with refrigerant/water heat exchange

- Three memory options

- Weight 5,250 lbs (maximum memory size)

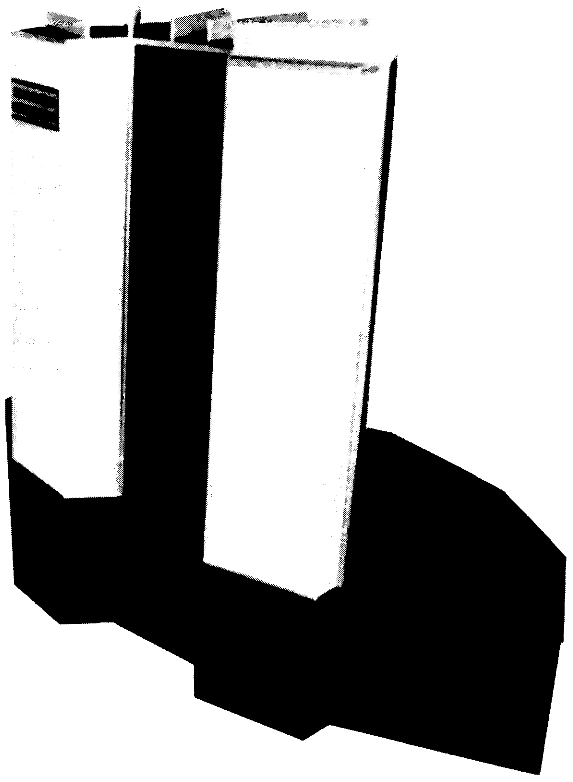


Figure B-1. CRAY-1 M mainframe

MODULES

The CRAY-1 M Computer System uses a basic module construction throughout the machine. The module consists of two or four 6 x 8 inch printed circuit boards mounted on opposite sides of a heavy copper heat transfer plate. Each printed circuit module has capacity for a maximum of 288 or 576 integrated circuit (IC) packages and approximately 600 resistor packages.

A 2-million or 4-million word mainframe has 748 modules. Modules are arranged up to 72 per chassis as illustrated in figure B-2. There are over 130 module types with usage varying from 1 to 144 modules per type. Each module type is identified by two letters, the first indicating the module series (A, D, F, G, H, J, M, R, S, T, V, Y, and Z) and the second letter identifying the type of module within a series.

The computation and I/O modules are on the eight chassis forming the center four columns. Each of the two chassis on either side of the four center columns contains four memory banks.

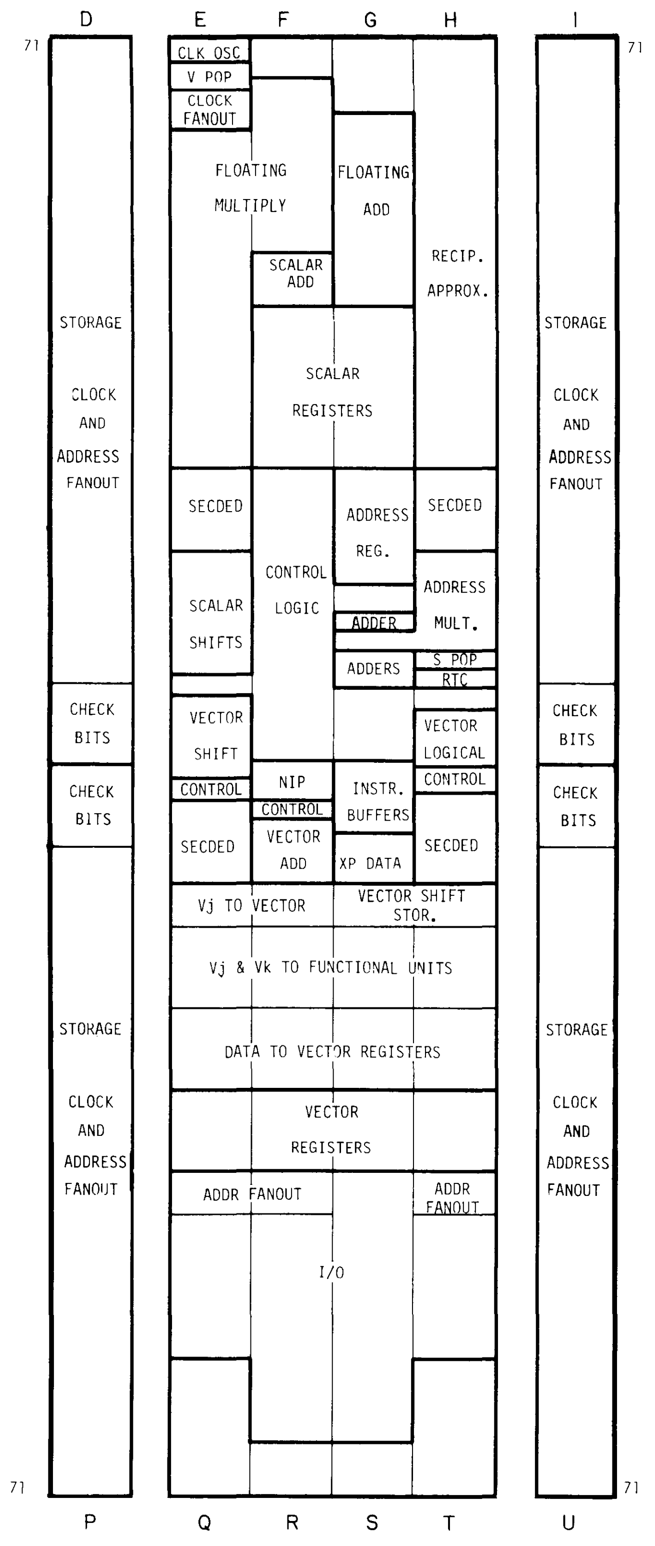


Figure B-2. General chassis layout

Two supply voltages are used for each module: -5.2 volts for IC power; -2.0 volts for line termination in the CPU; and -5.2 volts and +5 volts for the memory modules.

Each module has up to 96 or 192 pin pairs for interconnecting to other modules. All interconnections are via twisted pair wire. The average use of pins is approximately 60 percent.

Each module has up to 144 or 288 available test points used for trouble shooting. Test points are driven by circuits that do not drive other loads.

CLOCK

All timing within the mainframe is controlled by a single-phase synchronous clock network. All of the lines that carry the clock signal from the central clock source to the individual modules of the mainframe are of uniform length so that the leading edge of a clock signal arrives at all parts of the mainframe cabinet at the same time. A pulse is formed on each module.

References to clock periods in this manual are often given in the form CP *n* 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

Sixteen power supplies are used for a CRAY-1 M mainframe. There are 8 -5.2 volt power supplies and 4 -2.0 volt power supplies and 4 +5 volt supplies. A logic column uses one -5.2 volt power supply and one -2.0 volt power supply. A memory column uses two -5.2 volt power supplies and two +5.0 volt power supplies. The CPU power supply design assumes a constant load. The CPU power supplies do not have internal regulation but depend on a motor-generator to regulate incoming power. Power supplies use a 12-phase transformer, silicon diodes, balancing coil, and a filter choke to supply low ripple DC voltages. The power supplies are mounted on a refrigerant-22 cooled heat sink. Power is distributed via bus bars to the load.

A memory power supply is a switching supply that does not assume constant load. A memory power supply has internal regulation, and its building blocks include a rectifier, an inverter, a transformer, a secondary rectifier, an output filter, and control circuitry. A memory power supply is air-cooled, and power is distributed via bus bars to the load.

COOLING

Modules in the mainframe 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 drilled refrigerant tube.

SOFTWARE CONSIDERATIONS C

References to software in this publication are limited to those features of the mainframe 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 can also describe an operating system task that does not execute in monitor mode. A user program can be a machine language program such as a FORTRAN compiler or it can 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 Cray Research supplied operating system and organization of storage, which is a function of the operating system, are described in the CRAY-OS Version 1 Reference Manual, publication SR-0011.

SYSTEM OPERATION

System operation begins at system deadstart. Deadstart is that sequence of operations required to start a program running in the computer after normal operation has been interrupted.

The deadstart sequence is initiated from the I/O Subsystem. The sequence is described in detail in section 4. During the deadstart sequence, a program containing an exchange package is loaded at absolute address 0 in the Central Memory. A signal from the I/O Subsystem causes the CRAY-1 M mainframe to begin execution of the program pointed to by the exchange package.

FLOATING-POINT RANGE ERRORS

Detecting a 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. Through an instruction 0022, the programmer has the capability to clear the Floating-point Mode flag so that results going out of range are not interrupted. This is especially useful for the vector merge instruction used in subroutines such as TANGENT, where some results can be known to go out of range. At the end of the code sequence, the programmer normally resets the Floating-point Mode flag through an instruction 0021.

In code sequences that generate out-of-range values and the errors are true error conditions and the flag is not set, the programmer must check the results to determine if an out-of-range condition occurred. Normally, the scan can be done before the operation starts.

If a programmer clears the Floating-point Mode flag and wants it to remain cleared, the software Floating-point Mode flag must also be cleared before any library routines are called or the Floating-point Mode flag can set when the library routine exits.

CPU INSTRUCTION SUMMARY

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
000*xxx*	ERR	7-7	-	Error exit
†000*ijk*	ERR *exp*	7-7	-	Error exit
††0010*jk*	CA,A*j* A*k*	7-8	-	Set the channel (A*j*) current address to (A*k*) and begin the I/O sequence
††0011*jk*	CL,A*j* A*k*	7-8	-	Set the channel (A*j*) limit address to (A*k*)
††0012*jx*	CI,A*j*	7-8	-	Clear channel (A*j*) interrupt flag
††0013*jx*	XA A*j*	7-8	-	Enter XA register with (A*j*)
††0014*j*0	RT S*j*	7-10	-	Enter RTC register with (S*j*)
††0014*j*4	PCI S*j*	7-10	-	Enter interval register with (S*j*)
††0014*x*5	CCI	7-10	-	Clear PCI request
††0014*x*6	ECI	7-10	-	Enable PCI request
††0014*x*7	DCI	7-10	-	Disable PCI request
0020*xk*	VL A*k*	7-12	-	Transmit (A*k*) to VL register
†0020*x*0	VL 1	7-12	-	Transmit 1 to VL register
0021*xx*	EFI	7-13	-	Enable interrupt on floating-point error

† Special syntax form
†† Privileged to monitor mode

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
0022*xx*	DFI	7-13	-	Disable interrupt on floating-point error
003*xjx*	VM S*j*	7-14	-	Transmit (S*j*) to VM register
†003*x*0*x*	VM 0	7-14	-	Clear VM register
004*xxx*	EX	7-15	-	Normal exit
†004*ijk*	EX *exp*	7-15	-	Normal exit
005*xjk*	J B*jk*	7-16	-	Jump to (B*jk*)
006*ijkm*	J *exp*	7-17	-	Jump to *exp*
007*ijkm*	R *exp*	7-18	-	Return jump to *exp*; set B00 to P.
010*ijkm*	JAZ *exp*	7-19	-	Branch to *exp* if (A0)=0
011*ijkm*	JAN *exp*	7-19	-	Branch to *exp* if (A0)≠0
012*ijkm*	JAP *exp*	7-19	-	Branch to *exp* if (A0)≥0
013*ijkm*	JAM *exp*	7-19	-	Branch to *exp* if (A0)<0
014*ijkm*	JSZ *exp*	7-21	-	Branch to *exp* if (S0)=0
015*ijkm*	JSN *exp*	7-21	-	Branch to *exp* if (S0)≠0
016*ijkm*	JSP *exp*	7-21	-	Branch to *exp* if (S0)≥0
017*ijkm*	JSM *exp*	7-21	-	Branch to *exp* if (S0)<0
020*ijkm*	A*i* *exp*	7-23	-	Transmit *exp*=*jkm* to A*i*
021*ijkm*	A*i* *exp*	7-23	-	Transmit *exp*=ones complement of *jkm* to A*i*
022*ijk*	A*i* *exp*	7-24	-	Transmit *exp*=*jk* to A*i*
023*ijx*	A*i* S*j*	7-25	-	Transmit (S*j*) to A*i*
024*ijk*	A*i* B*jk*	7-26	-	Transmit (B*jk*) to A*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
025*ijk*	B*jk* A*i*	7-26	-	Transmit (A*i*) to B*jk*
026*ij*0	A*i* PS*j*	7-27	Pop/LZ	Population count of (S*j*) to A*i*
026*ij*1	A*i* QS*j*	7-27	Pop/LZ	Population count parity of (S*j*) to A*i*
027*ijx*	A*i* ZS*j*	7-28	Pop/LZ	Leading zero count of (S*j*) to A*i*
030*ijk*	A*i* A*j*+A*k*	7-29	A Int Add	Integer sum of (A*j*) and (A*k*) to A*i*
†030*i*0*k*	A*i* A*k*	7-29	A Int Add	Transmit (A*k*) to A*i*
†030*ij*0	A*i* A*j*+1	7-29	A Int Add	Integer sum of (A*j*) and 1 to A*i*
031*ijk*	A*i* A*j*-A*k*	7-29	A Int Add	Integer difference of (A*j*) less (A*k*) to A*i*
†031*i*00	A*i* -1	7-29	A Int Add	Transmit -1 to A*i*
†031*i*0*k*	A*i* -A*k*	7-29	A Int Add	Transmit the negative of (A*k*) to A*i*
†031*ij*0	A*i* A*j*-1	7-29	A Int Add	Integer difference of (A*j*) less 1 to A*i*
032*ijk*	A*i* A*j**A*k*	7-30	A Int Mult	Integer product of (A*j*) and (A*k*) to A*i*
033*i*0*x*	A*i* CI	7-31	-	Channel number to A*i* (*j*=0)
033*ij*0	A*i* CA,A*j*	7-31	-	Address of channel (A*j*) to A*i* (*j*≠0; *k*=0)
033*ij*1	A*i* CE,A*j*	7-31	-	Error flag of channel (A*j*) to A*i* (*j*≠0; *k*=1)
034*ijk*	B*jk*,A*i* ,A0	7-32	Memory	Read (A*i*) words to B register *jk* from (A0)

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
†034*ijk*	B*jk*,A*i* 0,A0	7-32	Memory	Read (A*i*) words to B register *jk* from (A0)
035*ijk*	,A0 B*jk*,A*i*	7-32	Memory	Store (A*i*) words at B register *jk* to (A0)
†035*ijk*	0,A0 B*jk*,A*i*	7-32	Memory	Store (A*i*) words at B register *jk* to (A0)
036*ijk*	T*jk*,A*i* ,A0	7-32	Memory	Read (A*i*) words to T register *jk* from (A0)
†036*ijk*	T*jk*,A*i* 0,A0	7-32	Memory	Read (A*i*) words to T register *jk* from (A0)
037*ijk*	,A0 T*jk*,A*i*	7-32	Memory	Store (A*i*) words at T register *jk* to (A0)
†037*ijk*	0,A0 T*jk*,A*i*	7-32	Memory	Store (A*i*) words at T register *jk* to (A0)
040*ijkm*	S*i* *exp*	7-34	-	Transmit *jkm* to S*i*
041*ijkm*	S*i* *exp*	7-34	-	Transmit *exp*=ones complement of *jkm* to S*i*
042*ijk*	S*i* <*exp*	7-35	S Logical	Form ones mask *exp* bits in S*i* from the right; *jk* field gets 64-*exp*.
†042*ijk*	S*i* #>*exp*	7-35	S Logical	Form zeros mask *exp* bits in S*i* from the left; *jk* field gets *exp*.
†042*i*77	S*i* 1	7-35	S Logical	Enter 1 into S*i*
†042*i*00	S*i* -1	7-35	S Logical	Enter -1 into S*i*
043*ijk*	S*i* >*exp*	7-35	S Logical	Form ones mask *exp* bits in S*i* from the left; *jk* field gets *exp*.
†043*ijk*	S*i* #<*exp*	7-35	S Logical	Form zeros mask *exp* bits in S*i* from the right; *jk* field gets 64-*exp*.
†043*i*00	S*i* 0	7-35	S Logical	Clear S*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
044*ijk*	S*i* S*j*&S*k*	7-36	S Logical	Logical product of (S*j*) and (S*k*) to S*i*
†044*ij*0	S*i* S*j*&SB	7-36	S Logical	Sign bit of (S*j*) to S*i*
†044*ij*0	S*i* SB&S*j*	7-36	S Logical	Sign bit of (S*j*) to S*i* (*j*≠0)
045*ijk*	S*i* #S*k*&S*j*	7-36	S Logical	Logical product of (S*j*) and ones complement of (S*k*) to S*i*
†045*ij*0	S*i* #SB&S*j*	7-36	S Logical	(S*j*) with sign bit cleared to S*i*
046*ijk*	S*i* S*j*\S*k*	7-36	S Logical	Logical difference of (S*j*) and (S*k*) to S*i*
†046*ij*0	S*i* S*j*\SB	7-36	S Logical	Toggle sign bit of S*j*, then enter into S*i*
†046*ij*0	S*i* SB\S*j*	7-36	S Logical	Toggle sign bit of S*j*, then enter into S*i* (*j*≠0)
047*ijk*	S*i* #S*j*\S*k*	7-36	S Logical	Logical equivalence of (S*k*) and (S*j*) to S*i*
†047*i*0*k*	S*i* #S*k*	7-36	S Logical	Transmit ones complement of (S*k*) to S*i*
†047*ij*0	S*i* #S*j*\SB	7-36	S Logical	Logical equivalence of (S*j*) and sign bit to S*i*
†047*ij*0	S*i* #SB\S*j*	7-36	S Logical	Logical equivalence of (S*j*) and sign bit to S*i* (*j*≠0)
†047*i*00	S*i* #SB	7-36	S Logical	Enter ones complement of sign bit into S*i*
050*ijk*	S*i* S*j*!S*i*&S*k*	7-36	S Logical	Logical product of (S*j*) and (S*k*) complement ORed with logical product of (S*j*) and (S*k*) to S*i*
†050*ij*0	S*i* S*j*!S*i*&SB	7-36	S Logical	Scalar merge of (S*i*) and sign bit of (S*j*) to S*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
051*ijk*	S*i* S*j*!S*k*	7-36	S Logical	Logical sum of (S*j*) and (S*k*) to S*i*
†051*i*0*k*	S*i* S*k*	7-36	S Logical	Transmit (S*k*) to S*i*
†051*ij*0	S*i* S*j*!SB	7-36	S Logical	Logical sum of (S*j*) and sign bit to S*i*
†051*ij*0	S*i* SB!S*j*	7-36	S Logical	Logical sum of (S*j*) and sign bit to S*i* (*j*≠0)
†051*i*00	S*i* SB	7-36	S Logical	Enter sign bit into S*i*
052*ijk*	S0 S*i*<*exp*	7-40	S Shift	Shift (S*i*) left *exp*=*jk* places to S0
053*ijk*	S0 S*i*>*exp*	7-40	S Shift	Shift (S*i*) right *exp*=64-*jk* places to S0
054*ijk*	S*i* S*i*<*exp*	7-40	S Shift	Shift (S*i*) left *exp*=*jk* places
055*ijk*	S*i* S*i*>*exp*	7-40	S Shift	Shift (S*i*) right *exp*=64-*jk* places
056*ijk*	S*i* S*i*,S*j*<A*k*	7-41	S Shift	Shift (S*i* and S*j*) left (A*k*) places to S*i*
†056*ij*0	S*i* S*i*,S*j*<1	7-41	S Shift	Shift (S*i* and S*j*) left one place to S*i*
†056*i*0*k*	S*i* S*i*<A*k*	7-41	S Shift	Shift (S*i*) left (A*k*) places to S*i*
057*ijk*	S*i* S*j*,S*i*>A*k*	7-41	S Shift	Shift (S*j* and S*i*) right (A*k*) places to S*i*
†057*ij*0	S*i* S*j*,S*i*>1	7-41	S Shift	Shift (S*j* and S*i*) right one place to S*i*
†057*i*0*k*	S*i* S*i*>A*k*	7-41	S Shift	Shift (S*i*) right (A*k*) places to S*i*
060*ijk*	S*i* S*j*+S*k*	7-43	S Int Add	Integer sum of (S*j*) and (S*k*) to S*i*

† Special syntax form

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
061*ijk*	S*i*	S*j*-S*k*	7-43	S Int Add	Integer difference of (S*j*) and (S*k*) to S*i*
†061*i*0*k*	S*i*	-S*k*	7-43	S Int Add	Transmit negative of (S*k*) to S*i*
062*ijk*	S*i*	S*j*+FS*k*	7-44	Fp Add	Floating-point sum of (S*j*) and (S*k*) to S*i*
†062*i*0*k*	S*i*	+FS*k*	7-44	Fp Add	Normalize (S*k*) to S*i*
063*ijk*	S*i*	S*j*-FS*k*	7-44	Fp Add	Floating-point difference of (S*j*) and (S*k*) to S*i*
†063*i*0*k*	S*i*	-FS*k*	7-44	Fp Add	Transmit normalized negative of (S*k*) to S*i*
064*ijk*	S*i*	S*j**FS*k*	7-46	Fp Mult	Floating-point product of (S*j*) and (S*k*) to S*i*
065*ijk*	S*i*	S*j**HS*k*	7-46	Fp Mult	Half-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*
066*ijk*	S*i*	S*j**RS*k*	7-46	Fp Mult	Full-precision rounded floating-point product of (S*j*) and (S*k*) to S*i*
067*ijk*	S*i*	S*j**IS*k*	7-46	Fp Mult	2-Floating-point product of (S*j*) and (S*k*) to S*i*
070*ijx*	S*i*	/HS*j*	7-48	Fp Rcpl	Floating-point reciprocal approximation of (S*j*) to S*i*
071*i*0*k*	S*i*	A*k*	7-49	-	Transmit (A*k*) to S*i* with no sign extension
071*i*1*k*	S*i*	+A*k*	7-49	-	Transmit (A*k*) to S*i* with sign extension
071*i*2*k*	S*i*	+FA*k*	7-49	-	Transmit (A*k*) to S*i* as unnormalized floating-point number

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
071*i*3*x*	S*i* 0.6	7-49	-	Transmit constant 0.75*2**48 to S*i*
071*i*4*x*	S*i* 0.4	7-49	-	Transmit constant 0.5 to S*i*
071*i*5*x*	S*i* 1.	7-49	-	Transmit constant 1.0 to S*i*
071*i*6*x*	S*i* 2.	7-49	-	Transmit constant 2.0 to S*i*
071*i*7*x*	S*i* 4.	7-49	-	Transmit constant 4.0 to S*i*
072*ixx*	S*i* RT	7-51	-	Transmit (RTC) to S*i*
073*ixx*	S*i* VM	7-51	-	Transmit (VM) to S*i*
074*ijk*	S*i* T*jk*	7-51	-	Transmit (T*jk*) to S*i*
075*ijk*	T*jk* S*i*	7-51	-	Transmit (S*i*) to T*jk*
076*ijk*	S*i* V*j*,A*k*	7-52	-	Transmit (V*j*, element (A*k*)) to S*i*
077*ijk*	V*i*,A*k* S*j*	7-52	-	Transmit (S*j*) to V*i* element (A*k*)
†077*i*0*k*	V*i*,A*k* 0	7-52	-	Clear V*i* element (A*k*)
10*hijkm*	A*i* *exp*,A*h*	7-53	Memory	Read from ((A*h*)+*exp*) to A*i* (A0=0)
†100*ijkm*	A*i* *exp*,0	7-53	Memory	Read from (*exp*) to A*i*
†100*ijkm*	A*i* *exp*,	7-53	Memory	Read from (*exp*) to A*i*
†10*hi*00 0	A*i* ,A*h*	7-53	Memory	Read from (A*h*) to A*i*
11*hijkm*	*exp*,A*h* A*i*	7-53	Memory	Store (A*i*) to (A*h*)+*exp* (A0=0)
†110*ijkm*	*exp*,0 A*i*	7-53	Memory	Store (A*i*) to *exp*
†110*ijkm*	*exp*, A*i*	7-53	Memory	Store (A*i*) to *exp*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
†11*hi*00 0	,A*h* A*i*	7-53	Memory	Store (A*i*) to (A*h*)
12*hijkm*	S*i* *exp*,A*h*	7-53	Memory	Read from ((A*h*)+*exp*) to S*i* (A0=0)
†120*ijkm*	S*i* *exp*,0	7-53	Memory	Read from (*exp*) to S*i*
†120*ijkm*	S*i* *exp*,	7-53	Memory	Read from (*exp*) to S*i*
†12*hi*00 0	S*i* ,A*h*	7-53	Memory	Read from (A*h*) to S*i*
13*hijkm*	*exp*,A*h* S*i*	7-53	Memory	Store (S*i*) to (A*h*)+*exp* (A0=0)
†130*ijkm*	*exp*,0 S*i*	7-53	Memory	Store (S*i*) to *exp*
†130*ijkm*	*exp*, S*i*	7-53	Memory	Store (S*i*) to *exp*
†13*hi*00 0	,A*h* S*i*	7-53	Memory	Store (S*i*) to (A*h*)
140*ijk*	V*i* S*j*&V*k*	7-55	V Logical	Logical products of (S*j*) and (V*k*) to V*i*
141*ijk*	V*i* V*j*&V*k*	7-55	V Logical	Logical products of (V*j*) and (V*k*) to V*i*
142*ijk*	V*i* S*j*!V*k*	7-55	V Logical	Logical sums of (S*j*) and (V*k*) to V*i*
†142*i*0*k*	V*i* V*k*	7-55	V Logical	Transmit (V*k*) to V*i*
143*ijk*	V*i* V*j*!V*k*	7-55	V Logical	Logical sums of (V*j*) and (V*k*) to V*i*
144*ijk*	V*i* S*j*\V*k*	7-55	V Logical	Logical differences of (S*j*) and (V*k*) to V*i*
145*ijk*	V*i* V*j*\V*k*	7-55	V Logical	Logical differences of (V*j*) and (V*k*) to V*i*
†145*iii*	V*i* 0	7-55	V Logical	Clear V*i*
146*ijk*	V*i* S*j*!V*k*&VM	7-55	V Logical	Transmit (S*j*) if VM bit=1; (V*k*) if VM bit=0 to V*i*.

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
†146*i*0*k*	V*i* #VM&V*k*	7-55	V Logical	Vector merge of (V*k*) and 0 to V*i*
147*ijk*	V*i* V*j*!V*k*&VM	7-55	V Logical	Transmit (V*j*) if VM bit=1; (V*k*) if VM bit=0 to V*i*.
150*ijk*	V*i* V*j*<A*k*	7-59	V Shift	Shift (V*j*) left (A*k*) places to V*i*
†150*ij*0	V*i* V*j*<1	7-59	V Shift	Shift (V*j*) left one place to V*i*
151*ijk*	V*i* V*j*>A*k*	7-59	V Shift	Shift (V*j*) right (A*k*) places to V*i*
†151*ij*0	V*i* V*j*>1	7-59	V Shift	Shift (V*j*) right one place to V*i*
152*ijk*	V*i* V*j*,V*j*<A*k*	7-60	V Shift	Double shift (V*j*) left (A*k*) places to V*i*
†152*ij*0	V*i* V*j*,V*j*<1	7-60	V Shift	Double shift (V*j*) left one place to V*i*
153*ijk*	V*i* V*j*,V*j*>A*k*	7-60	V Shift	Double shift (V*j*) right (A*k*) places to V*i*
†153*ij*0	V*i* V*j*,V*j*>1	7-60	V Shift	Double shift (V*j*) right one place to V*i*
154*ijk*	V*i* S*j*+V*k*	7-65	V Int Add	Integer sums of (S*j*) and (V*k*) to V*i*
155*ijk*	V*i* V*j*+V*k*	7-65	V Int Add	Integer sums of (V*j*) and (V*k*) to V*i*
156*ijk*	V*i* S*j*-V*k*	7-65	V Int Add	Integer differences of (S*j*) and (V*k*) to V*i*
†156*i*0*k*	V*i* -V*k*	7-65	V Int Add	Transmit negative of (V*k*) to V*i*
157*ijk*	V*i* V*j*-V*k*	7-65	V Int Add	Integer differences of (V*j*) and (V*k*) to V*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
160*ijk*	V*i* S*j**FV*k*	7-67	Fp Mult	Floating-point products of (S*j*) and (V*k*) to V*i*
161*ijk*	V*i* V*j**FV*k*	7-67	Fp Mult	Floating-point products of (V*j*) and (V*k*) to V*i*
162*ijk*	V*i* S*j**HV*k*	7-67	Fp Mult	Half-precision rounded floating-point products of (S*j*) and (V*k*) to V*i*
163*ijk*	V*i* V*j**HV*k*	7-67	Fp Mult	Half-precision rounded floating-point products of (V*j*) and (V*k*) to V*i*
164*ijk*	V*i* S*j**RV*k*	7-67	Fp Mult	Rounded floating-point products of (S*j*) and (V*k*) to V*i*
165*ijk*	V*i* V*j**RV*k*	7-67	Fp Mult	Rounded floating-point products of (V*j*) and (V*k*) to V*i*
166*ijk*	V*i* S*j**IV*k*	7-67	Fp Mult	2-floating-point products of (S*j*) and (V*k*) to V*i*
167*ijk*	V*i* V*j**IV*k*	7-67	Fp Mult	2-floating-point products of (V*j*) and (V*k*) to V*i*
170*ijk*	V*i* S*j*+FV*k*	7-70	Fp Add	Floating-point sums of (S*j*) and (V*k*) to V*i*
†170*i*0*k*	V*i* +FV*k*	7-70	Fp Add	Normalize (V*k*) to V*i*
171*ijk*	V*i* V*j*+FV*k*	7-70	Fp Add	Floating-point sums of (V*j*) and (V*k*) to V*i*
172*ijk*	V*i* S*j*-FV*k*	7-70	Fp Add	Floating-point differences of (S*j*) and (V*k*) to V*i*
†172*i*0*k*	V*i* -FV*k*	7-70	Fp Add	Transmit normalized negatives of (V*k*) to V*i*
173*ijk*	V*i* V*j*-FV*k*	7-70	Fp Add	Floating-point differences of (V*j*) and (V*k*) to V*i*

† Special syntax form

CRAY-1	CAL	PAGE	UNIT	DESCRIPTION
174*ij*0	V*i* /HV*j*	7-72	Fp Rcpl	Floating-point reciprocal approximations of (V*j*) to V*i*
174*ij*1	V*i* PV*j*	7-73	V Pop	Population counts of (V*j*) to V*i*
174*ij*2	V*i* QV*j*	7-73	V Pop	Population count parities of (V*j*) to V*i*
175*xj*0	VM V*j*,Z	7-74	V Logical	VM=1 where (V*j*)=0
175*xj*1	VM V*j*,N	7-74	V Logical	VM=1 where (V*j*)≠0
175*xj*2	VM V*j*,P	7-74	V Logical	VM=1 where (V*j*) positive
175*xj*3	VM V*j*,M	7-74	V Logical	VM=1 where (V*j*) negative
176*ixk*	V*i* ,A0,A*k*	7-76	Memory	Read (VL) words to V*i* from (A0) incremented by (A*k*)
†176*ix*0	V*i* ,A0,1	7-76	Memory	Read (VL) words to V*i* from (A0) incremented by 1
177*xjk*	,A0,A*k* V*j*	7-76	Memory	Store (VL) words from V*j* to (A0) incremented by (A*k*)
†177*xj*0	,A0,1 V*j*	7-76	Memory	Store (VL) words from V*j* to (A0) incremented by 1

Legend:

A	Address
Fp	Floating-point
Int	Integer
Mult	Multiply
Pop	Population/Parity
Pop/LZ	Population/Leading Zero
Rcpl	Reciprocal Approximation
S	Scalar
V	Vector

† Special syntax form

6 MBYTES PER SECOND CHANNEL DESCRIPTIONS

INTRODUCTION

Each input or output 6 Mbytes per second channel directly accesses 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 (see section 6).

Each input or output channel has a data channel (4 parity bits, 16 data bits, and 3 control lines), a 64-bit assembly or disassembly register, a channel Current Address (CA) register, and a channel Limit Address (CL) register.

Three control signals (Ready, Resume, and Disconnect) coordinate the transfer of parcels over the channels. In addition to the three control signals, the output channel of the pair has a Master Clear line.

This appendix describes the signal sequence of a 6 Mytes per second input channel and an output channel for 16-bit asynchronous channels and for 16-bit high-speed asynchronous channels.

16-BIT ASYNCHRONOUS CHANNELS

The 16-bit asynchronous input channels and output channels are described below.

INPUT CHANNELS

A general view of an input signal sequence is illustrated in table E-1. The data bits, parity bits, and each signal in the sequence are described in the following paragraphs.

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 Central Memory. The data bits 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 signal is received by the external device. Normally, data is sent coincidentally with the Ready signal and is held until the subsequent Ready signal.

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 follow:

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

Parity bits are sent from the external device to Central Memory at the same time as data bits and are held stable in the same way as the data bits.

Ready

The Ready signal sent to Central Memory indicates a parcel of data is being sent to the Central Memory input channel and can be sampled. A Ready signal is a pulse 50 ± 10 nanoseconds wide (at 50% voltage points). The leading edge of the Ready signal at Central Memory begins the timing for sampling the data bits.

Resume

The Resume signal is sent from Central Memory to the external device showing the parcel was received and Central Memory is ready for the next data transmission. A Resume signal is a pulse 50 ± 3 nanoseconds wide (at 50% voltage points).

Table E-1. 16-bit asynchronous input channel signal exchange

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

Disconnect

The Disconnect signal is sent from the external device to Central Memory and indicates transmission from the external device is complete. The Disconnect signal is sent after the Resume signal is received for the last Ready signal. A Disconnect signal is a pulse 50 ±10 nanoseconds wide (at 50% voltage points).

Channel Master Clear

The Channel Master Clear signal is programmed (see description of Programmed Master Clear in section 6) or results from a Clear I/O signal.

OUTPUT CHANNELS

A general view of an output signal sequence is illustrated in table E-2. The data bits, parity bits, and each signal in the sequence are described below.

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 Central Memory to an external device. The data bits are sent concurrently within 5 nanoseconds of the leading edge of the Ready signal. Data bit signals remain steady on the lines until the Resume signal 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 follow:

Parity bit	Data bits
0	$2^0 - 2^3$
1	$2^4 - 2^7$
2	$2^8 - 2^{11}$
3	$2^{12} - 2^{15}$

Parity bits are sent from Central Memory to the external device at the same time as the data bits and are held stable in the same way as the data bits.

Ready

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

Table E-2. 16-bit asynchronous output channel signal exchange

Central Memory	Channel	External Equipment
1. Activate channel (set CL and CA).		
2. Read word from memory and advance current address.		
3. Data 2^{63} - 2^{48} with Ready	$\longrightarrow$	
4.	$\longleftarrow$	Resume
5. Data 2^{47} - 2^{32} with Ready	$\longrightarrow$	
6.	$\longleftarrow$	Resume
7. Data 2^{31} - 2^{16} with Ready	$\longrightarrow$	
8.	$\longleftarrow$	Resume
9. Data 2^{15} - 2^{0} with Ready	$\longrightarrow$	
10.	$\longleftarrow$	Resume
11. If (CA) $\neq$ (CL), go to 2.		
12. Disconnect	$\longrightarrow$	
13. Set interrupt and deactivate channel.		

Resume

The Resume signal is sent from the external device to Central Memory showing the parcel was received and the external device is ready for the next parcel transmission. A Resume signal is a pulse 50 $\pm$10 nanoseconds wide (at 50% voltage points).

Disconnect

The Disconnect signal is sent from Central Memory to the external device and indicates transmission from Central Memory is complete. The Disconnect signal is sent after Central Memory receives the Resume signal from the last Ready signal. A Disconnect signal is a pulse 50 ± 3 nanoseconds wide (at 50% voltage points).

16-BIT HIGH-SPEED ASYNCHRONOUS CHANNELS

The 16-bit high-speed asynchronous input channels and output channels are described below.

INPUT CHANNELS

A general view of an input signal sequence is illustrated in table E-3. The data bits, parity bits, and each signal in the sequence are described below.

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 Central Memory. The data lines must be stable no later than 80 nanoseconds after the leading edge of the associated Ready signal and must be held stable until at least 120 nanoseconds after the leading edge of the same Ready signal. Note that if the device is transmitting at the maximum allowable rate, it is normal for a data parcel to overlap the subsequent Ready signal. Typically, data is transmitted 50 nanoseconds after the leading edge of a Ready signal and held until 50 nanoseconds after the leading edge of the following Ready signal.

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 follow:

Parity bit	Data bits
0	$2^0 - 2^3$
1	$2^4 - 2^7$
2	$2^8 - 2^{11}$
3	$2^{12} - 2^{15}$

Table E-3. 16-bit high-speed asynchronous input channel signal exchange

Central Memory	Channel	External Equipment
1. Activate channel (set CL and CA).		
2. Resume	$\longrightarrow$	
3. Resume	$\longrightarrow$	
4. Resume	$\longrightarrow$	
5. Resume	$\longrightarrow$	If done, go to 11.
6.	$\longleftarrow$	Data 2^{63} - 2^{48} with Ready
7.	$\longleftarrow$	Data 2^{47} - 2^{32} with Ready
8.	$\longleftarrow$	Data 2^{31} - 2^{16} with Ready
9.	$\longleftarrow$	Data 2^{15} - 2^{0} with Ready
10. Write word to memory and advance current address; go to 2.		
11.	$\longleftarrow$	Disconnect
12. Set interrupt and deactivate channel.		

Parity bits are sent from the external device to Central Memory at the same time as the data bits and are held stable in the same way as data bits.

Ready

The Ready signal sent to Central Memory indicates data is being sent to the Central Memory input channel and can be sampled. A Ready signal is a pulse 50 ± 10 nanoseconds wide (at 50% voltage points) sent in groups of four. The leading edge of a Ready signal at Central Memory begins timing for sampling of data bits.

The first Ready pulse of a group can 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

The Resume signal is sent to the external device showing that Central Memory is ready for the next data transmission. A Resume signal is a pulse 50 ± 3 nanoseconds wide (at 50% voltage points) sent in groups of four.

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

Disconnect

The Disconnect signal is sent from the external device to Central Memory and indicates transmission from the external device is complete. The Disconnect signal is sent after the last Ready signal. An input Disconnect signal must be transmitted no earlier than 20 nanoseconds after the leading edge of the final Ready signal. A Disconnect signal is a pulse 50 ± 10 nanoseconds wide (at 50% voltage points).

OUTPUT CHANNELS

A general view of an output signal sequence is illustrated in table E-4. The data bits, parity bits, and each signal in the sequence are described in the following paragraphs.

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 Central Memory to an external device. The data bits are sent concurrently within 5 nanoseconds of the leading edge of the Ready signal. Data bits remain steady on the lines until the next parcel is sent or until the Resume signal 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. Parity bits are set or cleared to give the bit group odd parity.

Table E-4. 16-bit high-speed asynchronous output channel signal exchange

Central Memory	Channel	External Equipment
1. Activate channel (set CL and CA).		
2. Read word from memory and advance current address.		
3. Data $2^{63} - 2^{48}$ with Ready	$\longrightarrow$	
4. Data $2^{47} - 2^{32}$ with Ready	$\longrightarrow$	
5. Data $2^{31} - 2^{16}$ with Ready	$\longrightarrow$	
6. Data $2^{15} - 2^{0}$ with Ready (with Disconnect if this is last word)	$\longrightarrow$	
7.	$\longleftarrow$	Resume
8. If (CA) $\neq$ (CL), go to 2.		
9. Set interrupt and deactivate channel.		

Bit assignments follow:

Parity bit	Data bits
0	$2^{0} - 2^{3}$
1	$2^{4} - 2^{7}$
2	$2^{8} - 2^{11}$
3	$2^{12} - 2^{15}$

Parity bits are sent from Central Memory to the external device at the same time as the data bits and are held stable in the same way as the data bits.

Channel Master Clear

The Channel Master Clear is programmed (see description of Programmed Master Clear in section 6) or results from a Clear I/O signal. The Master Clear signal is used by the external devices for control purposes or is ignored.

Ready

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

Resume

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

Disconnect

The Disconnect signal is sent from Central Memory to the external device and indicates the transmission from Central Memory is complete. The Disconnect signal is sent with the last Ready signal ±3 nanoseconds. A Disconnect signal is a pulse 50 ±3 nanoseconds wide (at 50% voltage points).

INDEX

INDEX

READERS COMMENT FORM

CRAY-1 M Series Mainframe Reference Manual HR-0064

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

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

FOLD

STAPLE

READERS COMMENT FORM

CRAY-1 M Series Mainframe Reference Manual HR-0064

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.

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

Attention:
PUBLICATIONS

FOLD

STAPLE

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444